

WPILib LabVIEW Math Control / Trajectory Library Reference

Version 2024.1

Table of Contents

Introduction.....	6
Copyright.....	7
Description and Conventions.....	8
Function Naming.....	8
"Execute" Functions.....	8
Function Menu.....	9
Function Snippets.....	13
Function Help.....	14
Function Examples.....	15
Utility Programs.....	16
Create/Edit Trajectory.....	16
Create/Edit PathFinder Differential Drive Trajectory.....	19
Create/Edit PathFinder Swerve Drive Trajectory.....	20
Convert Trajectory JSON file to CSV.....	21
Convert Trajectory CSV file to JSON.....	21
Romi Robot Simulator.....	21
Differential Drive Simulator.....	22
Trajectory Creation.....	24
Reporting Issues and Suggesting Enhancements.....	25
Contributing.....	26
Function Groups.....	27
AnalogDelay.....	28
AngleStats.....	29
AprilTag.....	33
AprilTagFieldLayout.....	35
AprilTagPoseEstimate.....	41
ArmFF.....	43
AutoHelper.....	49
BangBang.....	51
BatterySim.....	57
BoolCmd.....	59
BumplessTransfer.....	61
CallbackHelp.....	62
CentripetalAccelConstraint.....	65
ChassisSpeeds.....	67
CompVisionUtil.....	70
ConstrainedState.....	75
Constraint.....	77
ControllerUtil.....	78
Conv.....	79
CoordAxis.....	84
CoordSystem.....	88
CubicHermiteSpline.....	93
DCMotor.....	95

DcMotorSim.....	105
Debouncer.....	107
DiffDriveKinematicsConstraint.....	110
DiffDrivePoseEst.....	112
DiffDrivePoseEst2.....	121
DiffDriveTrainSim.....	131
DiffDriveVoltageConstraint.....	151
DiffDrvAccelLimit.....	153
DiffKinematics.....	155
DiffOdometry.....	158
DiffWheel.....	160
DigSeqLogic.....	161
Discretization.....	165
DoubleSolenoid.....	170
DrumSequence.....	171
ElevatorSim.....	173
ElevFF.....	182
EllipRegionConstraint.....	187
ExtendedKalmanFilter.....	190
FieldDisp.....	198
FlyWheelSim.....	204
FunctionGenerator.....	209
FunctionGeneratorMatrix.....	212
HolDrvCtrl.....	214
ImplModelFollow.....	225
JerkConstraint.....	229
KalmanFilter.....	231
KalmanFilterLatencyComp.....	237
LeadLag.....	242
LinearFilter.....	243
LinearPlntInvFF.....	257
LinearQuadraticRegulator.....	263
LinearSystem.....	272
LinearSystemId.....	278
LinearSystemLoop.....	290
LinearSystemSim.....	312
LTVDiffDriveCtrl.....	319
LTVUnicycleCtrl.....	328
MatBuilder.....	335
MathUtil.....	337
Matrix.....	341
MatrixHelper.....	353
MaxVelocityConstraint.....	355
MecaDriveKinematicsConstraint.....	357
MecaDrivePoseEst.....	359
MecaKinematics.....	369
MecaOdometry.....	374
MecaWheel.....	379
MecaWheelPos.....	381

MedianFilter.....	383
MerweScSigPts.....	386
NetworkUDP.....	392
numCmd.....	395
NumIntegrate.....	398
NumJacobian.....	410
PathfinderUtil.....	412
PIDAutoTune.....	414
PIDController.....	420
PosCtrl.....	445
Pose2d.....	447
Pose3d.....	456
PoseWithCurve.....	466
ProfiledPIDController.....	467
Quaternion.....	484
QuinticHermiteSpline.....	490
Ramsete.....	492
RectRegionConstraint.....	504
Riccati.....	507
Rotation2d.....	513
Rotation3D.....	522
RungeKuttaTimeVarying.....	531
SimpleMatrix.....	532
SimpleMotorFF.....	533
SlewRateLimiter.....	540
SngJntArmSim.....	546
Spline.....	555
SplineHelp.....	556
SplineParam.....	561
StateSpaceUtil.....	564
SwerveDriveKinematicsConstraint.....	570
SwerveDrivePoseEst.....	572
SwerveDrivePoseEst2.....	582
SwerveKinematics.....	591
SwerveModulePosition.....	599
SwerveModuleState.....	601
SwerveOdometry.....	604
TimeInterpBoolean.....	609
TimeInterpDouble.....	613
TimeInterpPose2d.....	618
TimeInterpRotation2d.....	622
TimeInterpVariant.....	626
Timer.....	631
TrajConstraint.....	638
Trajectory.....	640
TrajectoryConfig.....	646
TrajectoryGenerate.....	657
TrajectoryParam.....	661
TrajectoryState.....	664

TrajectoryUtil.....	667
Transform2d.....	670
Transform3d.....	676
Translation2d.....	681
Translation3d.....	690
TrapProfConstraint.....	698
TrapProfile.....	699
TrapProfState.....	707
Twist2d.....	708
Twist3d.....	710
Units.....	712
UnscentedKalmanFilter.....	718
Util.....	731
VecBuilder.....	751
Vector.....	757
Type Definitions.....	758
TypeDef.....	759
Enumerated Type Definitions.....	921
Enum.....	922
Appendix A – Snippet List.....	927
Macro.....	928

Introduction

The WPILib LabVIEW Math contains LabVIEW functions, data clusters, and utility programs for use in the First Robotics Competition. It implements the routines found mostly in the "math" section WPILIB library for Java and C++ that don't already exist in LabVIEW.

The utility programs, found under WPILib LabVIEW in the Windows start menu, include trajectory editors, trajectory file conversions, and robot code simulators.

The library has an extensive set of examples that can be found under the LabVIEW "Find examples..." menu.

A separate github repository stores additional function examples and sample robot code. It can be found here.
https://github.com/jsimpso81/WPilibMathLabVIEW_Examples

The library source code, package build specifications, help builder tools, and test package can be found here
<https://github.com/jsimpso81/WPilibMathLabVIEW>

Disclaimer

The WPILib LabVIEW Math library and its developers have no association with Worcester Polytechnic Institute.

Copyright

Copyright (c) 2020 James A. Simpson
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted solely for non-profit, non-commercial, education use provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the authors nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

All other use is strictly prohibited.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Description and Conventions

Function Naming

The VI in this library are grouped into categories mostly according to functionality. The VI file name starts with the group name followed by a suffix describing the function of the particular VI.

Since this library was patterned after the C++/Java version of the WPILIB, many of the group names are similar to their C++/Java counterpart. As such, those familiar with the C++/Java library should be able to use this library fairly easily. Also, while these functions are fully documented, the C++ or Java documentation will mostly describe this library as well.

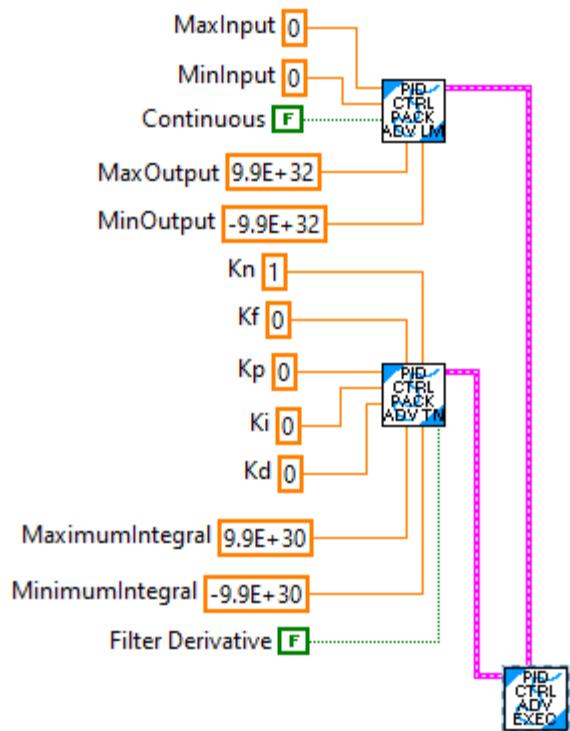
The HTML and PDF reference documentation for the library is organized by these function groups.

"Execute" Functions

In order simplify engineering control systems with LabVIEW and this library, additional functions providing all the functionality of a function group in a single VI have been provided for most function groups. The names of these VI end with the "_EXECUTE" suffix. For many function groups they may be the only function that is needed.

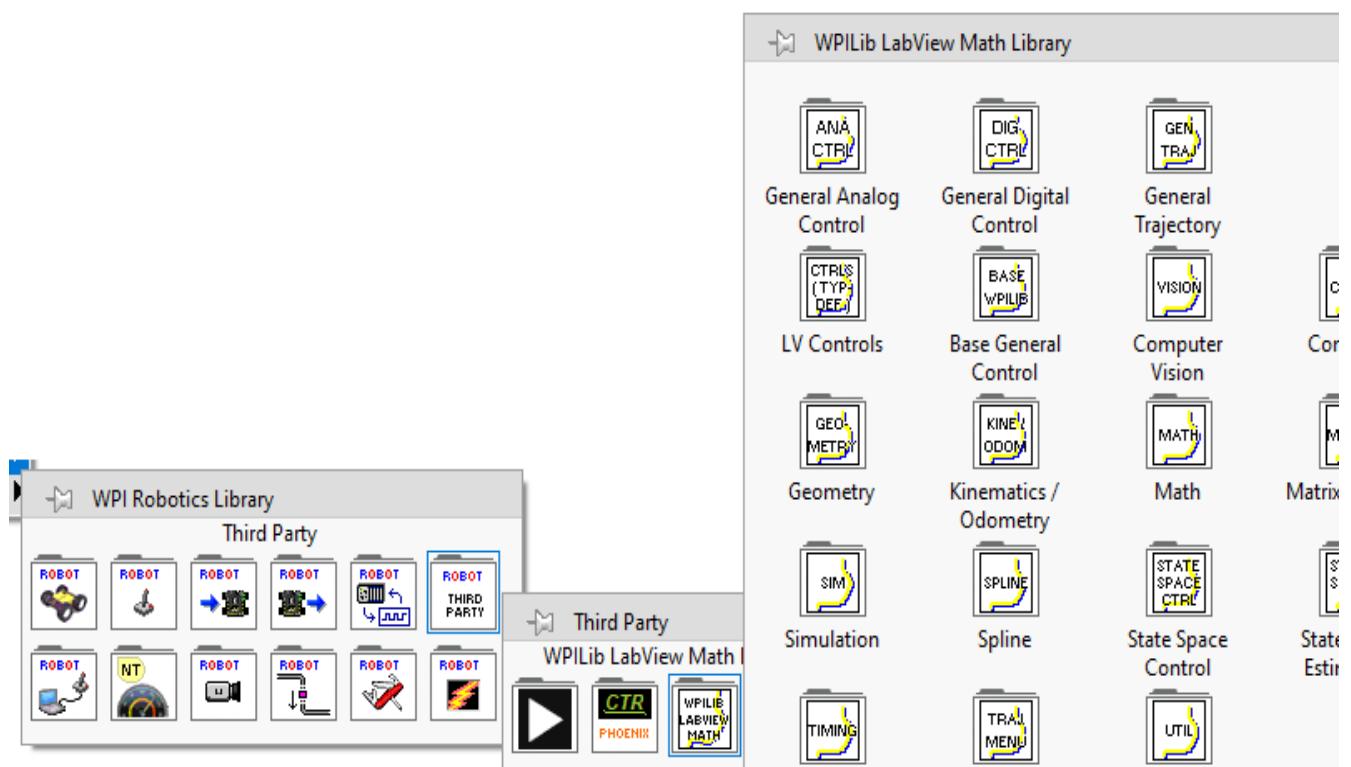
In some cases where the number of inputs to a function is large, helper functions have also been provided that feed the "_EXECUTE" function all the needed data.

The following picture shows the "_EXECUTE" function for the Advanced PID Function".

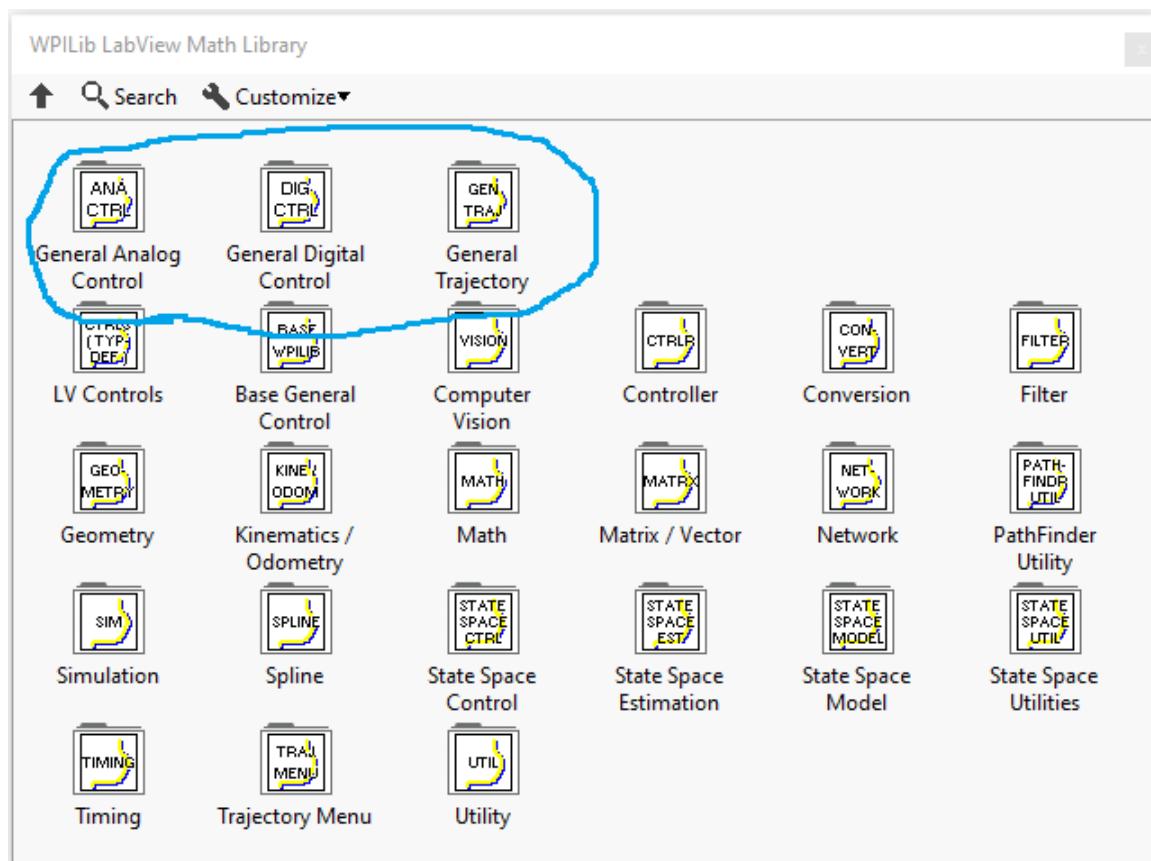


Function Menu

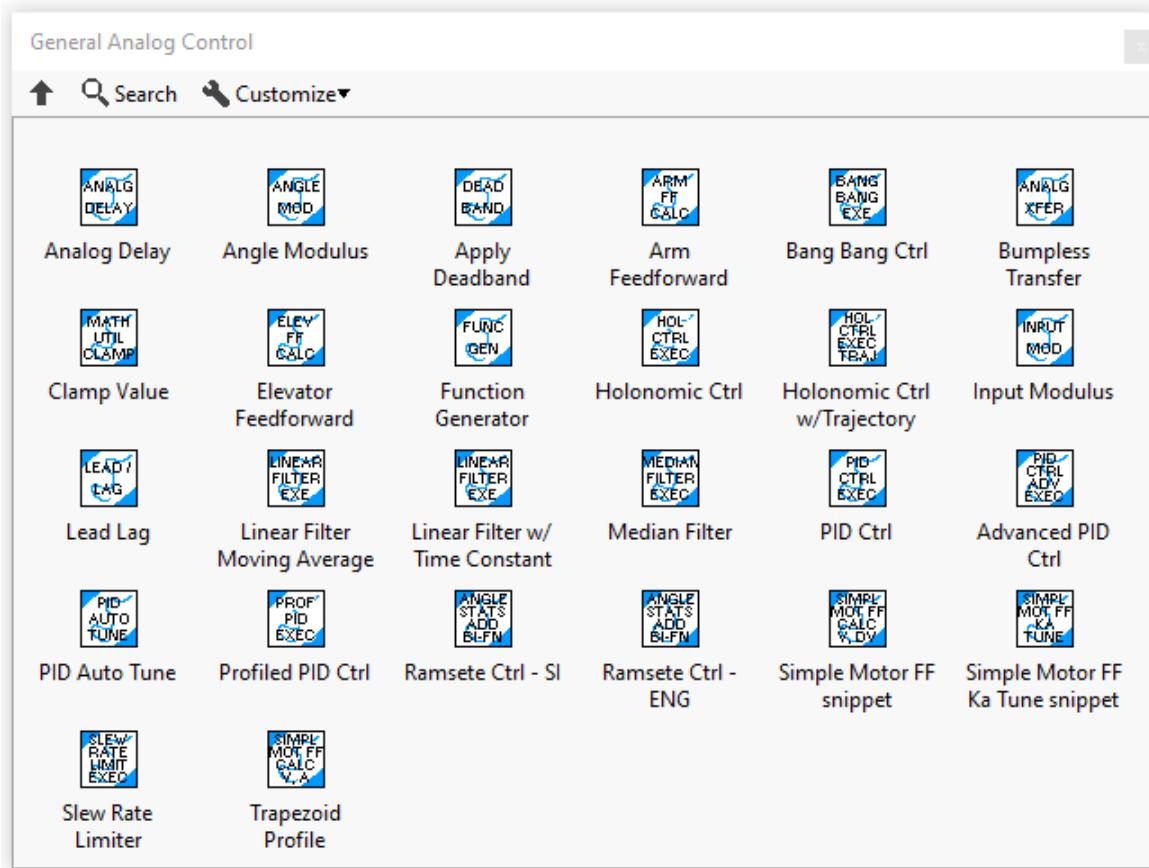
Menu items for all the subVI's are included as part of the standard LabVIEW WPI Robotics Library menu

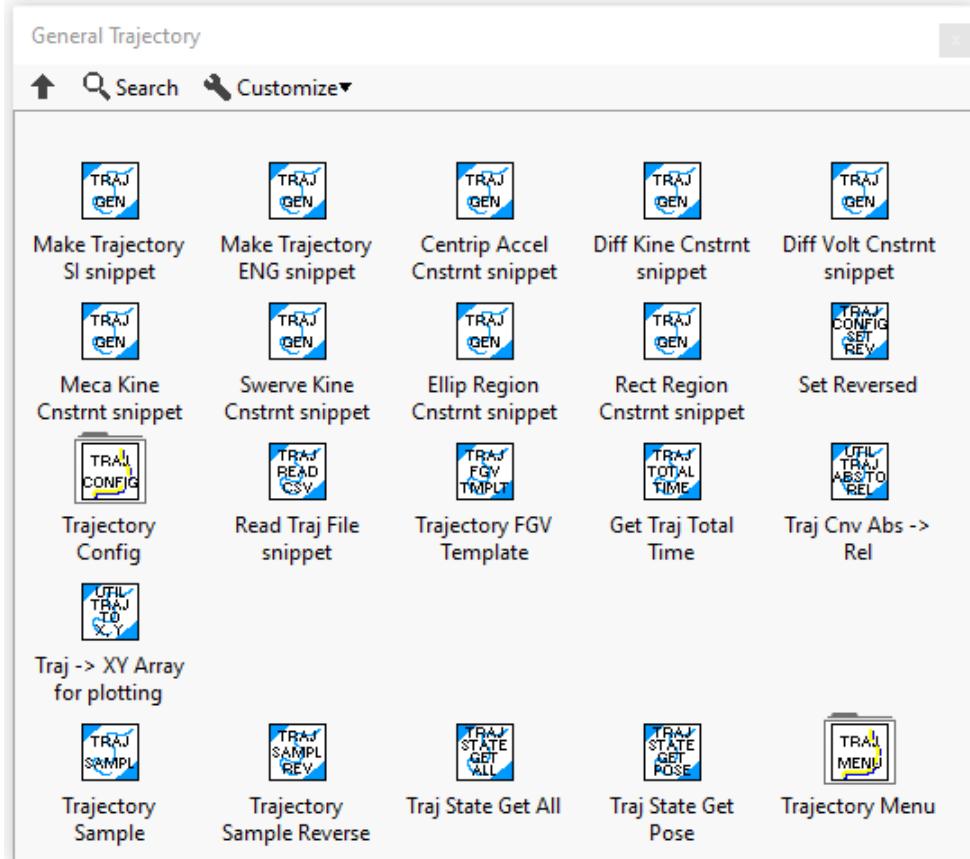
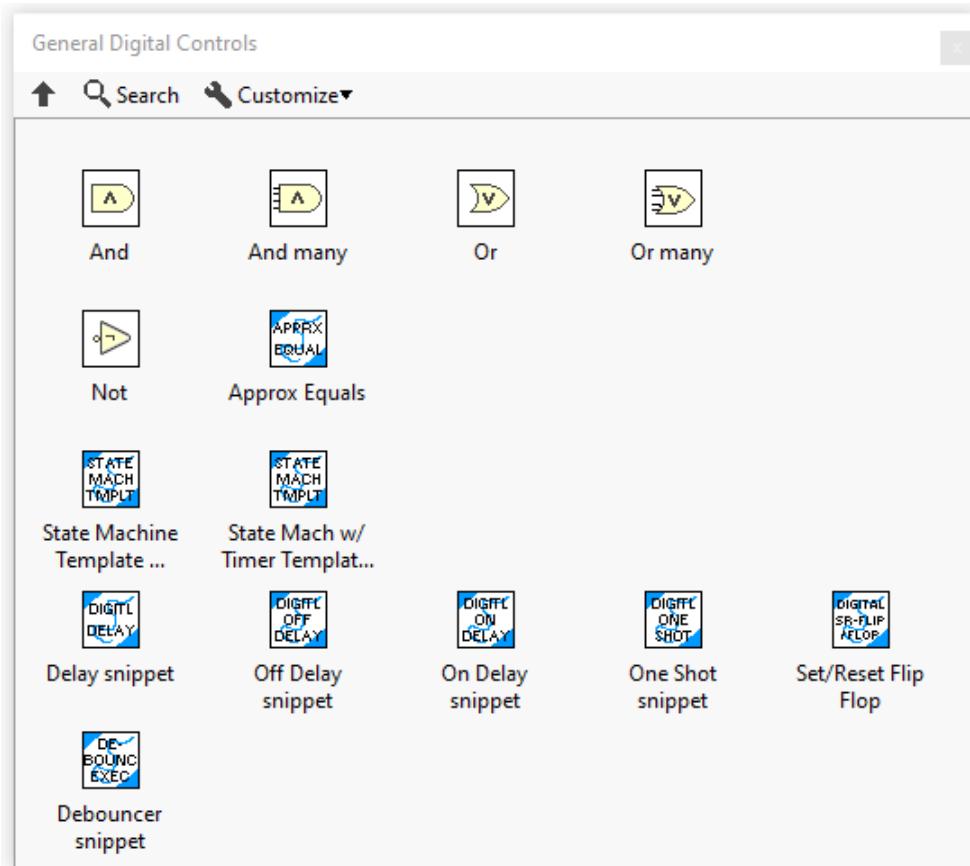


The functions anticipated to be the most used have been gathered into general sub-menus located at the top of the main menu.



Below are the Analog, Digital (boolean), and Trajectory general menus



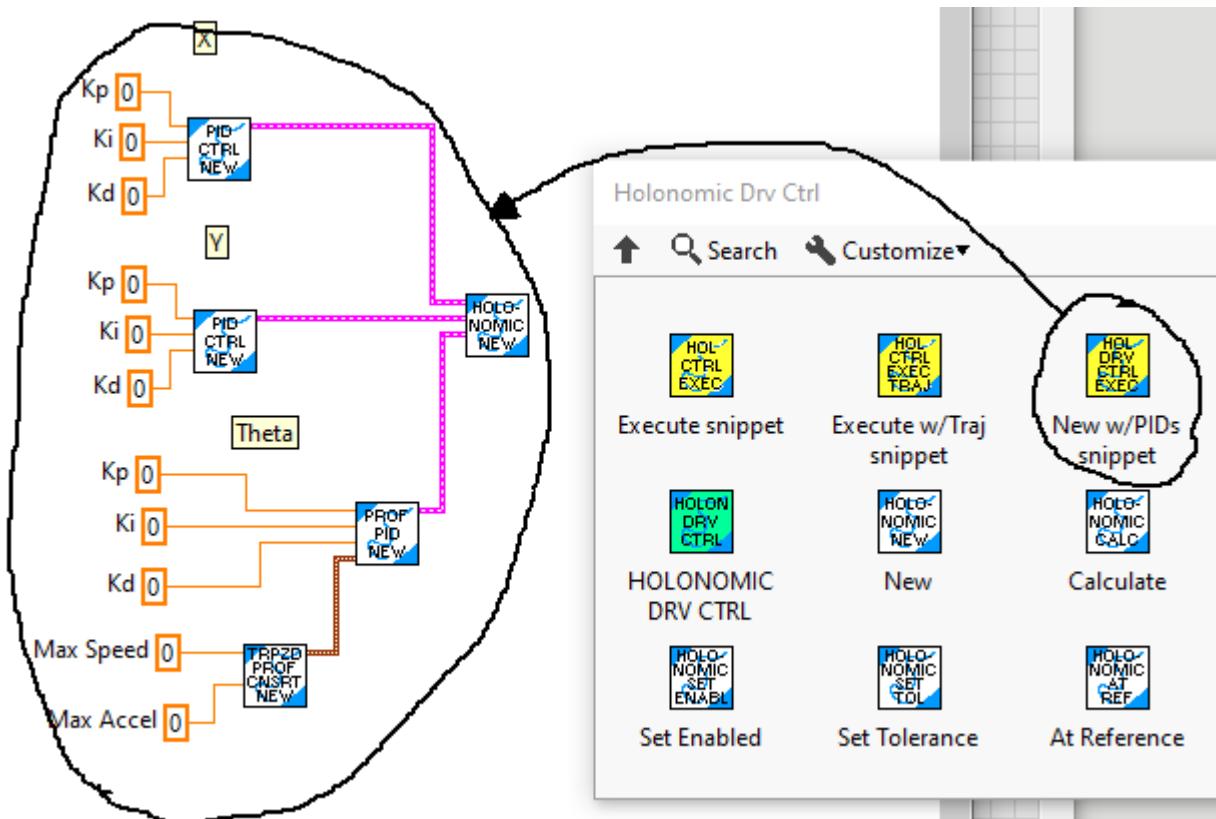


Function Snippets

To make the functions easier to use, a number of the menu items place "snippets" of code that includes the desired function and other functions or constants often wired to the function. Menu item icons for snippets use a unique background color. Unless the name is too long, the menu item name ends with "snippet".

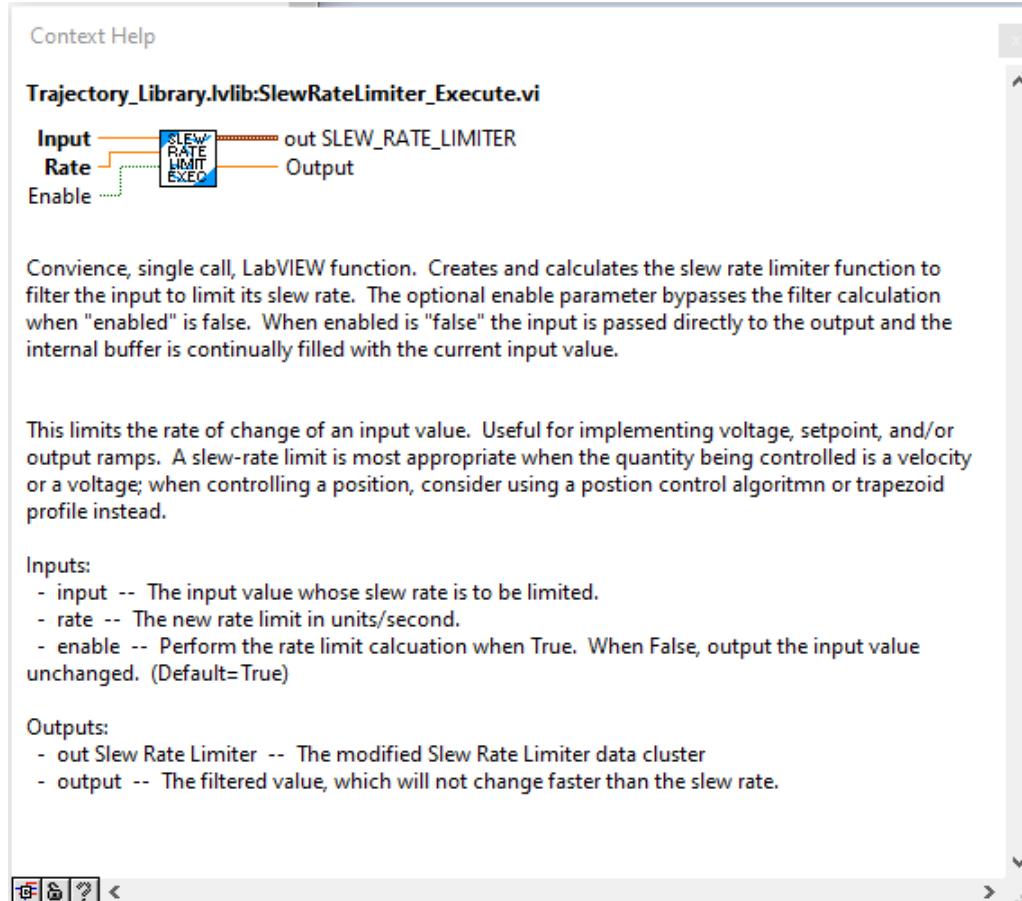
The "general" function sub-menus are all snippets.

The following shows an example of a “snippet”.



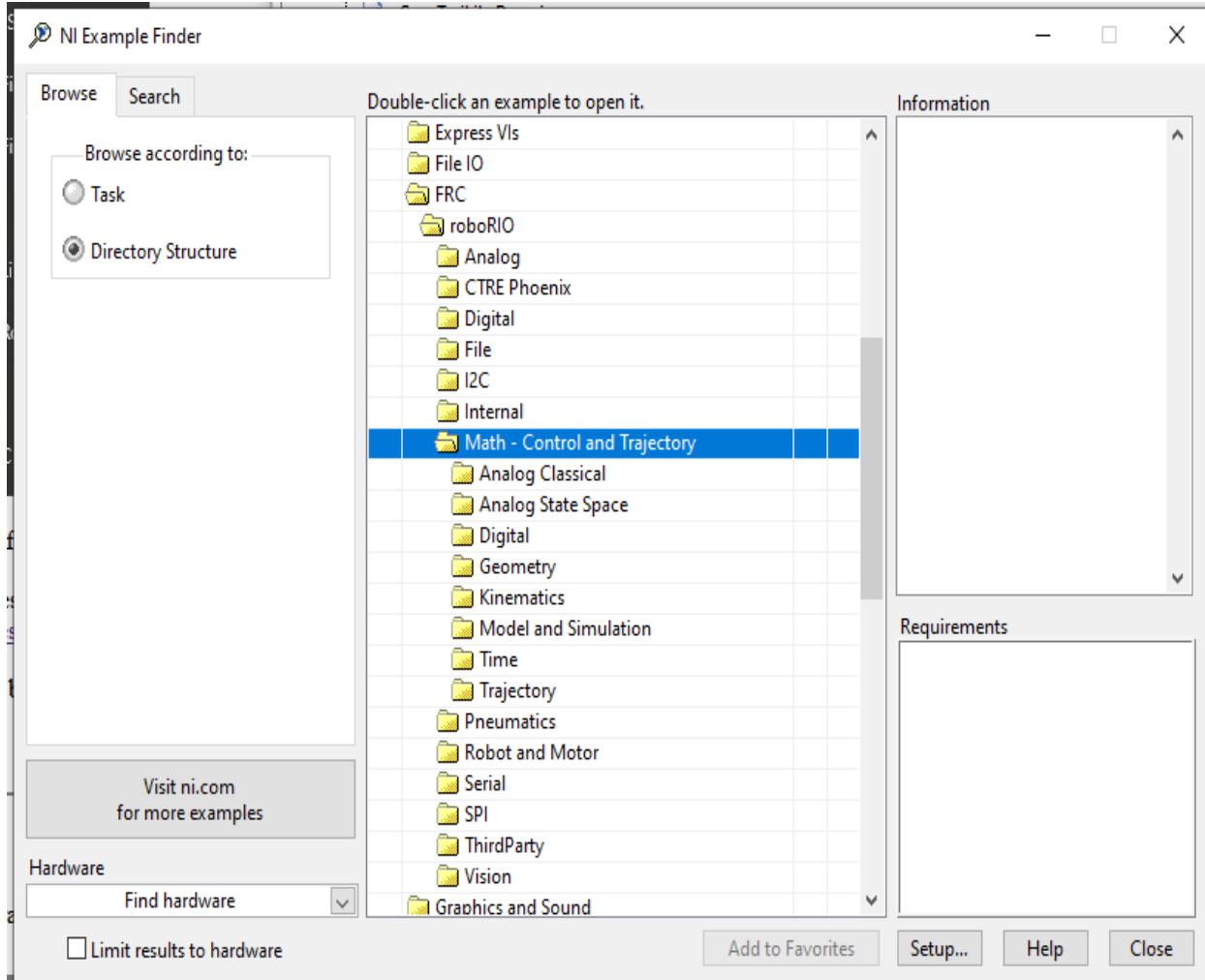
Function Help

Each VI includes help that can be accessed using the standard LabVIEW help toggle (Ctrl H).



Function Examples

Many of the functions have examples that can be found under the LabVIEW "Find examples..." function. (Help -> Find Examples...). The function examples are easiest to find when "Directory Structure" is selected.



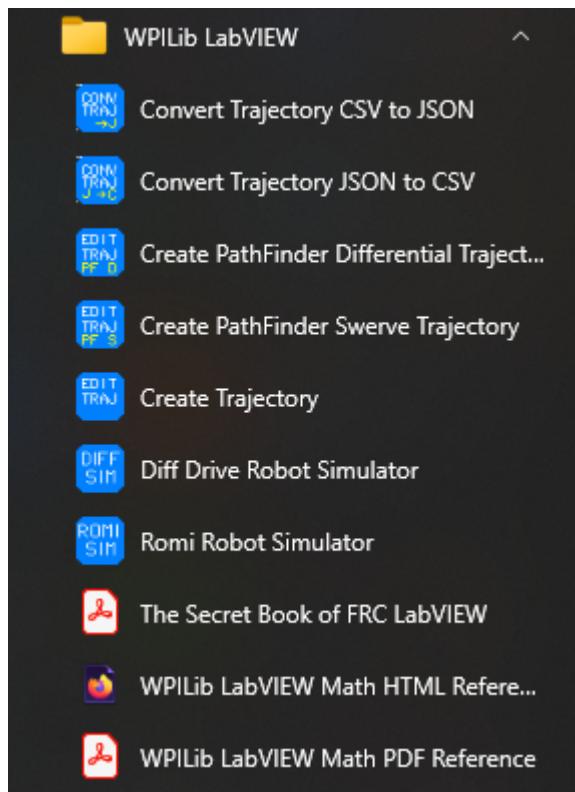
There are additional function samples, robot samples, and dashboard samples here:

https://github.com/jsimpso81/WPILibMathLabVIEW_Examples

Utility Programs

The WPILIB LabVIEW Math library contains a number of utility programs. These can be accessed from the Windows Start menu under the "WPILib LabVIEW" group. These programs include:

- Create/Edit Trajectory
- Create/Edit PathFinder Differential Drive Trajectory
- Create/Edit PathFinder Swerve Drive Trajectory
- Convert Trajectory JSON file to CSV
- Convert Trajectory CSV file to JSON
- Simulate a Romi robot
- Simulate a Differential Drive robot
- WPILib LabVIEW Math HTML help
- WPILib LabVIEW Math PDF help

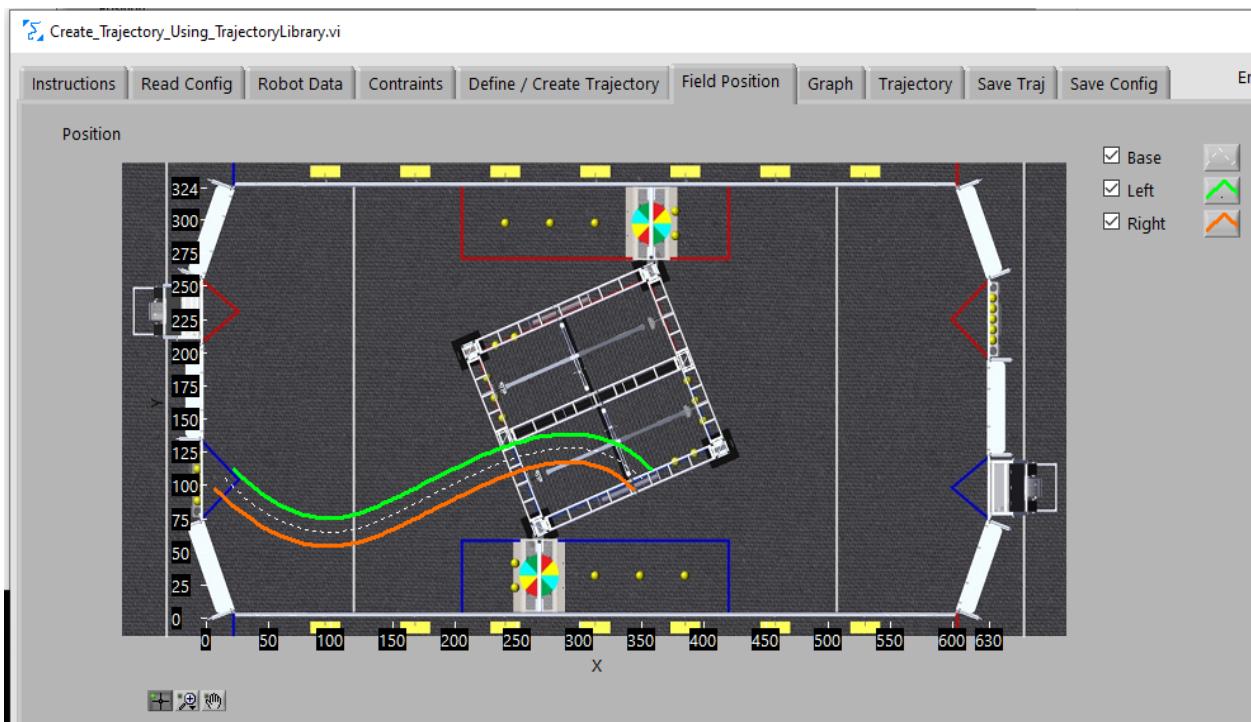


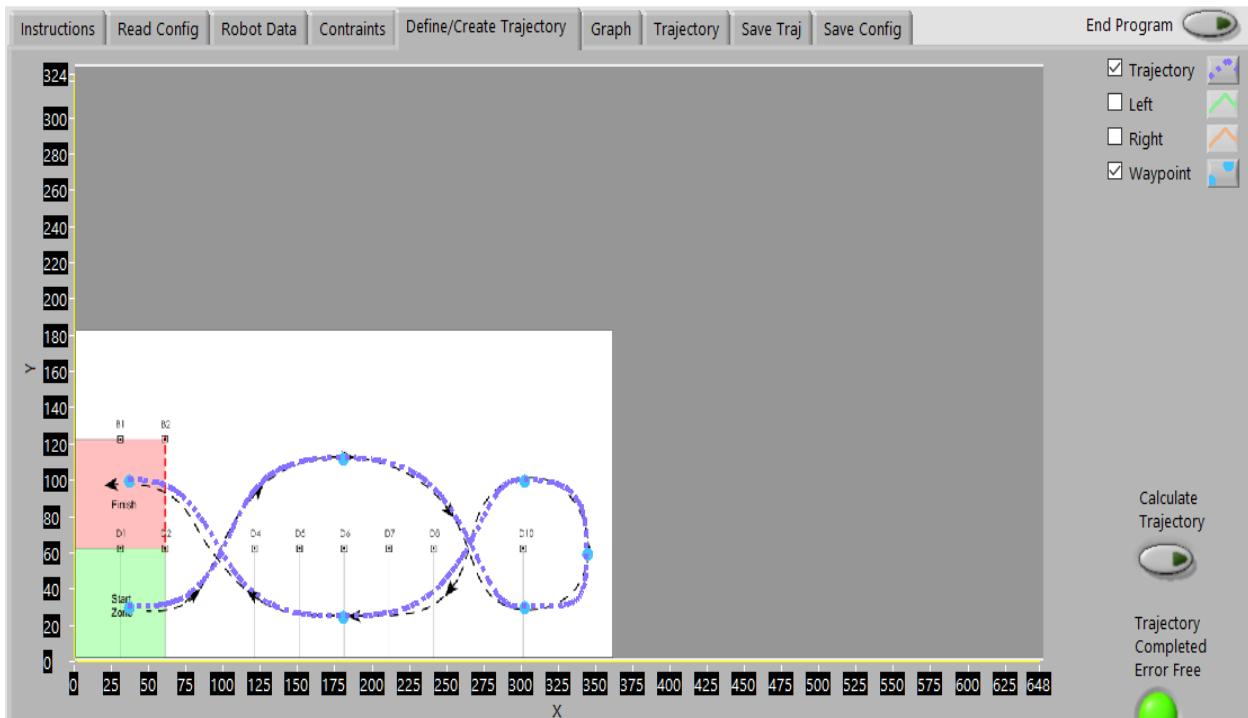
Create/Edit Trajectory

Create or edit a trajectory using the WPILIB LabVIEW Math routines and write it to a CSV (comma separated value) or JSON file that can be copied to a RoboRIO and read by the robot code. The trajectory is created by:

- Defining robot properties
- Defining drive system constraints
- Defining the trajectory "waypoints".
- Creating and reviewing the trajectory. If changes need to be made, repeat the previous steps as needed.
- Write the trajectory to a file. (CSV or JSON)
- Optionally write trajectory definition configuration. This allows future modification without redefining entire trajectory information.

Sample screen shots





Sample CSV trajectory file

```

#===== TRAJECTORY BEGIN: testing
#
#===== way points: X(abs), Y(abs), Angle(abs), X(rel), Y(rel), Angle(rel)
#----- meters, meters, radians, meters, meters, radians
#-----Waypoint: 0.381 2.667 -0.785 0.000 0.000 0.000
#-----Waypoint: 2.667 1.651 0.000 2.335 0.898 0.785
#-----Waypoint: 5.334 2.692 0.785 3.484 3.520 1.571
#-----Waypoint: 8.890 2.667 -0.785 6.017 6.017 0.000
#
#=====Robot Configuration
#-----Maximum Velocity: 1.27000 m/s
#-----Maximum Acceleration: 1.01600 m/s^2
#-----Start Velocity: 0.00000 m/s
#-----End Velocity: 0.00000 m/s
#-----Trajectory Reversed: FALSE
#
#=====Differential Drive Kinematics Constraint
#-----Configured: Yes
#-----Maximum Velocity: 1.27000 m/s
#-----Track width: 0.53000 m
#
#=====Differential Drive Voltage Constraint
#-----Configured: Yes
#-----Track width: 0.53000 m
#-----Maximum voltage: 12.00000 V
#-----Feedforward Ks: 0.90000 V
#-----Feedforward Kv: 8.74000 V s/m
#-----Feedforward Ka: 1.92000 V s^2/m
#
#=====Centripetal Acceleration Constraint
#-----Configured: Yes
#-----Maximum Centripetal Acceleration: 0.50800 m/s^2
#
#=====Mecanum Drive Kinematics Constraint
#-----Configured: No
#
#=====Swerve Drive Kinematics Constraint
#-----Configured: No
#
#=====Trajectory Creation
#-----Spline type: Cubic
#-----Trajectory orientation: Robot Relative
#
#-- Time, Velocity, Accel, X pos, Y pos, Heading, Curvature, Comment
#-- Seconds, m/s, m/s^2, m, m, radians, radians/meter,
0.00000, 0.00000, 1.01600, 0.00000, 0.00000, 0.00000, 0.20010, Segment: 1
0.42853, 0.43539, 1.01600, 0.09328, 0.00088, 0.01898, 0.20685, Segment: 2
0.60435, 0.61402, 1.01600, 0.18550, 0.00352, 0.03838, 0.21390, Segment: 3
0.73813, 0.74994, 1.01600, 0.27663, 0.00792, 0.05823, 0.22124, Segment: 4
0.84996, 0.86356, 1.01600, 0.36663, 0.01408, 0.07853, 0.22889, Segment: 5
0.94766, 0.96282, 0.64069, 0.45550, 0.02200, 0.09931, 0.23685, Segment: 6

```

Create/Edit PathFinder Differential Drive Trajectory

Create a trajectory using Jaci Brunning's Pathfinder library for a robot using a "differential drive" and write it to a CSV (comma separated value) file that can be used by the WPILib LabVIEW Math library.

More information on Pathfinder can be found here <https://github.com/JaciBrunning/Pathfinder>

One of the biggest differences between this Trajectory Library and Pathfinder is that Pathfinder contains acceleration limiting (jerk) that does not exist in the other available libraries.

Sample trajectory file

```
#===== TRAJECTORY BEGIN: Robot Sample 2 - Trajectory using Pathfinder
#
#===== way points: X(abs), Y(abs), Angle(abs), X(rel), Y(rel), Angle(rel)
#----- meters, meters, radians, meters, meters, radians
#-----Waypoint: 0.381 2.667 -0.785 0.000 0.000 0.000
#-----Waypoint: 2.667 1.651 0.000 2.335 0.898 0.785
#-----Waypoint: 5.334 2.692 0.785 3.484 3.520 1.571
#
#=====Robot Configuration
#-----Robot type: Tank
#-----Maximum Velocity: 55.00000 in/s
#-----Maximum Acceleration: 40.00000 in/s^2
#-----Maximum Jerk: 70.00000 in/s^2
#-----Sample count: 100000.00000
#-----Wheelbase width: 20.86600 inch
#-----Robot maximum velocity: 65.00000 in/sec
#
#=====Trajectory Creation
#-----Trajectory source: Pathfinder
#-----Trajectory orientation: Robot Relative
#
#-- Time, Velocity, Accel, X pos, Y pos, Heading, Curvature, Comment
#-- Seconds, m/s, m/s^2, m, m, radians, radians/meter,
0.00000, 0.00070, 0.03491, 0.00000, -0.00000, 0.00000, 0.42225,Segment: 1
0.06000, 0.00698, 0.13963, 0.00018, 0.00000, 0.00004, 0.21642,Segment: 2
0.12000, 0.01955, 0.24435, 0.00095, 0.00000, 0.00020, 0.21310,Segment: 3
0.18000, 0.03840, 0.34908, 0.00266, 0.00000, 0.00057, 0.21234,Segment: 4
0.24000, 0.06353, 0.45380, 0.00569, 0.00000, 0.00121, 0.21225,Segment: 5
0.30000, 0.09495, 0.55852, 0.01042, 0.00001, 0.00221, 0.21251,Segment: 6
0.36000, 0.13265, 0.66324, 0.01722, 0.00003, 0.00365, 0.21306,Segment: 7
0.42000, 0.17663, 0.76797, 0.02647, 0.00007, 0.00563, 0.21387,Segment: 8
```

Create/Edit PathFinder Swerve Drive Trajectory

Create a trajectory using Jaci Brunning's Pathfinder library for a robot using a "swerve drive" and write it to a CSV (comma separated value) file that can be used by the WPILib LabVIEW Math library.

More information on Pathfinder can be found here <https://github.com/JaciBrunning/Pathfinder>

One of the biggest differences between this Trajectory Library and Pathfinder is that Pathfinder contains acceleration limiting (jerk) that does not exist in the other available libraries.

sample trajectory file

```

===== TRAJECTORY BEGIN: Robot Sample 2 - Trajectory using Pathfinder
#
#===== way points: X(abs), Y(abs), Angle(abs), X(rel), Y(rel), Angle(rel)
#----- meters, meters, radians, meters, meters, radians
#-----Waypoint: 0.381 2.667 -0.785 0.000 0.000 0.000
#-----Waypoint: 2.667 1.651 0.000 2.335 0.898 0.785
#-----Waypoint: 5.334 2.692 0.785 3.484 3.520 1.571
#
#=====Robot Configuration
#-----Robot type: Tank
#-----Maximum Velocity: 55.00000 in/s
#-----Maximum Acceleration: 40.00000 in/s^2
#-----Maximum Jerk: 70.00000 in/s^3
#-----Sample count: 100000.00000
#-----Wheelbase width: 20.86600 inch
#-----Robot maximum velocity: 65.00000 in/sec
#
#=====Trajectory Creation
#-----Trajectory source: Pathfinder
#-----Trajectory orientation: Robot Relative
#
#-- Time, Velocity, Accel, X pos, Y pos, Heading, Curvature, Comment
#-- Seconds, m/s, m/s^2, m, m, radians, radians/meter,
  0.00000, 0.00070, 0.03491, 0.00000, -0.00000, 0.00000, 0.42225,Segment: 1
  0.06000, 0.00698, 0.13963, 0.00018, 0.00000, 0.00004, 0.21642,Segment: 2
  0.12000, 0.01955, 0.24435, 0.00095, 0.00000, 0.00020, 0.21310,Segment: 3
  0.18000, 0.03840, 0.34908, 0.00266, 0.00000, 0.00057, 0.21234,Segment: 4
  0.24000, 0.06353, 0.45380, 0.00569, 0.00000, 0.00121, 0.21225,Segment: 5
  0.30000, 0.09495, 0.55852, 0.01042, 0.00001, 0.00221, 0.21251,Segment: 6
  0.36000, 0.13265, 0.66324, 0.01722, 0.00003, 0.00365, 0.21306,Segment: 7
  0.42000, 0.17663, 0.76797, 0.02647, 0.00007, 0.00563, 0.21387,Segment: 8

```

Convert Trajectory JSON file to CSV

This utility reads a JSON file created by the Trajectory Creation utility, PathWeaver, PathPlanner, or other WPILib compatible tool and converts it to a CSV file that can be used by the WPILib LabVIEW Math library (The WPILib LabVIEW math library can also use the JSON file directly.)

Convert Trajectory CSV file to JSON

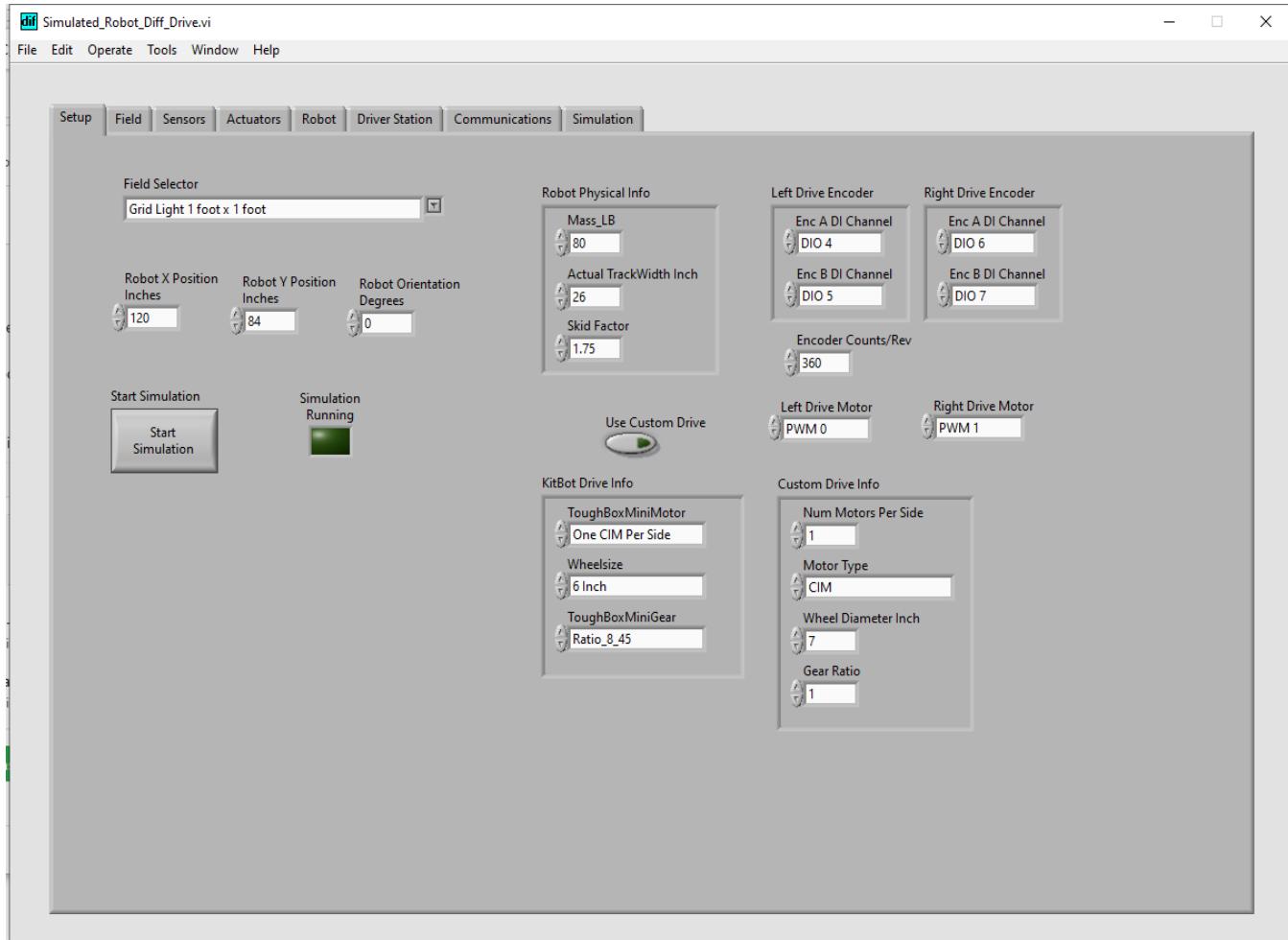
This utility reads a CSV file created by the Trajectory Creation utility and converts it to a JSON file that can be used by the WPILib C++/Java library. (The WPILib LabVIEW math trajectory creation tool can also write JSON files directly.)

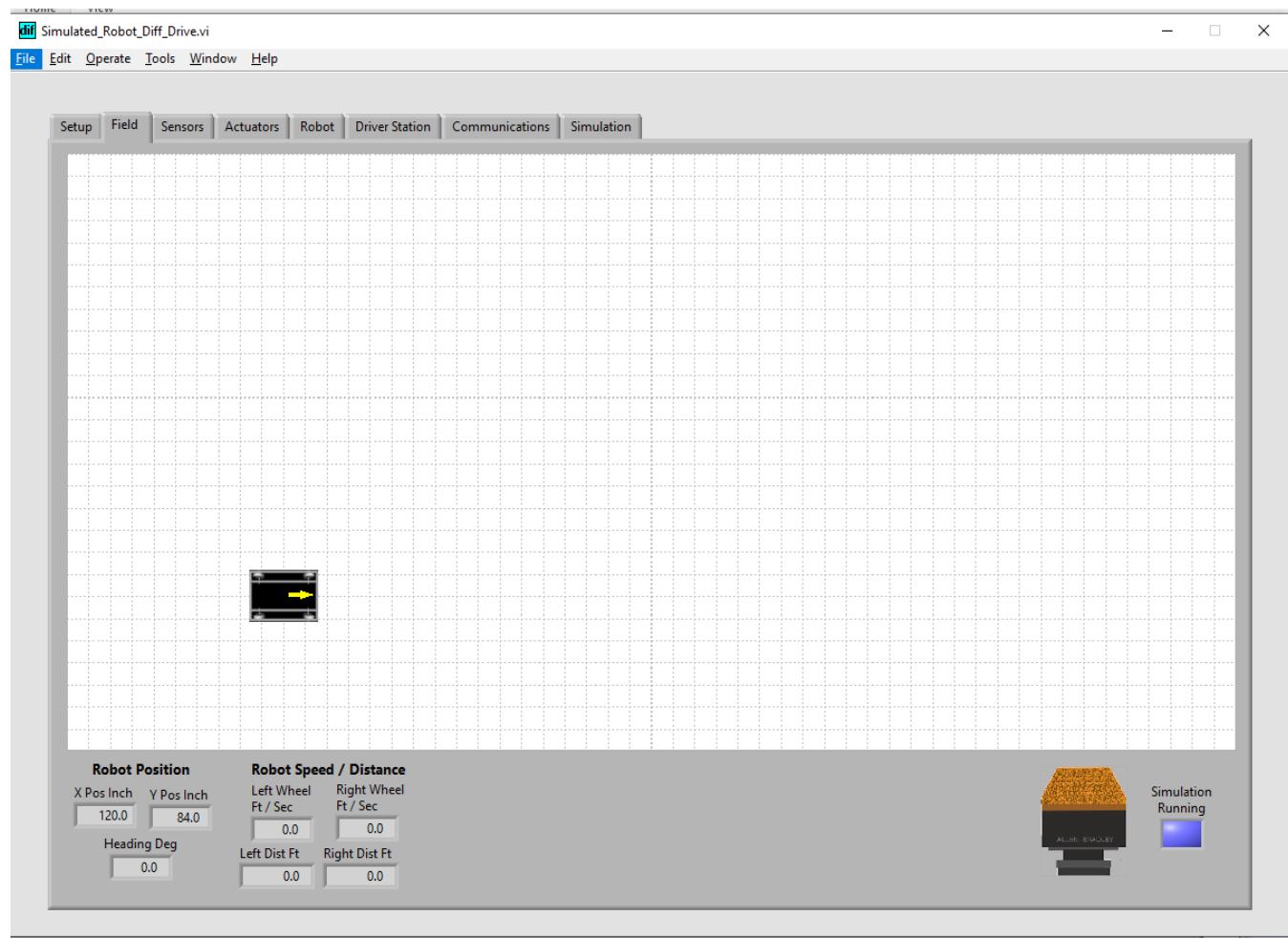
Romi Robot Simulator

Simulate a Differential Drive robot.

Differential Drive Simulator

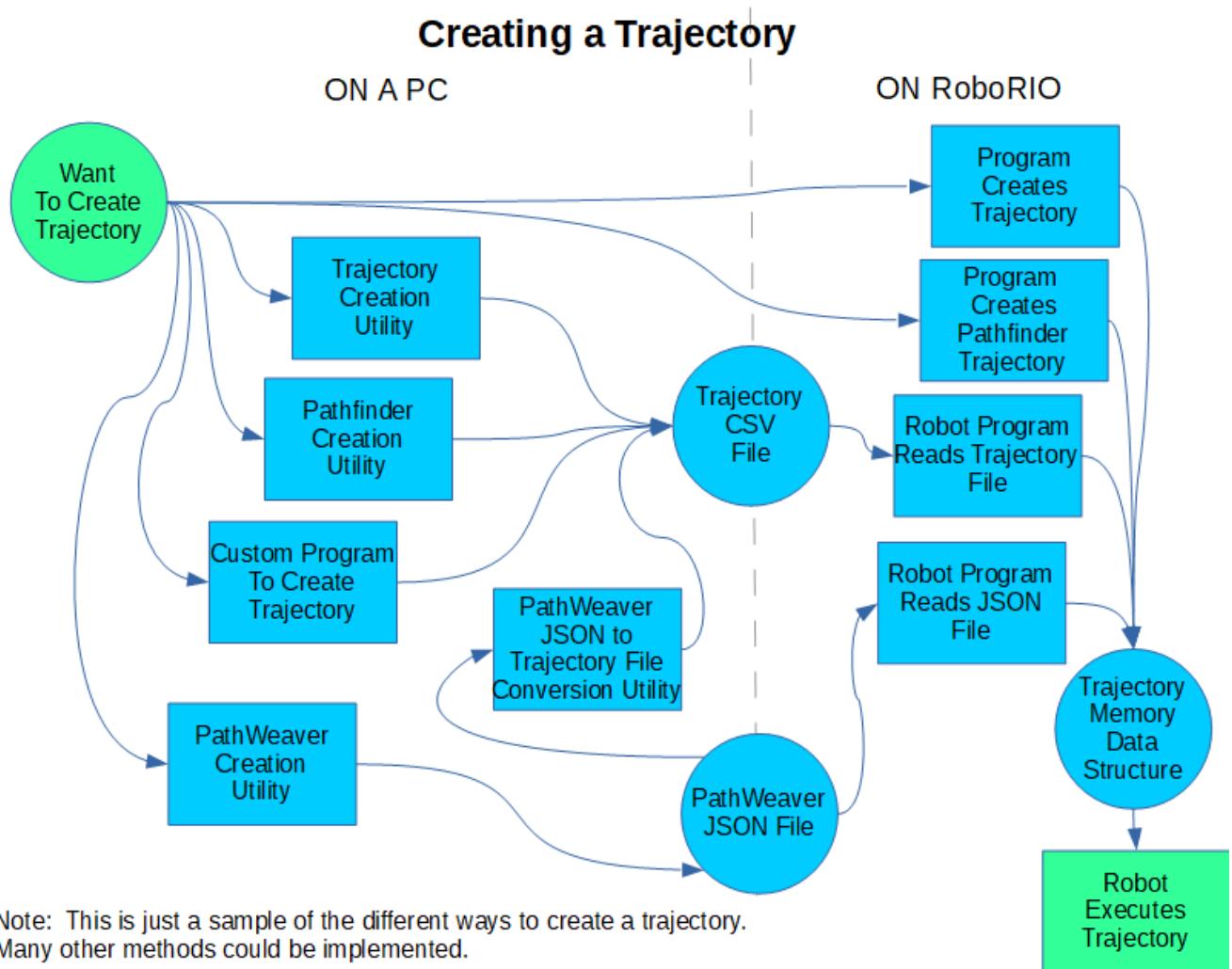
Simulate a Differential Drive robot. This allows for testing of drive code and execution of trajectories.





Trajectory Creation

Robot trajectories can be created a number of different ways. The following chart shows some (maybe most) of the different ways of creating a trajectory for robot execution.



Reporting Issues and Suggesting Enhancements

Use the github issues menu <https://github.com/jsimpso81/WPIlibMathLabVIEW/issues> to submit information on issues and suggested enhancements.

When submitting an issue, please follow these guidelines:

- Make certain you are using the latest version of the library. If not, install the latest version and see if the issue still occurs.
- See if the issue has already been reported. If so, add any new details to the existing issue.
- Write a very detailed specific description of the problem.
- Include any screen shots or other data needed to show the issue.
- Describe exactly how to reproduce the error. (Likely, I can't try this on your robot...)
- If possible, try and condense the problem to a small test program that isolates the issue.
- If you found a work around, please include it for others to use. See the Contributing section of the github repository for additional details.

I'll attempt to deal with issues in a reasonable time, but there are no guarantees.

Contributing

Contributions of enhancements, bug fixes, and new routines are welcome. If new routines are not already part of the C++/Java WPILIB, then it seems fair that they should be written solely by FRC student members, with confirmation from a mentor on their team to be able to be candidates for inclusion.

See the Contributing section on the github repository for additional details.

Function Groups

AnalogDelay

AnalogDelay_Execute



This VI implements an analog "time delay". The output is the input delayed by "Delay" seconds. If the "Delay" time does not exactly match the sample period,, the value is calculated by interpolating the nearest buffer samples.

Inputs:

- Input -- Double input value
- Delay Time -- Time (seconds) to delay the output value. If time delay is increased, then the buffer is re-created. FALSE will be returned until the buffer has sufficient data to return a delayed value.
- Time -- Continuously counting system time, Seconds. If not wired the FPGA time will be used.

Outputs:

- Output -- Double output value with delay applied
- IsPresent -- Set to TRUE if the buffer has enough data to return a delayed value.

AngleStats

AngleStats_AngleAdd



Adds a and b while normalizing the resulting value in the selected row as an angle.

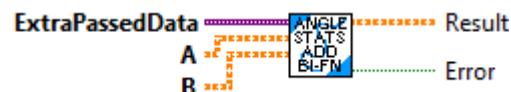
Inputs:

- a -- A vector to add with.
- b -- A vector to add with.
- row -- The row containing angles to be normalized.

Outputs:

- Result -- Sum of two vectors with angle at the given index normalized.

AngleStats_AngleAdd_CallbackHelp



Returns a function that adds two vectors while normalizing the resulting value in the selected row as an angle. The calling parameters for this function adhere to those required to be called as F or H functions.

Inputs:

- ExtraPassedData -- Variant containing extra data passed to the function. For this instance the data contains.
 - integer -- row -- The row containing angles to be normalized.
- A -- A matrix

- B -- B matrix

Outputs:

- Result -- Function returning of two vectors with angle at the given index normalized.
- Error -- If TRUE, an error has occurred.

AngleStats_AngleMean



Computes the mean of sigmas with the weights Wm while computing a special angle mean for a select row.

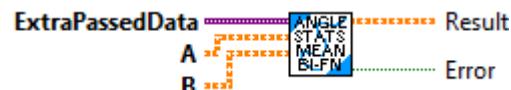
Inputs:

- sigmas -- Sigma points.
- Wm -- Weights for the mean.
- Row -- The row containing the angles.

Outputs:

- Result -- Mean of sigma points.

AngleStats_AngleMean_CallbackHelp



Returns a function that computes the mean of sigmas with the weights Wm while computing a special angle mean for a select row.

Inputs:

- ExtraPassedData -- Variant containing extra data passed to this routine. For this routine the data contains:

- integer -- Row -- The row containing the angles.
- A -- A matrix
- B -- B matrix

Outputs:

- Result -- Function returning mean of sigma points.
- Error -- If TRUE, an error occurred.

AngleStats_AngleResidual



Subtracts a and b while normalizing the resulting value in the selected row as if it were an angle.

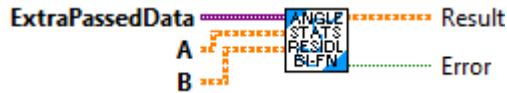
Inputs:

- a -- A vector to subtract from.
- b -- A vector to subtract with.
- row -- The row containing angles to be normalized.

Outputs:

- Result -- Difference of two vectors with angle at the given index normalized.

AngleStats_AngleResidual_CallbackHelp



Returns a function that subtracts two vectors while normalizing the resulting value in the selected row as if it were an angle. The calling parameters for this function adhere to those required to be called as F or H functions.

Inputs:

- ExtraPassedData -- Extra data passed into this function. For this function, it contains:
 - angleStateIdx -- The row containing angles to be normalized.
- A -- A matrix
- B -- B matrix

Outputs:

- Result -- Function returning difference of two vectors with angle at the given index normalized.
- Error -- If TRUE, an error occurred.

AprilTag

AprilTag_Equals



Determine if two April Tag data clusters are equal.

Inputs:

- AprilTag -- AprilTag -- First data cluster to check
- OtherAprilTag -- AprilTag -- Second data cluster to check

Outputs:

- Equal -- Boolean -- TRUE if both data clusters are the same.

AprilTag_GetAll



Get individual items from the AprilTag data cluster

Inputs:

- AprilTag -- AprilTag -- Data cluster storing AprilTag

Outputs:

- ID -- Integer -- AprilTag ID value
- POSE3d -- Pose3d -- Position of the April Tag. This could be relative or absolute. It is most often absolute position.

AprilTag_New



Create a new April Tag data cluster

Inputs:

-- ID -- Integer -- April Tag ID value

-- Pose3d - Pose3d -- Position of the April tag. Most often this will be an absolute position. Relative positions can be used.

AprilTagFieldLayout

AprilTagFieldLayout_GetField



Returns data about an AprilTagFieldLayout.

Inputs:

-- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster

Outputs:

-- FieldLength -- double -- Field length (meters)
-- FieldWidth -- double -- Field width (meters)
-- OriginPosition -- enum -- Location of field origin
-- OriginPose3d -- Pose3 -- Position of the field origin.

AprilTagFieldLayout_GetOriginPosition



Returns data about an AprilTagFieldLayout.

Inputs:

-- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster

Outputs:

-- OriginPosition -- enum -- Location of field origin
-- OriginPose3d -- Pose3 -- Position of the field origin.

AprilTagFieldLayout_GetTagPose



Returns the location of a particular AprilTag on the field.

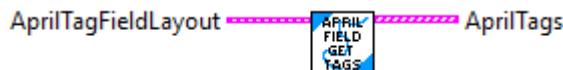
Inputs:

- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster
- TagID -- Integer -- The AprilTag to locate.

Outputs:

- TagPose3d -- Pose3d -- Location of the AprilTag. This is relative to the specific origin specified in the AprilTagFieldLayout.
 - Found -- Boolean -- TRUE if the TagID was found in the field definition.
-

AprilTagFieldLayout_GetTags



Returns an array of all the AprilTags for the provided field layout.

Inputs:

- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster

Outputs:

- AprilTags -- Array of AprilTags -- All AprilTags defined for this field.
-

AprilTagFieldLayout_New



Construct a new AprilTagFieldLayout from a list of AprilTag data clusters and other field definition data.

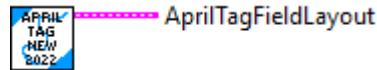
Inputs:

- AprilTags -- Array of AprilTags -- List of AprilTags
- FieldLength -- double -- Field length (meters)
- FieldWidth -- double -- Field width (meters)

Outputs:

- AprilTagFieldLayout -- AprilTagFieldLayout -- Field layout data cluster.

AprilTagFieldLayout_New2022



Define the FRC 2022 April Tag field layout.

NOTE -- This function has not been completed.

Inputs:

Outputs:

- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster containing field definition.

AprilTagFieldLayout_New2023



Define the FRC 2023 April Tag field layout.

Inputs:

Outputs:

-- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster containing field definition.

AprilTagFieldLayout_New2024



Define the FRC 2024 April Tag field layout.

Inputs:

Outputs:

-- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster containing field definition.

AprilTagFieldLayout_NewSelect



Define the April Tag field layout based on the input FRC year.

Inputs:

-- AprilTagField -- enum -- Selected Field.

Outputs:

-- AprilTagFieldLayout -- AprilTagFieldLayout -- Data cluster containing field definition.

AprilTagFieldLayout_SetOrigin



Sets the origin based on a predefined enumeration of coordinate frame origins. The origins are calculated from the field dimensions.

This transforms the Pose3d objects returned by {@link #getTagPose(int)} to return the correct pose relative to a predefined coordinate frame.

Inputs:

- in AprilTagFieldLayout -- AprilTagFieldLayout -- Input field data cluster.
- OriginPosition -- enum -- The predefined origin

Outputs:

- out AprilTagFieldLayout -- AprilTagFieldLayout -- Modified field data cluster
- OriginPosition3d -- Pose3d -- Position of the origin.

AprilTagFieldLayout_SetOrigin_Position



Sets the origin for tag pose transformation.

This transforms the Pose3d objects returned by getTagPose(int) to return the correct pose relative to the provided origin.

Inputs:

- In AprilTagFieldLayout -- AprilTagFieldLayout -- Input data cluster
- OriginPose3d -- Pose3d -- Position of the origin.

Outputs:

- outAprilTagFieldLayout -- AprilTagFieldLayout -- Modified data cluster.

* @param origin The new origin for tag transformations

AprilTagPoseEstimate

AprilTagPoseEstimate_GetAll



Get individual data items from the AprilTagPoseEstimate data cluster.

Inputs:

- AprilTagPoseEstimate -- AprilTagPoseEstimate -- Input data cluster

Outputs:

- AprilTransform3d_1 -- Transform3d -- First april tag transform
- AprilTransform3d_1 -- Transform3d -- First april tag transform
- Error1 -- double -- Object-space error of pose 1
- Error2 -- double -- Object-space error of pose 2

AprilTagPoseEstimate_GetAmbiguity



Get the ratio of pose reprojection errors, called ambiguity. Numbers above 0.2 are likely to be ambiguous.

Inputs:

- AprilTagPoseEstimate -- AprilTagPoseEstimate -- Input data cluster

Outputs:

- Ambiguity -- double -- The ratio of pose reprojection errors.

AprilTagPoseEstimate_New



Create a new AprilTagPoseEstimate data cluster from two transform and their individual errors.

Inputs:

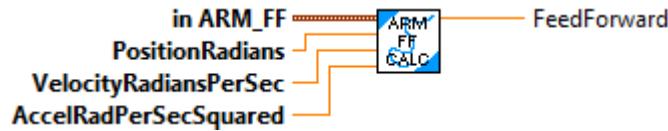
```
-- AprilTransform_1 -- Transform3d --
-- AprilTransform_1 -- Transform3d --
-- Error1 -- double -- Object-space error of pose 1
-- Error2 -- double -- Object-space error of pose 2
```

Outputs:

```
-- AprilTagPoseEstimate -- AprilTagPoseEstimate -- Output data cluster.
```

ArmFF

ArmFF_Calculate



Calculates the feedforward from the gains and setpoints.

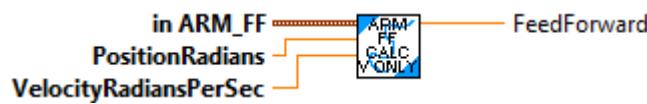
Inputs:

- ArmFF -- The ArmFF data cluster
- positionRadians -- The position (angle) setpoint.
- velocityRadPerSec -- The velocity setpoint.
- accelRadPerSecSquared -- The acceleration setpoint.

Outputs:

- feedForward -- The computed feedforward.

ArmFF_CalculateVelocityOnly



Calculates the feedforward from the gains and velocity setpoint (acceleration is assumed to be zero).

Inputs:

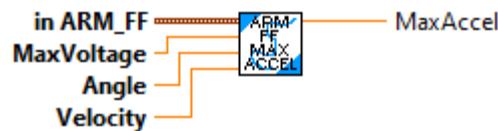
- ArmFF -- The ArmFF data cluster

- positionRadians -- The position (angle) setpoint.
- velocity -- The velocity setpoint.

Outputs:

- feedForward -- The computed feedforward.

ArmFF_MaxAchieveAccel



Calculates the maximum achievable acceleration given a maximum voltage supply, a position, and a velocity. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the velocity constraint, and this will give you a simultaneously-achievable acceleration constraint.

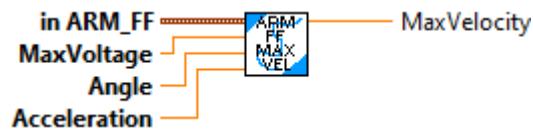
Inputs:

- ArmFF -- The ArmFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the arm.
- angle -- The angle of the arm.
- velocity -- The velocity of the arm.

Outputs:

- MaxAccel -- The maximum possible acceleration at the given velocity.

ArmFF_MaxAchieveVelocity



Calculates the maximum achievable velocity given a maximum voltage supply, a position, and an acceleration. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the acceleration constraint, and this will give you a simultaneously-achievable velocity constraint.

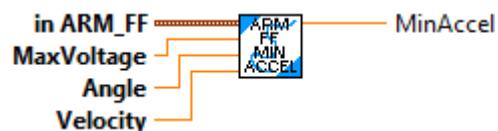
Inputs:

- ArmFF -- The ArmFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the arm.
- angle -- The angle of the arm. (radians)
- acceleration -- The acceleration of the arm.

Outputs:

- MaxVelocity -- The maximum possible velocity at the given acceleration and angle.

ArmFF_MinAchieveAccel



Calculates the minimum achievable acceleration given a maximum voltage supply, a position, and a velocity. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the velocity constraint, and this will give you a simultaneously-achievable acceleration constraint.

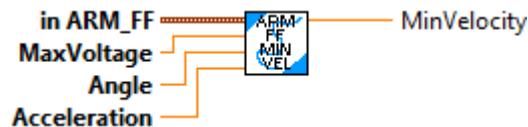
Inputs:

- ArmFF -- The ArmFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the arm.
- angle -- The angle of the arm.
- velocity -- The velocity of the arm.

Outputs:

- MinAccel -- The minimum possible acceleration at the given velocity.

ArmFF_MinAchieveVelocity



Calculates the minimum achievable velocity given a maximum voltage supply, a position, and an acceleration. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the acceleration constraint, and this will give you a simultaneously-achievable velocity constraint.

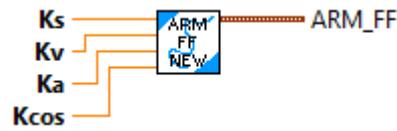
Inputs:

- ArmFF -- The ArmFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the arm.
- angle -- The angle of the arm.
- acceleration -- The acceleration of the arm.

Outputs:

- MinVelocity -- The minimum possible velocity at the given acceleration and angle.

ArmFF_New



Creates a new ArmFeedforward data cluster with the specified gains. Units of the gain values will dictate units of the computed feedforward.

This set of subVIs computes feedforward outputs for a simple arm (modeled as a motor acting against the force of gravity on a beam suspended at an angle).

Inputs:

- ks -- The static gain.
- kv -- The velocity gain.
- ka -- The acceleration gain.
- kcos -- The gravity gain.

Outputs:

- ArmFF -- The initialized ArmFF data cluster

ArmFF_New_ZeroGravity



Creates a new ArmFeedforward data cluster with the specified gains. Acceleration gain is defaulted to zero. Units of the gain values will dictate units of the computed feedforward.

This set of subVIs computes feedforward outputs for a simple arm (modeled as a motor acting against the force of gravity on a beam suspended at an angle).

Inputs:

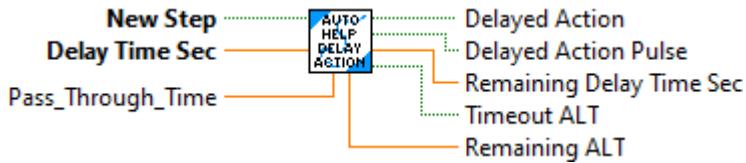
- ks -- The static gain.
- kv -- The velocity gain.
- ka -- The acceleration gain.
- kcos -- The gravity gain.

Outputs:

- ArmFF -- The initialized ArmFF data cluster

AutoHelper

AutoHelper_DelayedAction



Implements a simplified state machine similar to a mechanical / electrical drum sequencer. When the Drum Sequencer is "Ready" and the "Initiate" input becomes TRUE,, the drum sequences through its set of steps from 1 to N. After each Step is executed it is followed by a wait. During the wait the Step output indicates "Do Nothing". The number of Steps is determined by the number of elements in the "Step Times" input array. Once all the steps have been executed, the Drum Sequencer cycles bac to wait for another initiate.

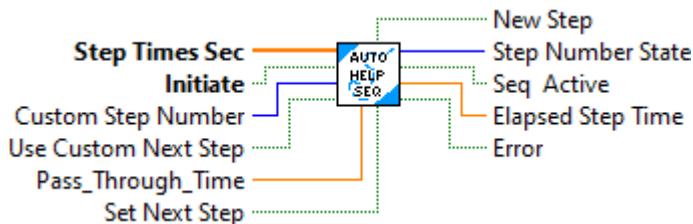
There is a Cancel input that interrupts the sequence, executes the Cancel Step, performs the cancel wait, and then returns to wait for an Initiate action.

If any of the Step Times, or Cancel times are negative, or there more than 32 steps, or less than 1 step an error is signaled. The drum sequencer selects the error state and stays there.

Inputs:

Outputs:

AutoHelper_Sequence_Execute



Implements a simplified state machine similar to a mechanical / electrical drum sequencer. When the Drum Sequencer is "Ready" and the "Initiate" input becomes TRUE,, the drum sequences through its set of steps from 1 to N. After each Step is executed it is followed by a wait. During the wait the Step output indicates "Do Nothing". The number of Steps is determined by the number of elements in the "Step Times" input array. Once all the steps have been executed, the Drum Sequencer cycles bac to wait for another initiate.

There is a Cancel input that interrupts the sequence, executes the Cancel Step, performs the cancel wait, and then returns to wait for an Initiate action.

If any of the Step Times, or Cancel times are negative, or there more than 32 steps, or less than 1 step an error is signaled. The drum sequencer selects the error state and stays there.

Inputs:

Outputs:

BangBang

BangBang_AtSetpoint



Returns true if the error is within the tolerance of the setpoint.

Input:

- BangBang -- Data cluster

Output

- AtSetpoint -- Returns TRUE if within error tolerance.

BangBang_Calculate_PV



Returns the calculated control output.

Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

Input:

- BangBang -- Data cluster
- measurement -- The most recent measurement of the process variable.
- setpoint -- The setpoint for the process variable.

Output

- outBangBang -- Updated data cluster
- ControlOutput -- The calculated motor output (0 or 1).

BangBang_Calculate_SP_PV



Returns the calculated control output.

Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

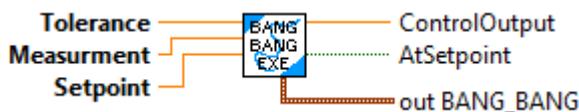
Input:

- BangBang -- Data cluster
- measurement -- The most recent measurement of the process variable.
- setpoint -- The setpoint for the process variable.

Output

- outBangBang -- Updated data cluster
- ControlOutput -- The calculated motor output (0 or 1).

BangBang_Execute



Implements and executes bang-bang controller, which outputs either 0 or 1 depending on whether the measurement is less than the setpoint. This maximally-aggressive control approach works very well for velocity control of high-inertia mechanisms, and poorly on most other things.

Note that this is an **asymmetric** bang-bang controller - it will not exert any control effort in the reverse direction (e.g. it won't try to slow down an over-speeding shooter wheel). This asymmetry is **extremely**

important.* Bang-bang control is extremely simple, but also potentially hazardous. Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

Inputs:

- Tolerance -- Tolerance for "at setpoint"
- measurement -- The most recent measurement of the process variable.
- setpoint -- The setpoint for the process variable.

Outputs:

- ControlOutput -- The calculated motor output (0 or 1).
- AtSetpoint -- Returns TRUE if within error tolerance.
- BangBang -- Data cluster Returns the calculated control output.

BangBang_GetAll



Gets elements of the internal data cluster

Input:

- BangBang -- Data cluster

Output

- Setpoint -- Current stored setpoint (SP)
- Measurement -- Current stored process variable (PV)
- Tolerance -- Current stored tolerance.

BangBang_GetError



Returns current error, which is Setpoint - Process Variable

Input:

- BangBang -- Data cluster

Output:

- Error -- Current error value.

BangBang_New



Implements a bang-bang controller, which outputs either 0 or 1 depending on whether the measurement is less than the setpoint. This maximally-aggressive control approach works very well for velocity control of high-inertia mechanisms, and poorly on most other things.

Note that this is an **asymmetric** bang-bang controller - it will not exert any control effort in the reverse direction (e.g. it won't try to slow down an over-speeding shooter wheel). This asymmetry is **extremely important.** Bang-bang control is extremely simple, but also potentially hazardous. Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

Creates a new bang-bang controller.

Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

Inputs:

- Tolerance -- Tolerance for "at setpoint"

Outputs:

- BangBang -- Data cluster
-

BangBang_SetSetpoint



Sets the setpoint for the bang-bang controller.

Input:

- BangBang -- Data cluster
- Setpoint -- The desired setpoint.

Output

- outBangBang -- Updated data cluster
-

BangBang_SetTolerance



Sets the error within which atSetpoint will return true.

Input:

- BangBang -- Data cluster
- tolerance -- Position error which is tolerable.

Output

- outBangBang -- Updated data cluster

BatterySim

BatterySim_CalculateDefaultBatteryLoadedVoltage



Calculate the loaded battery voltage. This function assumes a nominal voltage of 12v and a resistance of 20 milliohms (0.020 ohms)

Inputs:

- currents -- An array of currents drawn from the battery.

Outputs:

- Voltage -- The battery's voltage under load.

BatterySim_CalculateLoadedBatteryVoltage



Calculate the loaded battery voltage.

Inputs:

- nominalVoltage -- The nominal battery voltage. Usually 12v.
- resistanceOhms -- The forward resistance of the battery. Most batteries are at or below 20 milliohms.
- currents -- The array of currents drawn from the battery.

Outputs:

- Voltage -- The battery's voltage under load.

BatterySim_Execute



LabVIEW single call function to simulate loaded battery voltage.

Inputs:

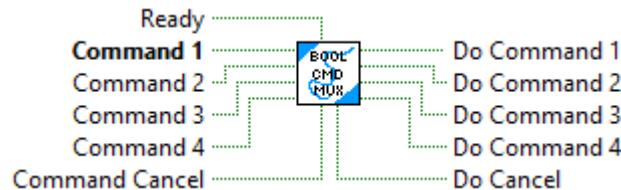
- current -- double -- sum of all currents connected to this battery (amps)
- nominalVoltage -- double -- The nominal battery voltage. (Volts) Usually 12v. (Default: 12)
- Battery Resistance -- double -- The forward resistance of the battery. Most batteries are at or below 20 milliohms. (Ohms) (Default: 0.03)

Outputs:

- Voltage -- The battery's voltage under load.

BoolCmd

BoolCmd_Multiplexor



BoolCmd_Multiplexor_Array



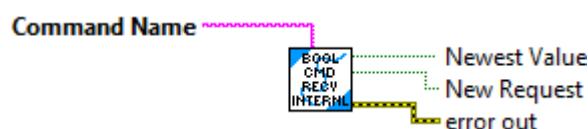
BoolCmd_ObtainQueue



BoolCmd_Recv

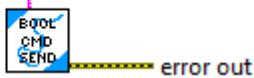


BoolCmd_Recv_Internal



BoolCmd_Send

Command Name 



BoolCmd_Send_Internal

Command Name 



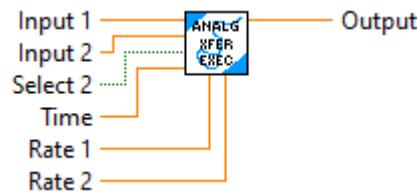
BoolCmd_Send_OnEdge

Command Name 



BumplessTransfer

BumplessTransfer_Execute



This function selects one of two inputs depending on the state of a third boolean input. The output will equal the selected input. When switching between input values the difference between the two inputs values at the time of the switched is ramped to zero at the rate defined by the input parameters. This provides a bumpless transfer function.

Inputs:

- Input 1 -- double float -- Input Value 1
- Input 2 -- double float -- Input value 2
- Select 2 -- boolean -- When true Input Value 2 is selected. When false Input Value 1 is selected.
- Rate 1 -- double float -- Rate (units/sec) used when transferring to Input 1 to remove the difference between inputs. If the value is zero, the value is switched immediately.
- Rate 2 -- double float -- Rate (units/sec) used when transferring to Input 2 to remove the difference between inputs. If the value is zero, the value is switched immediately.

Outputs:

- Output -- double float -- Selected value.

CallbackHelp

CallbackHelp_MatrixMinus



matrix callback function that performs

$$A - B$$

Inputs:

- ExtraData -- Variant containing extra data -- not used
- A -- A matrix
- B -- B matrix

Outputs:

- Output -- Resulting matrix
- Error -- If TRUE, an error occurred.

CallbackHelp_MatrixMult



matrix callback function that performs

$$A \times B$$

Inputs:

- ExtraData -- Variant containing extra data -- not used
- A -- A matrix

- B -- B matrix

Outputs:

- Output -- Resulting matrix
- Error -- If TRUE, an error occurred.

CallbackHelp_MatrixMult_CoerceSizeB



matrix callback function that performs

$$A \times B$$

The size of the B matrix is coerced to ensure the multiply succeeds.

Inputs:

- ExtraData -- Variant containing extra data -- not used
- A -- A matrix
- B -- B matrix

Outputs:

- Output -- Resulting matrix
- Error -- If TRUE, an error occurred.

CallbackHelp_MatrixPlus



matrix callback function that performs

A + B**Inputs:**

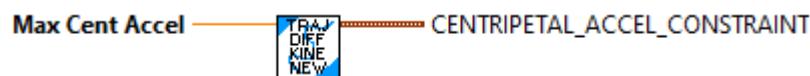
- ExtraData -- Variant containing extra data -- not used
- A -- A matrix
- B -- B matrix

Outputs:

- Output -- Resulting matrix
- Error -- If TRUE, an error occurred.

CentripetalAccelConstraint

CentripetalAccelConstraint_New



Constructs a centripetal acceleration constraint.

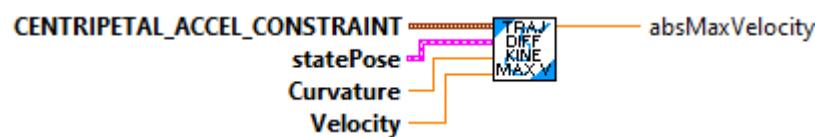
Parameters:

- MaxCentripitalAccel - Maximum Centripetal acceleration (meters/sec²)

Returns

- CentripetalAccelConstraint - Constraint data structure.

CentripetalAccelConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

Calculation comments.

```

// ac = v^2 / r
// k (curvature) = 1 / r
// therefore, ac = v^2 * k
// ac / k = v^2
// v = std::sqrt(ac / k)
  
```

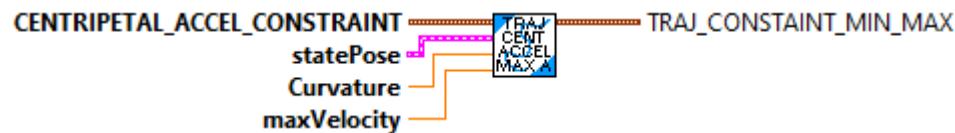
Parameters:

- CentripetalAccelConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

CentripetalAccelConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values. Note says. The acceleration of the robot has no impact on the centripetal acceleration of the robot.

Parameters:

- CentripetalAccelConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

ChassisSpeeds

ChassisSpeeds_FromFieldRelativeChassisSpeeds



Converts a user provided field-relative ChassisSpeeds object into a robot-relative ChassisSpeeds object.

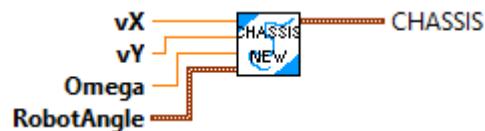
Parameters:

- fieldRelativeSpeeds -- The ChassisSpeeds object representing the speeds in the field frame of reference. Positive x is away from your alliance wall. Positive y is to your left when standing behind the alliance wall.
- robotAngle -- The angle of the robot as measured by a gyroscope. The robot's angle is considered to be zero when it is facing directly away from your alliance station wall. Remember that this should be CCW positive.

Returns:

- ChassisSpeeds - Data structure representing the speeds in the robot's frame of reference.

ChassisSpeeds_FromFieldRelativeSpeeds



Converts a user provided field-relative set of speeds into a robot-relative ChassisSpeeds object.

Parameters:

- VX - The component of speed in the x direction relative to the field. Positive x is away from your alliance wall. (Meters/sec)

- VY - The component of speed in the y direction relative to the field. Positive y is to your left when standing behind the alliance wall. (Meters/sec)

-omega - The angular rate of the robot. (Radians/Sec)

-robotAngle - The angle of the robot as measured by a gyroscope. The robot's angle is considered to be zero when it is facing directly away from your alliance station wall. Remember that this should be CCW positive. (radians)

Returns:

- ChassisSpeeds - Data structure representing the speeds in the robot's frame of reference.

ChassisSpeeds_GetXYOmega



Retrieve individual components from a Chassis data cluster

Parameters:

- Chassis - CHASSIS_SPEEDS Data structure

Returns:

- vx - Forward velocity. (Meters/Second)

- vy - Sideways velocity. (Meters/Second)

- omega - Angular velocity. (Radians/Second)

ChassisSpeeds_New



Constructs a ChassisSpeeds object.

Represents the speed of a robot chassis. Although this struct contains similar members compared to a Twist2d, they do NOT represent the same thing. Whereas a Twist2d represents a change in pose w.r.t to the robot frame of reference, this ChassisSpeeds struct represents a velocity w.r.t to the robot frame of reference.

A strictly non-holonomic drivetrain, such as a differential drive, should never have a dy component because it can never move sideways. Holonomic drivetrains such as swerve and mecanum will often have all three components.

Parameters:

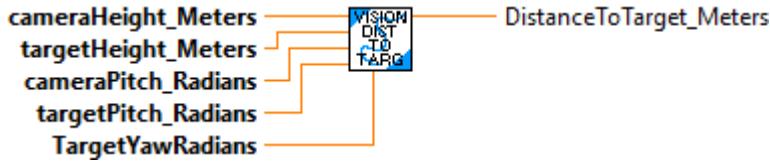
- vx - Forward velocity. (Meters/Second)
- vy - Sideways velocity. (Meters/Second)
- omega - Angular velocity. (Radians/Second)

Returns:

- Chassis - CHASSIS_SPEEDS Data structure

CompVisionUtil

CompVisionUtil_CalculateDistanceToTarget



Algorithm from https://docs.limelightvision.io/en/latest/cs_estimating_distance.html. Estimates range to a target using the target's elevation. This method can produce more stable results than SolvePNP when well tuned, if the full 6d robot pose is not required. Note that this method requires the camera to have 0 roll (not be skewed clockwise or CCW relative to the floor), and for there to exist a height differential between goal and camera. The larger this differential, the more accurate the distance estimate will be.

Units can be converted using the conversion functions.

Inputs:

- CameraHeight_Meters -- The physical height of the camera off the floor
in meters.
- TargetHeightMeters -- The physical height of the target off the floor in meters.

This should be the height of whatever is being targeted (i.e. if the targeting region is set to top, this should be the height of the top of the target).
- CameraPitch_Radians -- The pitch of the camera from the horizontal plane
in radians.

Positive values up.
- TargetPitchRadian -- The pitch of the target in the camera's lens in radians.

Positive values up.
- TargetYawRadians -- The yaw of the target in the camera's lens in radians.

Outputs

- DistanceToTarget_Meters -- The estimated distance to the target in meters.
-

CompVisionUtil_EstimateCameraToTarget



Estimates a Transform2d that maps the camera position to the target position, using the robot's gyro. Note that the gyro angle provided ***must*** line up with the field coordinate system -- that is, it should read zero degrees when pointed towards the opposing alliance station, and increase as the robot rotates CCW.

Inputs:

- CameraToTargetTranslation -- A Translation3d that encodes the x/y position of the target relative to the camera.
- FieldToTarget -- A Pose3d representing the target position in the field coordinate system.
- GyroAngle -- A rotation2d representing the current robot gyro angle, likely from odometry.

Outputs:

- EstimateCameraToTarget -- A Transform3d that takes us from the camera to the target.
-

CompVisionUtil_EstimateFieldToCamera



Estimates the pose of the camera in the field coordinate system, given the position of the target relative to the camera, and the target relative to the field. This ***only*** tracks the position of the camera, not the position of the robot itself.

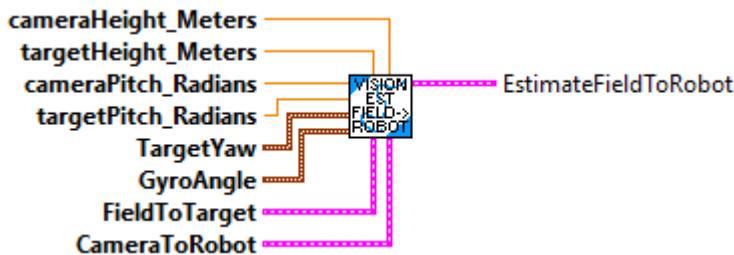
Inputs:

- CameraToTarget -- Transform3d containing the position of the target relative to the camera.
- FieldToTarget -- Pose3d containing the position of the target in the field.

Output:

- EstimateFieldToCamera -- Pose3d containing position of the camera in the field.

CompVisionUtil_EstimateFieldToRobot



Estimate the position of the robot in the field.

Inputs:

- CameraHeightMeters -- The physical height of the camera off the floor in meters.
- TargetHeightMeters -- The physical height of the target off the floor in meters.

This should be the height of whatever is being targeted (i.e. if the targeting region is set to top, this should be the height of the top of the target).

- CameraPitchRadians -- The pitch of the camera from the horizontal plane in radians. Positive values up.
- TargetPitchRadians -- The pitch of the target in the camera's lens in radians. Positive values up.
- TargetYaw -- Rotation2d representing the observed yaw of the target. Note that this *must* be CCW-positive, and Photon returns CW-positive.
- GyroAngle -- Rotation2d representing the current robot gyro angle, likely from odometry.
- FieldToTarget -- A Pose3d representing the target position in the field coordinate

system.

- CameraToRobot -- The Transform3d position of the robot relative to the camera. If the camera was mounted 3 inches behind the "origin" (usually physical center) of the robot, this would be Transform3d (3 inches, 0 inches, 0 inches, 0 degrees).
(Make certain to convert from inches to meters!)

Outputs

- EstimateFieldToRobot -- The Pose3d position of the robot in the field.

CompVisionUtil_EstimateFieldToRobot_Alt



Estimates the pose of the robot in the field coordinate system, given the position of the target relative to the camera, the target relative to the field, and the robot relative to the camera.

Inputs:

- CameraToTarget -- Transform3d containing the position of the target relative to the camera.
- FieldToTarget -- Pose3d containing the position of the target in the field.
- CameraToRobot -- Transform3d containing the position of the robot relative to the camera. If the camera was mounted 3 inches behind the "origin" (usually physical center) of the robot, this would be Transform2d(3 inches, 0 inches, 0 inches, 0 degrees).

Output:

- EstimatedFieldToRobot -- A Pose3d containing the position of the robot in the field.

CompVisionUtil_ObjectToRobotPose



Returns the robot's pose in the field coordinate system given an object's field-relative pose, the transformation from the camera's pose to the object's pose (obtained via computer vision), and the transformation from the robot's pose to the camera's pose.

The object could be a target or a fiducial marker.

Inputs:

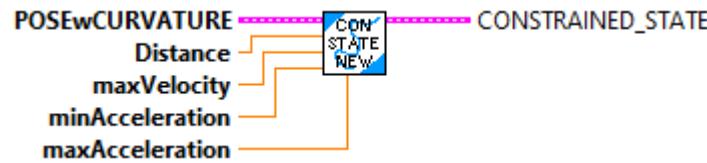
- ObjectInField -- Pose3d -- An object's field-relative pose.
- CameraToObject -- Transform3d -- The transformation from the camera's pose to the object's pose.
This comes from computer vision.
- RobotToCamera -- Transform3d -- The transformation from the robot's pose to the camera's pose.
This can either be a constant for a rigidly mounted camera, or variable if the camera is mounted to
a turret. If the camera was mounted 3 inches in front of the "origin" (usually physical center) of the robot,
this would be new Transform3d(Units.inchesToMeters(3.0), 0.0, 0.0,
new Rotation3d()).

Returns:

- ObjectToRobotPose -- Pose3d -- The robot's field-relative pose.

ConstrainedState

ConstrainedState_New



Internal routine used only by TrajectoryParam_timeParam.

ConstrainedState_SetMaxAccel



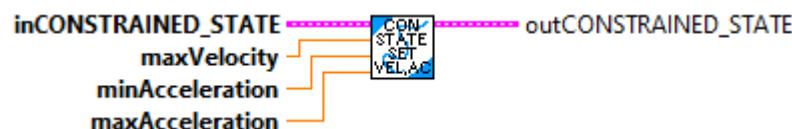
Internal routine used only by TrajectoryParam_timeParam.

ConstrainedState_SetMinAccel



Internal routine used only by TrajectoryParam_timeParam.

ConstrainedState_SetVelAccel



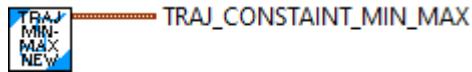
ConstrainedState_SetVelocity



Internal routine used only by TrajectoryParam_timeParam.

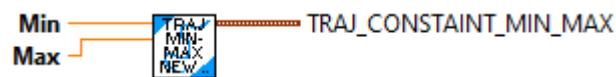
Constraint

Constraint_MinMax_New



Constructs a MinMax with default values.

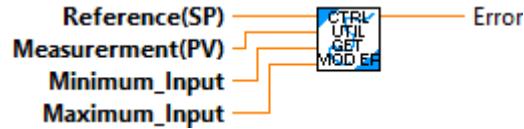
Constraint_MinMax_NewMinMax



Constructs a MinMax with provided values

ControllerUtil

ControllerUtil_GetModulusError



Returns modulus of error where error is the difference between the reference and a measurement.

This is used for continuous systems, for example rotational systems.

Inputs:

- Reference (SP) -- Reference input of a controller
- Measurement (PV) -- The current measurement.
- MinimumInput -- The minimum value expected from the input.
- MaximumInput -- The maximum value expected from the input.

Outputs:

- Error -- Error value of a continuous system.

Conv

Conv_AngleDegrees_Heading



Convert angle (degrees) to heading (+/- 180) (degrees)

NOTE: increases counter clockwise

Conv_AngleRadians_Heading



Convert angle (radians) to heading (+/- PI) (radians)

NOTE: increases counter clockwise

Conv_Centimeters_Meters



Convert Centimeters to Meters

Conv_Deg_Radians



Convert Degrees to Radians

Conv_Deg_Rotations



Convert Degrees to Rotations

Conv_Feet_Meters



Convert Feet to Meters

Conv_GyroDegrees_Heading



Convert angle (radians) to heading (+/- PI) (radians)

NOTE: gyro increments clockwise. Headings increment counterclockwise. (NOTE: Be careful to check the rotation of your gyro before using this routine to adjust it.)

Conv_Heading_AngleRadians



This routine does nothing!

Conv_Inches_Meters



Convert Inches to Meters

Conv_Kilograms_Pounds



Convert mass in kilograms to pounds.

Input:

- kilograms -- mass in kilograms

Outputs:

- pounds -- mass in pounds
-
-

Conv_Meters_Feet



Convert Meters to Feet

Conv_Meters_Inches



Convert Meters to Inches

Conv_POSE2D_Eng_SI



Convert POSE from Meters, Radians to Feet, Degrees. (This is really only for external use. All the internal routines use meters and radians.)

Conv_POSE2D_SI_Eng



Convert POSE from Meters, Radians to Feet, Degrees. (This is really only for external use. All the internal routines use meters and radians.)

Conv_Pounds_Kilograms



Convert mass in pounds to kilograms

Input:

- pounds -- mass in pounds

Output:

- kilograms -- mass in kilograms

Conv_Radians_Deg



Convert Radians to Degrees

Conv_Radians_Rotations



Convert Radians to Rotations

Conv_Rotations_Degrees



Convert Rotations to Degrees

Conv_Rotations_Radians



Convert Rotations to Radians

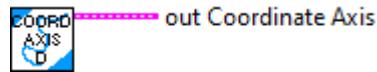
Conv_Yards_Meters



Convert Yards to Meters

CoordAxis

CoordAxis_D



Returns a coordinate axis corresponding to -Z in the NWU coordinate system. (Down)

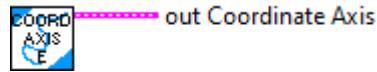
Inputs:

-- none --

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.
-
-

CoordAxis_E



Returns a coordinate axis corresponding to -Y in the NWU coordinate system. (East)

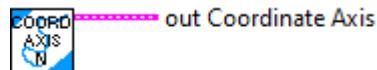
Inputs:

-- none --

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.
-
-

CoordAxis_N



Returns a coordinate axis corresponding to +X in the NWU coordinate system. (North)

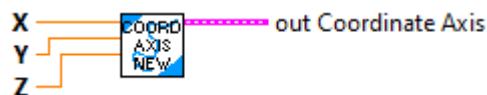
Inputs:

-- none --

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.

CoordAxis_New



Constructs a coordinate system axis within the NWU coordinate system and normalizes it.

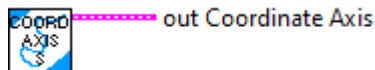
Inputs:

- X -- The x component.
- Y -- The y component.
- Z -- The z component.

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.

CoordAxis_S



Returns a coordinate axis corresponding to -X in the NWU coordinate system. (South)

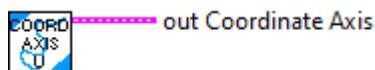
Inputs:

-- none --

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.
-

CoordAxis_U



Returns a coordinate axis corresponding to +Z in the NWU coordinate system. (Up)

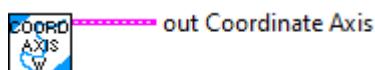
Inputs:

-- none --

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.
-

CoordAxis_W



Returns a coordinate axis corresponding to +Y in the NWU coordinate system. (West)

Inputs:

-- none --

Outputs:

- Coordinate Axis -- A data cluster representing a coordinate system axis with respect to the NWU coordinate system.

CoordSystem

CoordSystem_Convert_Pose3d



Converts the given pose from one coordinate system to another.

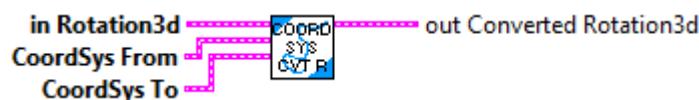
Inputs:

- in pose3d -- The pose to convert.
- Coord Sys from -- The coordinate system the pose starts in.
- Coord Sys to -- The coordinate system to which to convert.

Returns:

- Converted Pose -- The given pose in the desired coordinate system.

CoordSystem_Convert_Rotation3d



Converts the given rotation from one coordinate system to another.

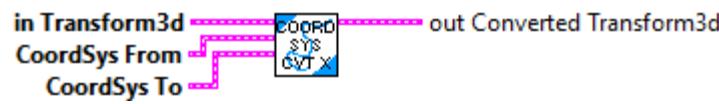
Inputs:

- in rotation3d -- The rotation to convert.
- Coord Sys from -- The coordinate system the pose starts in.
- Coord Sys to -- The coordinate system to which to convert.

Returns:

- Converted Rotation -- The given rotation in the desired coordinate system.

CoordSystem_Convert_Transform3d



Converts the given transform3d from one coordinate system to another.

Inputs:

- in transform3d -- The transform3d to convert.
- Coord Sys from -- The coordinate system the pose starts in.
- Coord Sys to -- The coordinate system to which to convert.

Returns:

- Converted transform3d -- The given transform3d in the desired coordinate system.

CoordSystem_Convert_Translation3d



Converts the given translation from one coordinate system to another.

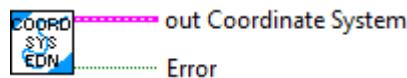
Inputs:

- in translation3d -- The translation to convert.
- Coord Sys from -- The coordinate system the pose starts in.
- Coord Sys to -- The coordinate system to which to convert.

Returns:

- Converted Translation -- The given translation in the desired coordinate system.

CoordSystem_EDN



Returns an instance of the East-Down-North (EDN) coordinate system.

The +X axis is east, the +Y axis is down, and the +Z axis is north.

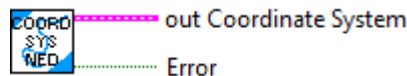
Inputs::

-- none --

Returns:

-- out Coordinate System -- Data cluster containing the Coordinate System
-- Error -- TRUE indicates an error occurred.

CoordSystem_NED



Returns an instance of the North-East-Down (NED) coordinate system.

The +X axis is north, the +Y axis is east, and the +Z axis is down.

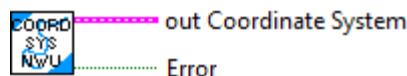
Inputs::

-- none --

Returns:

-- out Coordinate System -- Data cluster containing the Coordinate System
-- Error -- TRUE indicates an error occurred.

CoordSystem_NWU



Returns an instance of the North-West-Up (NWU) coordinate system.

The +X axis is north, the +Y axis is west, and the +Z axis is up.

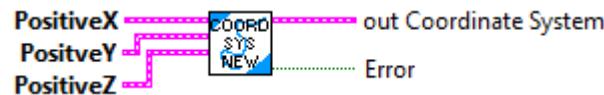
Inputs::

-- none --

Returns:

-- out Coordinate System -- Data cluster containing the Coordinate System
-- Error -- TRUE indicates an error occurred.

CoordSystem_New



Construct a Coordinte System data cluster which converts Pose3d objects between different standard coordinate frames. Contains a coordinate system with the given cardinal directions for each axis.

Inputs::

-- PositiveX -- The cardinal direction of the positive x-axis.
-- PositiveY -- The cardinal direction of the positive y-axis.
-- PositiveZ -- The cardinal direction of the positive z-axis.

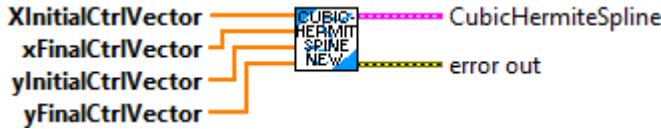
Returns:

-- out Coordinate System -- Data cluster containing the Coordinate System

-- Error -- TRUE indicates an error occurred.

CubicHermiteSpline

CubicHermiteSpline_New



Constructs a cubic hermite spline with the specified control vectors. Each control vector contains info about the location of the point and its first derivative.

Parameters:

- xInitialControlVector - The control vector for the initial point in the x dimension.
- xFinalControlVector - The control vector for the final point in the x dimension.
- yInitialControlVector - The control vector for the initial point in the y dimension.
- yFinalControlVector - The control vector for the final point in the y dimension.

Returns:

- CubicHermiteSpline - Spline data structure
- Error Out - Error data structure

CubicHermiteSpline_getControlVectorFromArrays



Returns the control vector for each dimension as a matrix from the user-provided arrays in the constructor.

Parameters:

- initialVector - The control vector for the initial point.
- finalVector - The control vector for the final point.

Returns:

- ControlVector - The control vector matrix for a dimension.
 - Error Out - The output error cluster
-
-

CubicHermiteSpline_makeHermiteBasis



Returns the hermite basis matrix for cubic hermite spline interpolation.

Parameters:

- none -

Returns:

- HermiteBasis - The hermite basis matrix for cubic hermite spline interpolation.

DCMotor

DCMotor_GetAndymark9015



Creates a new DC MOTOR data cluster with the specified values. for a set of AndyMark 9015 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetAndymarkAM2235A



Creates a new DC MOTOR data cluster with the specified values. for a set of AndyMark 9015 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetAndymarkAM3493



Creates a new DC MOTOR data cluster with the specified values. for a set of AndyMark 9015 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
-

DCMotor_GetAndymarkRs775_125



Creates a new DC MOTOR data cluster with the specified values. for a set of AndyMark Rs775-125 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
-

DCMotor_GetBag



Creates a new DC MOTOR data cluster with the specified values. for a set of BAG motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetBanebotsRs550



Creates a new DC MOTOR data cluster with the specified values. for a set of BaneBots RS550 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetBanebotsRs775



Creates a new DC MOTOR data cluster with the specified values. for a set of BaneBots RS775 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
-

DCMotor_GetCIM



Creates a new DC MOTOR data cluster with the specified values. for a set of CIM motors.

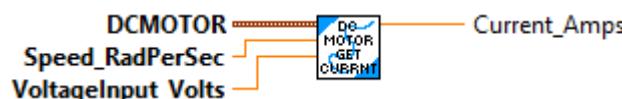
Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
-

DCMotor_GetCurrent



Estimate the current for this motor at the given speed and voltage.

Inputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
- Speed -- Current motor speed (Radians/Second).
- Voltage -- The current motor input voltage (Volts)

Outputs:

- Stall Current -- Stall current (Amps).

DCMotor_GetFalcon500



Creates a new DC MOTOR data cluster with the specified values. for a set of Falcon 500 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetMiniCIM



Creates a new DC MOTOR data cluster with the specified values. for a set of Mini-CIM motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetNEO



Creates a new DC MOTOR data cluster with the specified values. for a set of NEO motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
-

DCMotor_GetNEO550



Creates a new DC MOTOR data cluster with the specified values. for a set of NEO 550 motors.

Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
-

DCMotor_GetRomiBuiltIn



Creates a new DC MOTOR data cluster with the specified values. for a set of Romi Built In motors.

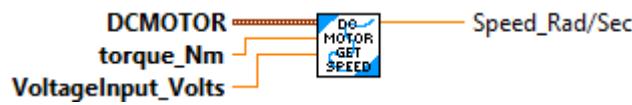
Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_GetSpeed



Calculate the speed of the mtor given torque and voltage input.

Inputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
- Torque -- The torque produced by the motor. (NM)
- Voltage -- The voltage applied to the motor. (Volts)

Outputs:

- Speed -- double -- Motor speed (Rad/Sec)

DCMotor_GetTorque



Estimate the torque for this motor at the given current

Inputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

- Current -- Motor current (Amps)

Outputs:

- Torque -- Motor torque (NM)

DCMotor_GetVex775Pro



Creates a new DC MOTOR data cluster with the specified values. for a set of Vex 775 PRO motors.

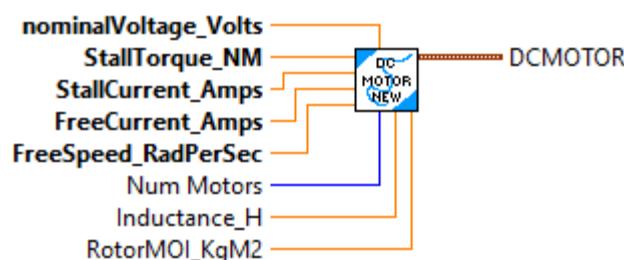
Inputs:

- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_New



Creates a new DC MOTOR data cluster for a custom motor with the specified values.

Inputs:

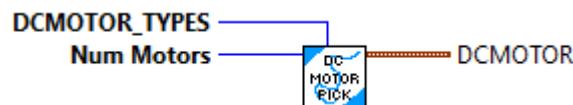
- Nominal Voltage -- The nominal voltage used for obtaining these parameters. Usually 12. (Volts)

- Stall Torque -- Stall torque (Newton Meters).
- Stall Current -- Stall current (Amps).
- Free Current -- Free current - no attached mechanical load (Amps).
- Free Speed -- Free speed - no attached mechanical load (Radians/Second).
- Num Motors -- The number of motors used together in this system driving the same load.
- Inductance -- Motor inductance. Not used in "standard" motor models (Henrys).
- Rotor MOI -- The motors moment of inertial. (Kilogram Meters squared)

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_PickMotor



Creates a new DC MOTOR data cluster for a custom motor of the chosen type..

Inputs:

- DC Motor Type -- The model of the motor.
- Num Motors -- The number of motors used together in this system driving the same load.

Outputs:

- DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.

DCMotor_WithReduction



Returns a copy of this motor with the given gearbox reduction applied.

Inputs:

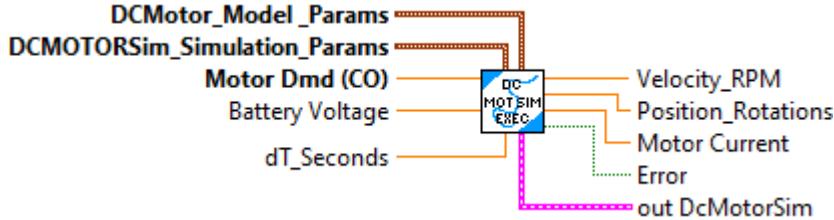
- in DCMOTOR -- Cluster containing the motor modeling constants for this set of motors.
- GearboxReduction -- double -- The gearbox reduction.

Outputs:

- out DCMOTOR -- Revised cluster containing the motor modeling constants for this set of motors.

DcMotorSim

DcMotorSim_Execute



Single call LabVIEW function to simulate a DC Motor system. This simulates both velocity and position.

The linear system for this is:

- States
 - angular position
 - , angular velocity
- Inputs
 - voltage
- Outputs
 - angular position
 - angular velocity

Inputs:

- DCMotor Model Params -- cluster -- Contains physical configuration for DC Motor system.
- DCMotor Sim Simulation Params -- cluster -- Contains siulation configuration.
- Motor Dmd (CO) -- double -- motor demand value (+/- 1.0)
- Battery Voltage -- double -- Current simulated battery voltage (Volts)
- dT Sec -- double -- Update period (Seconds, Default: 0.02)

Outputs:

- Velocity -- double -- Current motor velocity (RPM)

- Position -- double -- Current motor position (Rotations)
- Motor Current -- double -- Current motor current (Amps)
- Error -- boolean -- If TRUE an error occurred.
- outDCMotorSim -- cluster -- Current internal data cluster.

Debouncer

Debouncer_Calculate



Applies the debouncer to the input stream.

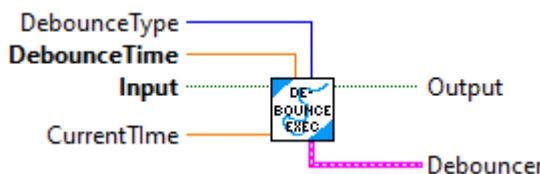
Inputs:

- DebounceIn -- Debounce data cluster
- input -- The current value of the input stream.
- CurrentTime -- (Optional - Default: read system timer) The current system elapsed time.

Outputs:

- DebounceOut -- Updated data cluster
- output -- The debounced value of the input stream.
- CurrentTimeOut -- The current system time. If not read and not supplied, value will be -1.0.

Debouncer_Execute



A simple debounce filter for boolean streams. Requires that the boolean change value from baseline for a specified period of time before the filtered value changes.

This is a LabVIEW style single function call.

Inputs:

- DebounceType -- Which type of state change the debouncing will be performed on. (Optional: Default: Rising)
- DebounceTime -- The number of seconds the value must change from baseline for the filtered value to change.
- Input -- Boolean input value.
- CurrentTime -- Current system running time. (Optional - Default: Read system time.)

Output:

- Output -- Value of debounced input.
- Debouncer -- Initialized data cluster for debouncer.

Debouncer_HasElapsed



Internal function

See if debounce time has elapsed.

Debouncer_New



A simple debounce filter for boolean streams. Requires that the boolean change value from baseline for a specified period of time before the filtered value changes.

Creates a new Debouncer.

Inputs:

- DebounceTime -- The number of seconds the value must change from baseline for the filtered value to change.

- DebounceType -- Which type of state change the debouncing will be performed on. (Optional: Default: Rising)

- CurrentTime -- Current system running time. (Optional - Default: Read system time.)

Output:

- Debouncer -- Initialized data cluster for debouncer.

Debouncer_Reset



This is an internal function.

Resets previous time.

DiffDriveKinematicsConstraint

DiffDriveKinematicsConstraint_New



Constructs a differential drive dynamics constraint.

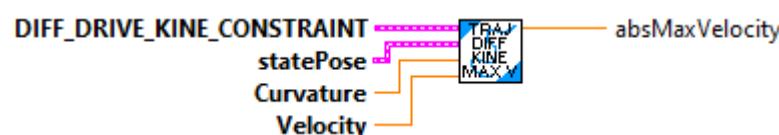
Parameters:

- MaxSpeed - Max speed (meters/sec)
- DiffDriveKinematics - Diff Drive Kinematics data structure

Returns

- DiffDriveKineConstraint - Constraint data structure.

DiffDriveKinematicsConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

Ensure each individual normalized wheel speed is within the defined maximum velocity.

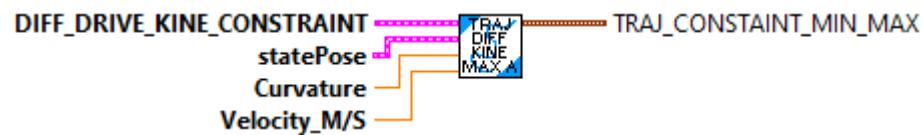
Parameters:

- DiffDriveKinematicsConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.
-

DiffDriveKinematicsConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values.

Parameters:

- DiffDriveKinematicsConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

DiffDrivePoseEst

DiffDrivePoseEst_AddVisionMeasurement



Add a vision measurement to the Unscented Kalman Filter. This will correct the odometry pose estimate while still accounting for measurement noise.

This method can be called as infrequently as you want, as long as you are calling `DifferentialDrivePoseEstimator_update` every loop.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

Inputs:

- `InDiffDrivePoseEstimate` -- Data cluster for this system
- `visionRobotPoseMeters` -- The pose of the robot as measured by the vision camera.
- `timestampSeconds` -- The timestamp of the vision measurement in seconds. Note that if you don't use your own time source by calling `DifferentialDrivePoseEstimator_updateWithTime` then you must use a timestamp with an epoch since FPGA startup (i.e. the epoch of this timestamp is the same epoch as `Timer.getFPGATimestamp`.) This means that you should use `Timer.getFPGATimestamp` as your time source in this case.

Outputs:

- `OutDiffDrivePoseEstimate` -- Data cluster for this system
- `Error` -- Returns TRUE if an error occurred.

DiffDrivePoseEst_FillStateVector



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Create a vector with the current robot pose, encoder distances, and gyro

Inputs:

- initialPose -- robot pose
- LeftDist -- left encoder distance
- RightDist -- right encoder distance

Outputs:

- output Matrix -- filled matrix.

DiffDrivePoseEst_GetEstimatedPosition



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Gets the pose of the robot at the current time as estimated by the Unscented Kalman Filter.

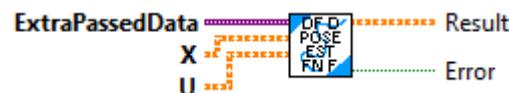
Inputs:

- DiffDrivePoseEst - System data cluster

Outputs:

- EstimatedPose - The estimated robot pose in meters.

DiffDrivePoseEst_Kalman_F_Callback



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Internal function used by DiffDrivePoseEst_Update routines to pass to the Kalman filter Predict routine.

Inputs:

- ExtraPassedData -- Variant of extra data used by this function. (This value is not used.)
- X -- X matrix
- U -- U matrix

Outputs:

- Result -- Resulting matrix
 - Error -- If TRUE, an error occurred.
-
-

DiffDrivePoseEst_Kalman_H_Callback



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Internal function used by DiffDrivePoseEst_Update routines to pass to the Kalman filter Correct routine.

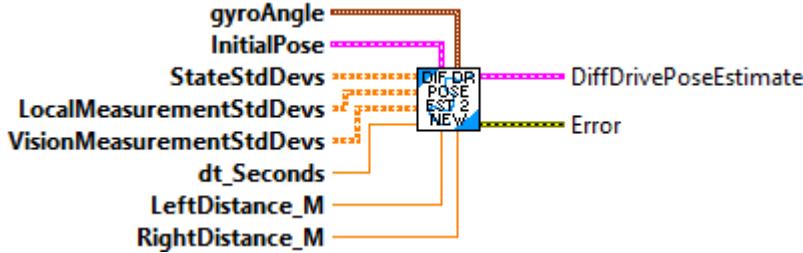
Inputs:

- ExtraPassedData -- Variant of extra data used by this function. (This value is not used.)
- X -- X matrix
- U -- U matrix

Outputs:

- Result -- Resulting matrix
 - Error -- If TRUE, an error occurred.
-
-

DiffDrivePoseEst_New



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

This set of subVI wraps an UnscentedKalmanFilter to fuse latency-compensated vision measurements with differential drive encoder measurements. It will correct for noisy vision measurements and encoder drift. It is intended to be an easy drop-in for {@link edu.wpi.first.math.kinematics.DifferentialDriveOdometry}; in fact, if you never call DifferentialDrivePoseEstimator_addVisionMeasurement and only call DifferentialDrivePoseEstimator_update then this will behave exactly the same as DifferentialDriveOdometry.

DifferentialDrivePoseEstimator_update} should be called every robot loop (if your robot loops are faster than the default then you should change the DifferentialDrivePoseEstimator_DifferentialDrivePoseEstimator(Rotation2d, Pose2d, Matrix, Matrix,

Matrix, double) nominal delta time.) DifferentialDrivePoseEstimator_addVisionMeasurement can be called as infrequently as you want; if you never call it then this class will behave exactly like regular encoder odometry.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

Our state-space system is:

$x = [[x, y, \theta, dist_l, dist_r]]?$ in the field coordinate system (dist_* are wheel distances.)

$u = [[vx, vy, \omega]]?$ (robot-relative velocities) -- NB: using velocities make things considerably easier, because it means that teams don't have to worry about getting an accurate model. Basically, we suspect that it's easier for teams to get good encoder data than it is for them to perform system identification well enough to get a good model.

$y = [[x, y, \theta]]?$ from vision, or $y = [[dist_l, dist_r, \theta]]$ from encoders and gyro.

Constructs a DifferentialDrivePoseEstimator.

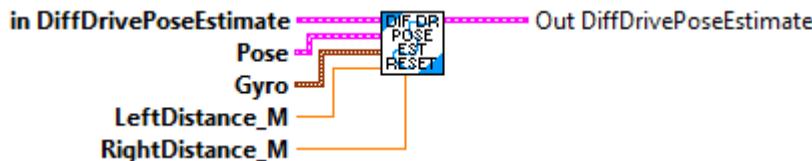
Inputs

- **gyroAngle** -- The current gyro angle. (radians)
- **initialPoseMeters** -- The starting pose estimate.
- **stateStdDevs** -- Standard deviations of model states. Increase these numbers to trust your model's state estimates less. This matrix is in the form [x, y, theta, dist_l, dist_r]?, with units in meters and radians.
- **localMeasurementStdDevs** -- Standard deviations of the encoder and gyro measurements. Increase these numbers to trust sensor readings from encoders and gyros less. This matrix is in the form [dist_l, dist_r, theta]?, with units in meters and radians.
- **visionMeasurementStdDevs** -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form [x, y, theta]?, with units in meters and radians.
- **nominalDtSeconds** -- The time in seconds between each robot loop. (seconds)
- **leftDistance_M** -- The initial left wheel distance measurement (meters)
- **rightDistance_M** -- The initial right wheel distance measurement (meters)

Outputs:

- **DiffDrivePoseEst** -- Data cluster for this instance of this system.
- **Error** -- A value of TRUE indicates an error occurred creating the system.

DiffDrivePoseEst_ResetPosition



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Resets the robot's position on the field.

The gyroscope angle does not need to be reset here on the user's robot code. The library automatically takes care of offsetting the gyro angle.

Inputs:

- **inDiffDrivePoseEst** -- system data cluster.
- **poseMeters** -- The position on the field that your robot is at.
- **gyroAngle** -- The angle reported by the gyroscope.
- **LeftDistance** -- The current left distance measurement (meters)
- **RightDistance** -- The current right distance measurement (meters)

Outputs:

- **outDiffDrivePoseEst** -- Updated system data cluster.

DiffDrivePoseEst_SetVisionMeasurementStdDevs



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Sets the pose estimator's trust of global measurements. This might be used to change trust in vision measurements after the autonomous period, or to change trust as distance to a vision target increases.

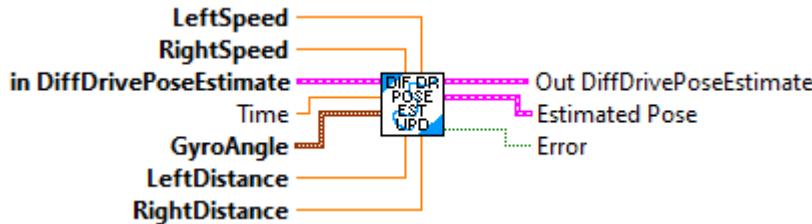
Inputs:

- **inDiffDrivePoseEst** -- System data cluster
- **visionMeasurementStdDevs** -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form [x, y, theta]?, with units in meters and radians.

Outputs:

- **outDiffDrivePoseEst** -- System data cluster
 - **error** -- A value of TRUE indicates an unexpected error occurred.
-

DiffDrivePoseEst_Update



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Updates the the Unscented Kalman Filter using only wheel encoder information. Note that this should be called every loop.

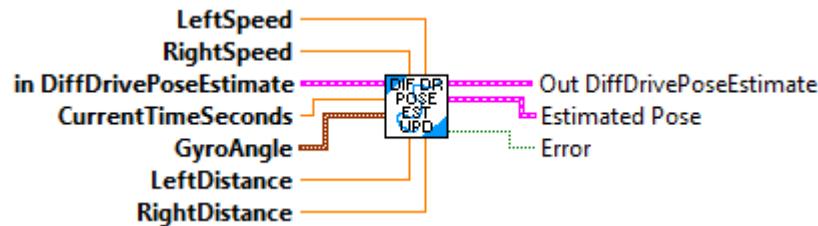
Inputs:

- **LeftSpeed** -- Left wheel speed (meters/sec)
- **RightSpeed** -- Right wheel speed (meters/sec)
- **inDiffDrivePoseEst** -- system data cluster
- **gyroAngle** -- The current gyro angle. (radians)
- **distanceLeftMeters** -- The total distance travelled by the left wheel in meters. This can be the encoder reading.
- **distanceRightMeters** -- The total distance travelled by the right wheel in meters. This can be the encoder reading.

Outputs:

- **outDiffDrivePoseEst** -- system data cluster
 - **EstimatedPose** -- The estimated pose of the robot in meters.
-

DiffDrivePoseEst_UpdateWithTime



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Updates the the Unscented Kalman Filter using only wheel encoder information. Note that this should be called every loop.

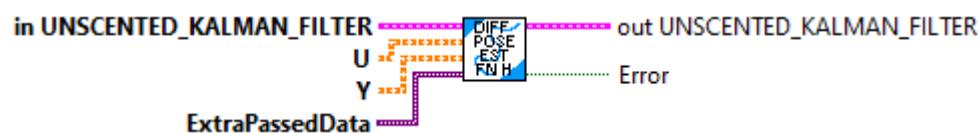
Inputs:

- **LeftSpeed** -- Left wheel speed (meters/sec)
- **RightSpeed** -- Right wheel speed (meters/sec)
- **inDiffDrivePoseEst** -- system data cluster
- **gyroAngle** -- The current gyro angle. (radians)
- **currentTime** -- Time at which this method was called, in seconds.
- **distanceLeftMeters** -- The total distance travelled by the left wheel in meters. This can be the encoder reading.
- **distanceRightMeters** -- The total distance travelled by the right wheel in meters. This can be the encoder reading.

Outputs:

- **outDiffDrivePoseEst** -- system data cluster
- **EstimatedPose** -- The estimated pose of the robot in meters.

DiffDrivePoseEst_VisionCorrect_Callback

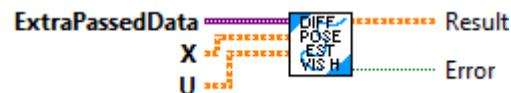


This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Function used by LatencyCompensator_ApplyPastGlobalMeasurement

This is a work in progress!!

DiffDrivePoseEst_VisionCorrect_Kalman_H_Callback



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

Internal function used by DiffDrivePoseEst_Update routine used by the VisionCorrect routine.

Inputs:

- ExtraPassedData -- Variant of extra data used by this function. (This value is not used.)
- X -- X matrix
- U -- U matrix

Outputs:

- Result -- Resulting matrix
- Error -- If TRUE, an error occurred.

DiffDrivePoseEst2

DiffDrivePoseEst2_AddVisionMeasurement



Add a vision measurement to the Unscented Kalman Filter. This will correct the odometry pose estimate while still accounting for measurement noise.

This method can be called as infrequently as you want, as long as you are calling `DifferentialDrivePoseEstimator_update` every loop.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

Inputs:

- `InDIffDrivePoseEstimate` -- Data cluster for this system
- `visionRobotPoseMeters` -- The pose of the robot as measured by the vision camera.
- `timestampSeconds` -- The timestamp of the vision measurement in seconds. Note that if you don't use your own time source by calling `DifferentialDrivePoseEstimator_updateWithTime` then you must use a timestamp with an epoch since FPGA startup (i.e. the epoch of this timestamp is the same epoch as `Timer.getFPGATimestamp`.) This means that you should use `Timer.getFPGATimestamp` as your time source in this case.

Outputs:

- `OutDIffDrivePoseEstimate` -- Data cluster for this system
- `Error` -- Returns TRUE if an error occurred.

DiffDrivePoseEst2_BufferDuration



Maximum buffer duration of diff pose est 2 buffer.

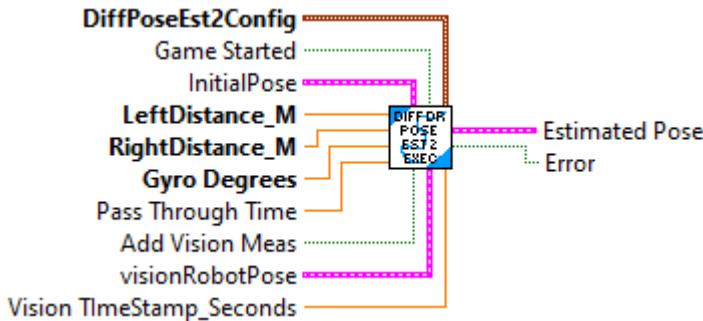
Inputs:

- none

Outputs:

- Buffer Duration -- double -- buffer duration (seconds)

DiffDrivePoseEst2_Execute



This single call LabVIEW function wraps Differential Drive Odometry to fuse latency-compensated vision measurements with differential drive encoder measurements. It is intended to be a drop-in replacement for DiffDrvOdom2; in fact, if you never call DiffDrvPoseEst2_AddVisionMeasurement and only call DiffDrvPoseEst2_Update then this will behave exactly the same as DiffDrvOdom2.

This function should be called every robot loop, or perhaps put into a separate loop.

When the "Game Started" input is false, the position is reset using the initial pose, wheel distances, and gyro position. Use this to wait processing changes in measurements until the game starts.

When a new vision measurement is available it can be passed into this function to merge it with the other odometry data. This is done by setting the "Add Vision Meas" input to TRUE. To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

The default standard deviations of the model states are 0.02 meters for x, 0.02 meters for y, and 0.01 radians for heading. The default standard deviations of the vision measurements are 0.1 meters for x, 0.1 meters for y, and 0.1 radians for heading.

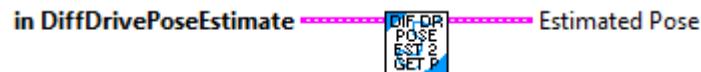
Inputs:

- DiffPoseEst2Config -- cluster -- Packed configuration data used to define how this function operates. It includes information used to define kinematics and position standard deviations.
- Game Started -- boolean -- When FALSE, resets the initial position. (Default: TRUE)
- InitialPose -- Pose2d -- Initial position. (meters, meters, rotation2d)
- leftDistanceMeters -- double -- The distance traveled by the left encoder. (Meters)
- rightDistanceMeters -- double -- The distance traveled by the right encoder. (Meters)
- gyroAngle -- Rotation2d -- The current gyro angle. (Degrees)
- Pass Through Time -- double -- Current FPGA time. If input is unwired, FPGA time will be read internally.
- AddVisionMeas -- boolean -- When TRUE the provided vision measurement is merged with the odometry data.
- VisionRobotPose -- Pose2d -- The pose from vision measurements.
- Vision Time Stamp -- double -- The FPGA time stamp of the vision measurement. This helps to account for latency of the vision measurement.

Outputs:

- EstimatedPose -- Pose2d -- Current estimated pose (meters, rotation2d)
- Error -- boolean -- If TRUE, an error occurred.

DiffDrivePoseEst2_GetEstimatedPosition



Gets the pose of the robot at the current time as estimated by the Unscented Kalman Filter.

Inputs:

- DiffDrivePoseEst - System data cluster

Outputs:

- EstimatedPose - The estimated robot pose in meters.

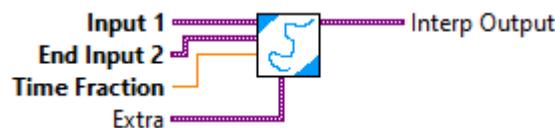
DiffDrivePoseEst2_InterpRecord_ExtractFromVar



This extracts the specific data cluster of Diff Drive Pose Estimator2 Interpolation Record from a Variant.

This is an internal routine. It should NOT be called by the end user.

DiffDrivePoseEst2_InterpRecord_Interp



Return the interpolated record. This object is assumed to be the starting position, or lower bound.

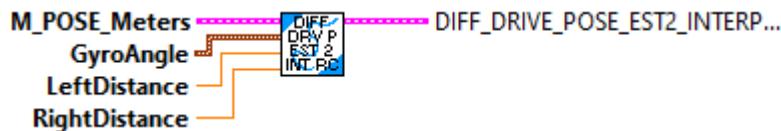
Inputs:

- Input 1 -- The lower bound data cluster
- end Input 2 -- The upper bound, or end, data cluster
- time frac -- How far between the lower and upper bound we are. This should be bounded in [0, 1].
- extra -- variant -- extra data needed for the calculation.

Returns:

- The interpolated cluster value.

DiffDrivePoseEst2_InterpRecord_New



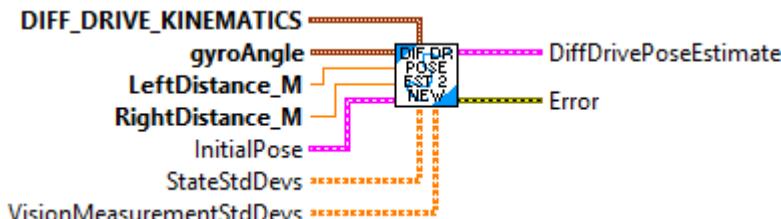
Diff Drive Pose Estimate 2 -- Interporatable Record

Represents an odometry record. The record contains the inputs provided as well as the pose that was observed based on these inputs, as well as the previous record and its inputs.

Constructs an Interpolation Record with the specified parameters.

This is an internal routine. It should NOT be used by the end user.

DiffDrivePoseEst2_New



This class wraps Differential Drive Odometry to fuse latency-compensated vision measurements with differential drive encoder measurements. It is intended to be a drop-in replacement for DiffDrvOdom2; in fact, if you never call DiffDrvPoseEst2_AddVisionMeasurement and only call DiffDrvPoseEst2_Update then this will behave exactly the same as DiffDrvOdom2.

DiffDrvPoseEst2_Update should be called every robot loop.

DiffDrvPoseEst2_AddVisionMeasurement can be called as infrequently as you want. If you never call it then this set of VI will behave exactly like regular encoder odometry.

Constructs a DifferentialDrivePoseEstimator.

The default standard deviations of the model states are 0.02 meters for x, 0.02 meters for y, and 0.01 radians for heading. The default standard deviations of the vision measurements are 0.1 meters for x, 0.1 meters for y, and 0.1 radians for heading.

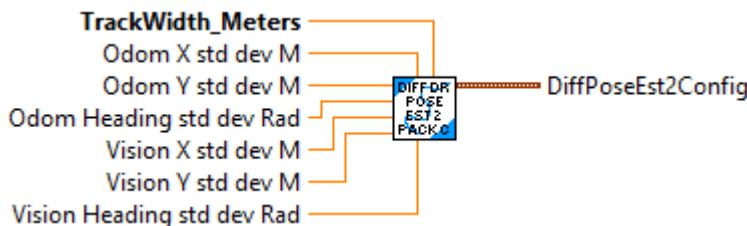
Inputs:

- kinematics -- DiffDriveKinematics -- A correctly-configured kinematics data cluster for your drivetrain.
- gyroAngle -- Rotation2d -- The current gyro angle.
- leftDistanceMeters -- double -- The distance traveled by the left encoder.
- rightDistanceMeters -- double -- The distance traveled by the right encoder.
- initialPoseMeters -- Pose2d -- The starting pose estimate.
- stateStdDevs -- <3,1> matrix -- Standard deviations of the pose estimate (x position in meters, y position in meters, and heading in radians). Increase these numbers to trust your state estimate less.
- visionMeasurementStdDevs -- <3,1> matrix -- Standard deviations of the vision pose measurement (x position in meters, y position in meters, and heading in radians). Increase these numbers to trust the vision pose measurement less.

Outputs:

- DiffDrivePoseEst2 -- DiffDrivePoseEst2 -- Created data cluster.

DiffDrivePoseEst2_Pack_Config



Pack individual values into the cluster required by the DiffDrivePoseEst2_Execute function.

Inputs

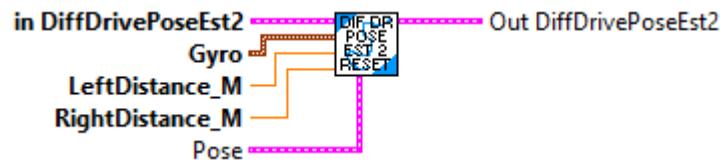
- TrackWidth_Meters -- double -- Effective tract width (meters)
- Odom X Std Dev M -- double -- Odometry X position standard deviation (Default: 0.02) (meters)

- Odom Y Std Dev M -- double -- Odometry Y position standard deviation (Default: 0.02) (meters)
- Odom Heading Std Dev M -- double -- Odometry Heading (gyro) position standard deviation (Default: 0.01) (radians)
- Vision X Std Dev M -- double -- Vision X position standard deviation (Default: 0.1) (meters)
- Vision Y Std Dev M -- double -- Vision Y position standard deviation (Default: 0.1) (meters)
- Vision Heading Std Dev M -- double -- Vision Heading (gyro) position standard deviation (Default: 0.05) (radians)

Outputs:

- DiffPoseEst2Config -- cluster -- Packed data values.

DiffDrivePoseEst2_ResetPosition



Resets the robot's position on the field.

The gyroscope angle does not need to be reset here on the user's robot code. The library automatically takes care of offsetting the gyro angle.

Inputs:

- inDiffDrvPoseEst2 -- DifDrvPoseEst2 -- Data cluster
- gyroAngle -- Rotation2d -- The angle reported by the gyroscope.
- leftPositionMeters -- double -- The distance traveled by the left encoder.
- rightPositionMeters -- double -- The distance traveled by the right encoder.
- poseMeters -- Pose2d -- The position on the field that your robot is at.

Outputs:

- outDiffDrvPoseEst2 -- DifDrvPoseEst2 -- Updated data cluster

DiffDrivePoseEst2_SetVisionMeasurementStdDevs



Sets the pose estimator's trust of global measurements. This might be used to change trust in vision measurements after the autonomous period, or to change trust as distance to a vision target increases.

Inputs:

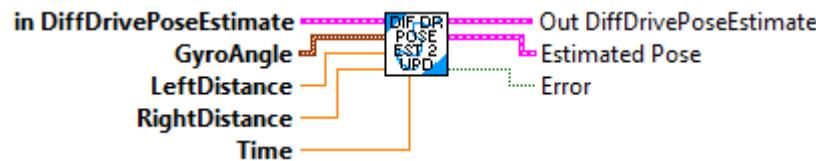
- inDiffDrvPoseEst2 -- DifDrvPoseEst2 -- Data cluster

- VisionMeasurementStdDevs -- <3,1> Matrix -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form [x, y, theta]Time, with units in meters and radians.

Outputs:

- outDiffDrvPoseEst2 -- DifDrvPoseEst2 -- Updated data cluster
- sizeCooerced -- boolean -- If TRUE, then the size of the vision measurement standard deviations was not 3,1. The size was modified to allow this routine to complete.

DiffDrivePoseEst2_Update



Updates the the Unscented Kalman Filter using only wheel encoder information. Note that this should be called every loop.

Inputs:

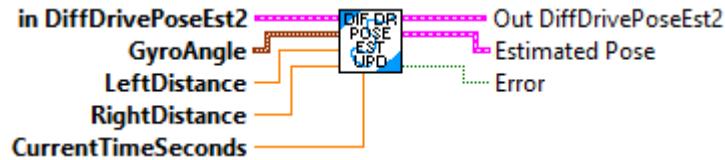
- LeftSpeed -- Left wheel speed (meters/sec)

- RightSpeed -- Right wheel speed (meters/sec)
- inDiffDrivePoseEst -- system data cluster
- gyroAngle -- The current gyro angle. (radians)
- distanceLeftMeters -- The total distance travelled by the left wheel in meters. This can be the encoder reading.
- distanceRightMeters -- The total distance travelled by the right wheel in meters. This can be the encoder reading.

Outputs:

- outDiffDrivePoseEst -- system data cluster
- EstimatedPose -- The estimated pose of the robot in meters.

DiffDrivePoseEst2_UpdateWithTime



Updates the the Unscented Kalman Filter using only wheel encoder information. Note that this should be called every loop.

Inputs:

- LeftSpeed -- Left wheel speed (meters/sec)
- RightSpeed -- Right wheel speed (meters/sec)
- inDiffDrivePoseEst -- system data cluster
- gyroAngle -- The current gyro angle. (radians)
- currentTime -- Time at which this method was called, in seconds.
- distanceLeftMeters -- The total distance travelled by the left wheel in meters. This can be the encoder reading.

- `distanceRightMeters` -- The total distance travelled by the right wheel in meters. This can be the encoder reading.

Outputs:

- `outDiffDrivePoseEst` -- system data cluster
- `EstimatedPose` -- The estimated pose of the robot in meters.

DiffDriveTrainSim

DiffDriveTrainSim_ClampInput



Clamp the input vector such that no element exceeds the battery voltage. If any does, the relative magnitudes of the input will be maintained.

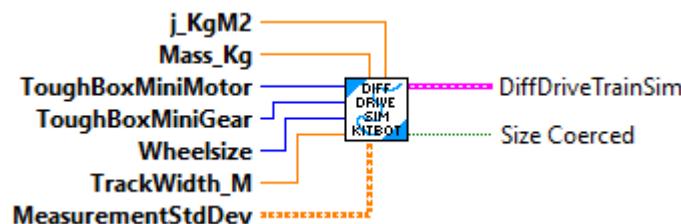
Inputs:

- u -- The input vector.
- BatteryVoltage -- The battery voltage (volts)

Outputs:

- NormalizedU -- The normalized input.
- SizeCoerced -- If TRUE, an error occurred.

DiffDriveTrainSim_CreateKitbotSim



Create a sim for the standard FRC kitbot.

Inputs:

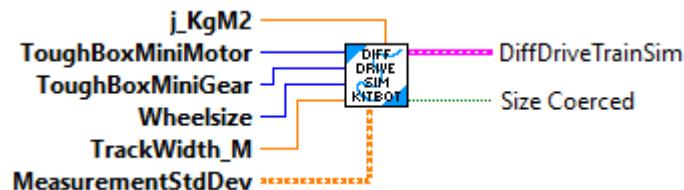
- jKgMetersSquared -- The moment of inertia of the drivebase. This can be calculated using SysId.
- mass -- mass of robot - kg
- motor -- The motors installed in the bot.

- gearing -- The gearing reduction used.
- wheelSize -- The wheel size.
- trackWidth -- Width between left and right wheels - Meters
- measurementStdDevs -- Standard deviations for measurements, in the form [x, y, heading, left velocity, right velocity, left distance, right distance]?. Can be null if no noise is desired. Gyro standard deviations of 0.0001 radians, velocity standard deviations of 0.05 m/s, and position measurement standard deviations of 0.005 meters are a reasonable starting point.

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.
- SizeCoerced -- If TRUE, an error occurred. Execution may continue

DiffDriveTrainSim_CreateKitbotSim_EstMass



Create a sim for the standard FRC kitbot.

The mass of the robot is estimated at 25 pounds, converted to kg before use.

Inputs:

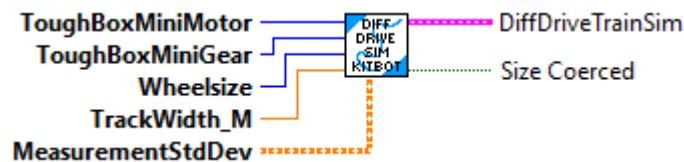
- jKgMetersSquared -- The moment of inertia of the drivebase. This can be calculated using SysId.
- motor -- The motors installed in the bot.
- gearing -- The gearing reduction used.
- wheelSize -- The wheel size.
- trackWidth -- Width between left and right wheels - Meters

- measurementStdDevs -- Standard deviations for measurements, in the form [x, y, heading, left velocity, right velocity, left distance, right distance]?. Can be null if no noise is desired. Gyro standard deviations of 0.0001 radians, velocity standard deviations of 0.05 m/s, and position measurement standard deviations of 0.005 meters are a reasonable starting point.

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.
- SizeCoerced -- If TRUE, an error occurred. Execution may continue

DiffDriveTrainSim_CreateKitbotSim_EstMassMOI



Create a sim for the standard FRC kitbot.

The robot's mass of the robot is estimated at 25 pounds, converted to kg before use.

The robot;s moment of inertia (MOI) is estimated.

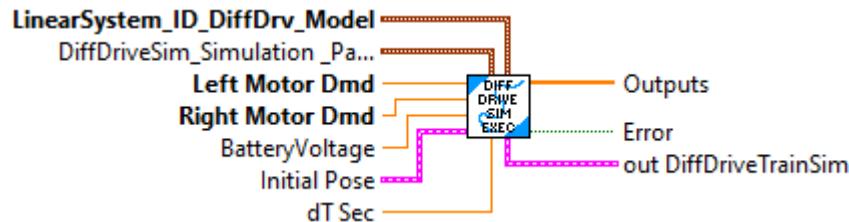
Inputs:

- motor -- The motors installed in the bot.
- gearing -- The gearing reduction used.
- wheelSize -- The wheel size.
- trackWidth -- Width between left and right wheels - Meters
- measurementStdDevs -- Standard deviations for measurements, in the form [x, y, heading, left velocity, right velocity, left distance, right distance]?. Can be null if no noise is desired. Gyro standard deviations of 0.0001 radians, velocity standard deviations of 0.05 m/s, and position measurement standard deviations of 0.005 meters are a reasonable starting point.

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.
- SizeCoerced -- If TRUE, an error occurred. Execution may continue

DiffDriveTrainSim_Execute



Single call LabVIEW function to simulate a Differential Drive Train system.

The diff drive simulation linear system contains:

- states
 - X position meters
 - Y position meters -
 - heading radians
 - left velocity meter/sec-
 - right velocity meters/sec
 - left distance meters
 - right distance meters
- inputs
 - left motor demand volts
 - right motor demand volts.
- outputs
 - X position meters
 - Y position meters -
 - heading radians

- left velocity meter/sec-
- right velocity meters/sec
- left distance meters
- right distance meters

Inputs:

- Linear System DiffDrv Model -- cluster -- Contains physical configuration for Diff Drive Train system.
- Diff Drv Sim Simulation Params -- cluster -- Contains siulation configuration.
- Initial Pose -- Pose2d -- Initial location of the drive train.
- Left Motor Dmd (CO) -- double -- left motor demand value (+/- 1.0)
- Right Motor Dmd (CO) -- double -- right motor demand value (+/- 1.0)
- Battery Voltage -- double -- Current simulated battery voltage (Volts)
- dT Sec -- double -- Update period (Seconds, Default: 0.02)

Outputs:

- Outputs -- double array -- Outputs. The index values are:
 - 0 - X position meters
 - 1 - Y position meters -
 - 2 - heading radians
 - 3 - left velocity meter/sec-
 - 4 - right velocity meters/sec
 - 5 - left distance meters
 - 6 - right distance meters
- Error -- boolean -- If TRUE an error occured.
- outDiffDriveTrainSim -- cluster -- Current internal data cluster.

DiffDriveTrainSim_GetCurrentDrawAmps



Get the current draw of the drivetrain.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- TotalCurrent_Amps -- the current draw, in amps

DiffDriveTrainSim_GetCurrentGearing



Get the drivetrain gearing.

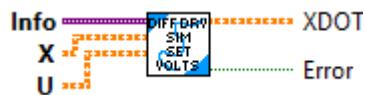
Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- CurrentGearing -- the gearing ration

DiffDriveTrainSim_GetDynamics



Diff Drive Train Sim internal call back function to get dynamics. This is used by the Update function.

Inputs:

- Info -- Extra data variant value containing:

Matrix A

Matrix B

Double CurrentGear

Double OriginalGear

Double m_rb

- X -- X matrix
- U -- U matrix

Outputs:

- xDot
- Error -- If TRUE, an error occurred.

DiffDriveTrainSim_GetHeading



Returns the direction the robot is pointing.

Note that this angle is counterclockwise-positive, while most gyros are clockwise positive.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- Heading -- The direction the robot is pointing.
-
-

DiffDriveTrainSim_GetLeftCurrentDrawAmps



Get the current draw of the left side of the drivetrain.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- LeftCurrentDraw_Amp -- the drivetrain's left side current draw, in amps
-
-

DiffDriveTrainSim_GetLeftPositionMeters



Get the left encoder position in meters.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- LeftPosition_Meters -- The encoder position.

DiffDriveTrainSim_GetLeftVelocityMetersPerSecond



Get the left encoder velocity in meters per second.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- LeftVelocity_M_Per_Sec -- The encoder velocity.

DiffDriveTrainSim_GetOutput_Single



Get the output

Inputs:

- DiffDriveTrainSim -- Data cluster
- Row -- The desired row.

values correspond to:

$$0 = X$$

$$1 = Y$$

$$2 = \text{Heading}$$

$$3 = \text{LeftVelocity}$$

$$4 = \text{RightVelocity}$$

5 = LeftPosition

6 = RightPosition

Outputs:

- Output -- The output value
-
-

DiffDriveTrainSim_GetPose



Returns the current pose.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- Pose -- The current pose.
-
-

DiffDriveTrainSim_GetRightCurrentDrawAmps



Get the current draw of the right side of the drivetrain.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- RightCurrentDraw_Amps -- the drivetrain's right side current draw, in amps
-

DiffDriveTrainSim_GetRightPositionMeters



Get the right encoder position in meters.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- RightPosition_Meters -- The encoder position.
-

DiffDriveTrainSim_GetRightVelocityMetersPerSecond



Get the right encoder velocity in meters per second.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- RightVel_M_Per_Sec -- The encoder velocity.
-

DiffDriveTrainSim_GetState



Returns the full simulated state of the drivetrain.

Inputs:

- DiffDriveTrainSim -- Data cluster

Outputs:

- X -- The state matrix

The states are: $[[x, y, \theta, v_l, v_r, d_l, d_r]]$

DiffDriveTrainSim_GetState_Single



Get one of the drivetrain states.

Inputs:

- DiffDriveTrainSim -- Data cluster
- row -- the state to get

The states are: $[[x, y, \theta, v_l, v_r, d_l, d_r]]$

Outputs:

- State -- the state value

DiffDriveTrainSim_KitBotWheelSize



Convenience function to allow user to select kit bot wheels size from an enum selector and return the diameter in meters.

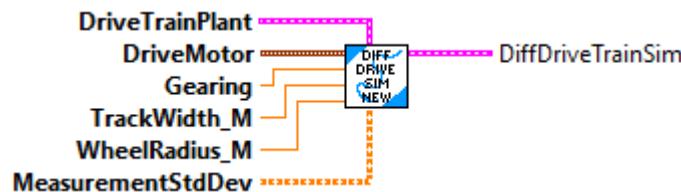
Input:

- Wheel Size -- Enumerated heel size selector

Output:

- WheelDiameter_M -- Wheel diameter in meters.
- WheelRadius_M -- Wheel radius in meters.

DiffDriveTrainSim_New



Create a SimDrivetrain .

This class simulates the state of the drivetrain. In simulationPeriodic, users should first set inputs from motors with setInputs(double, double)}, call update(double)} to update the simulation, and set estimated encoder and gyro positions, as well as estimated odometry pose.

The input linear state space system is:

The linear system is define as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs

- left velocity (m/s)
- right velocity (m/s)

Our state-space system is:

$$x = [[x, y, \theta, vel_l, vel_r, dist_l, dist_r]]$$

in the field coordinate system (dist_* are wheel distances.)

$$u = [[voltage_l, voltage_r]]?$$

This is typically the control input of the last timestep from a LTVDiffDriveController.

$$y = x$$

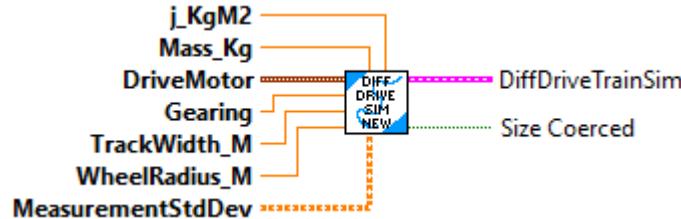
Inputs:

- drivetrainPlant -- The LinearSystem representing the robot's drivetrain. This system can be created with `LinearSystemId_createDrivetrainVelocitySystem(DCMotor, double, double, double, double, double)` or `LinearSystemId_identifyDrivetrainSystem(double, double, double, double)`
- driveMotor -- A DCMotor representing the drivetrain.
- gearing -- The gearingRatio ratio of the robot, as output over input. This must be the same ratio as the ratio used to identify or create the drivetrainPlant.
- trackWidthMeters -- The distance between the two sides of the drivetrain. Can be found with SysId.
- wheelRadiusMeters -- The radius of the wheels on the drivetrain, in meters.
- measurementStdDevs -- Standard deviations for measurements, in the form [x, y, heading, left velocity, right velocity, left distance, right distance]?. Can be null if no noise is desired. Gyro standard deviations of 0.0001 radians, velocity standard deviations of 0.05 m/s, and position measurement standard deviations of 0.005 meters are a reasonable starting point.

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.

DiffDriveTrainSim_New_Mass_MOI



Create a SimDrivetrain.

This class simulates the state of the drivetrain. In simulationPeriodic, users should first set inputs from motors with setInputs(double, double)}, call update(double)} to update the simulation, and set estimated encoder and gyro positions, as well as estimated odometry pose.

Our state-space system is:

$$x = [[x, y, \theta, vel_l, vel_r, dist_l, dist_r]]?$$

in the field coordinate system (dist_* are wheel distances.)

$$u = [[voltage_l, voltage_r]]?$$

This is typically the control input of the last timestep
from a LTVDiffDriveController.

$$y = x$$

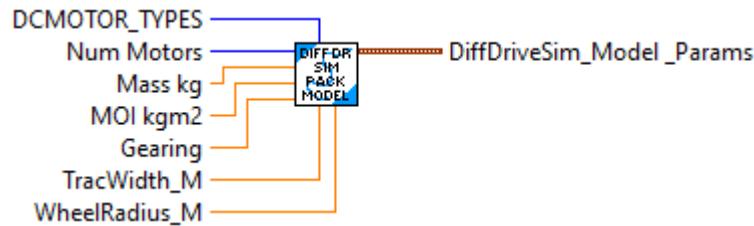
Inputs:

- DiffDriveTrainSim -- Data cluster
- driveMotor -- A DCMotor representing the left side of the drivetrain.
- gearing -- The gearing ratio between motor and wheel, as output over input. This must be the same ratio as the ratio used to identify or create the drivetrainPlant.
- jKgMetersSquared -- The moment of inertia of the drivetrain about its center.
- massKg -- The mass of the drivebase.
- wheelRadiusMeters -- The radius of the wheels on the drivetrain.
- trackWidthMeters -- The robot's track width, or distance between left and right wheels.
- measurementStdDevs -- Standard deviations for measurements, in the form [x, y, heading, left velocity, right velocity, left distance, right distance]?. Can be null if no noise is desired. Gyro standard deviations of 0.0001 radians, velocity standard deviations of 0.05 m/s, and position measurement standard deviations of 0.005 meters are a reasonable starting point.

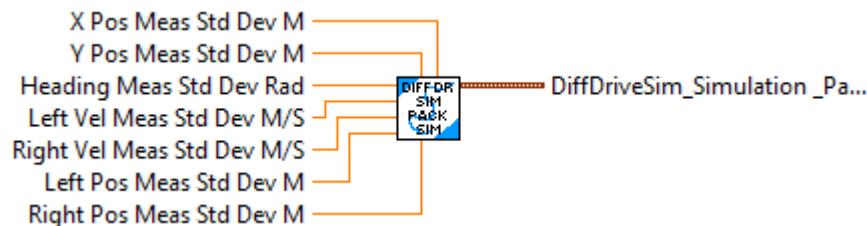
Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.
- Size Coerced -- If TRUE, an error occurred. Execution may continue.

DiffDriveTrainSim_Pack_Model_Params



DiffDriveTrainSim_Pack_Simulation_Params



DiffDriveTrainSim_SetCurrentGearing



Sets the gearing reduction on the drivetrain. This is commonly used for shifting drivetrains.

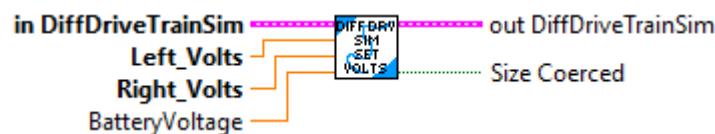
Inputs:

- DiffDriveTrainSim -- Data cluster
- newGearRatio -- The new gear ratio, as output over input.

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.

DiffDriveTrainSim_SetInputs



Sets the applied voltage to the drivetrain. Note that positive voltage must make that side of the drivetrain travel forward (+X).

Inputs:

- DiffDriveTrainSim -- Data cluster
- leftVoltageVolts -- The left voltage.

- rightVoltageVolts -- The right voltage.
- BatteryVoltage -- Current battery voltage (volts) (Default = 12 volts)

Outputs:

- OutDiffDriveTraimSim -- Updated data cluster.
 - SizeCoerced -- If TRUE, an error occurred. Execution may continue.
-

DiffDriveTrainSim_SetPose



Sets the system pose.

Inputs:

- DiffDriveTraimSim -- Data cluster
- pose -- The pose.

Outputs:

- OutDiffDriveTraimSim -- Updated data cluster.
-

DiffDriveTrainSim_SetState



Sets the system state.

Inputs:

- DiffDriveTrainSim -- Data cluster
- state -- The state.

The states are: [[x, y, theta, vel_l, vel_r, dist_l, dist_r]]

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.
- SizeCoerced -- If TRUE, an error occurred.

DiffDriveTrainSim_ToughBoxMiniGearRatio



Convenience function to allow user to select kit bot gear ratio from an enum selector and return the ratio value

Input:

- StandardGearRatios -- Enumerated gear ratio selector

Output:

- GearRatio -- Gear ratio value.

DiffDriveTrainSim_ToughBoxMiniMotor



Convenience function to allow user to select standard kit bot motor choices from an enum selector and return the motor data

Input:

- ToughBoxMiniMotor -- Enumerated motor selector

Output:

- DriveMotor -- Selected motor data.
-

DiffDriveTrainSim_Update



Update the drivetrain states with the current time difference.

Inputs:

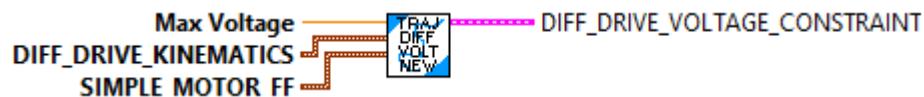
- DiffDriveTrainSim -- Data cluster
- dtSeconds -- the time difference

Outputs:

- OutDiffDriveTrainSim -- Updated data cluster.
- Error -- If TRUE, an error occurred.

DiffDriveVoltageConstraint

DiffDriveVoltageConstraint_New



Creates a new DifferentialDriveVoltageConstraint.

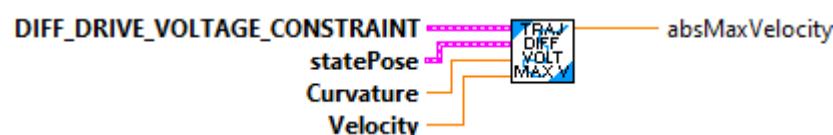
Parameters:

- maxVoltage - The maximum voltage available to the motors while following the path. Should be somewhat less than the nominal battery voltage (12V) to account for "voltage sag" due to current draw.
- DiffDriveKinematics - A kinematics component describing the drive geometry.
- SimpleMotorFeedforward - A feedforward component describing the behavior of the drive.

Returns

- DiffDriveVoltageConstraint - Constraint data structure

DiffDriveVoltageConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

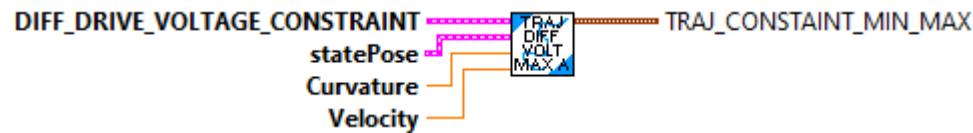
Parameters:

- DiffDriveVoltageConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.
-

DiffDriveVoltageConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values.

Parameters:

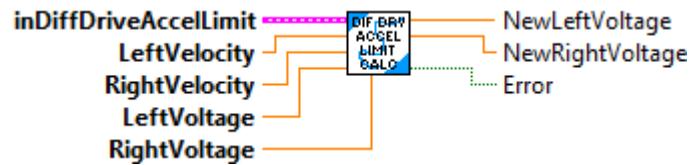
- DiffDriveVoltageConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

DiffDrvAccelLimit

DiffDrvAccelLimit_Calculate



Returns the next voltage pair subject to acceleration constraints.

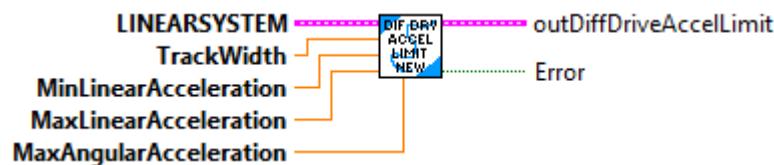
Inputs:

- DiffDriveAccelLimit -- Data cluster
- leftVelocity -- The left wheel velocity in meters per second.
- rightVelocity -- The right wheel velocity in meters per second.
- leftVoltage -- The unconstrained left motor voltage.
- rightVoltage -- The unconstrained right motor voltage.

Outputs:

- NewLeftVoltage -- The constraint drive left voltage output
- NewRightVoltage -- The constraint drive right voltage output
- Error -- If TRUE, an error occurred.

DiffDrvAccelLimit_New



Constructs a DifferentialDriveAccelerationLimiter data cluster.

Inputs:

- system -- The differential drive dynamics.
- trackwidth -- The trackwidth.
- minLinearAccel -- The minimum (most negative) linear acceleration in meters per second squared.
- maxLinearAccel -- The maximum (most positive) linear acceleration in meters per second squared.

Outputs:

- DiffDriveAccelLimit -- Created data cluster
- Error -- True if an error occurred. (Min Accel limit is > than Max Accel Limit)

DiffKinematics

DiffKinematics_New



Constructs a differential drive kinematics data structure.

Helper class that converts a chassis velocity (dx and dtheta components) to left and right wheel velocities for a differential drive.

Inverse kinematics converts a desired chassis speed into left and right velocity components whereas forward kinematics converts left and right component velocities into a linear and angular chassis speed.

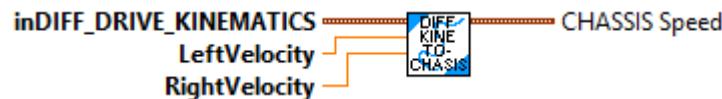
Parameters:

- TrackWidthMeters - The track width of the drivetrain. Theoretically, this is the distance between the left wheels and right wheels. However, the empirical value may be larger than the physical measured value due to scrubbing effects.

Returns:

- Diff Drive Kinematics -- Data structure for Differential Drive Kinematics

DiffKinematics_toChassisSpeed



Returns a chassis speed from left and right component velocities using forward kinematics.

Parameters:

- in Diff Drive Kinematics -- This Differential Drive Kinematics data structure
- LeftVelocity - The left wheel speed (meters/sec).
- RightVelocity - The right wheel speed (meters/sec)

Returns:

- Chassis Speed - The chassis speed data structure

DiffKinematics_toTwist2d



Performs forward kinematics to return the resulting Twist2d from the given left and right side distance deltas. This method is often used for odometry -- determining the robot's position on the field using changes in the distance driven by each wheel on the robot.

Parameters:

- DiffDriveKinematics -- Diff_Drive_Kinematics -- This differential drive kinematics data cluster
- LeftDistance -- double -- Left distance (meters)
- RightDistance -- double -- Right distance (meters)

Returns:

- Twist2d -- Twist2d -- The resulting Twist2d

DiffKinematics_toWheelSpeed



Returns left and right component velocities from a chassis speed using inverse kinematics.

Parameters:

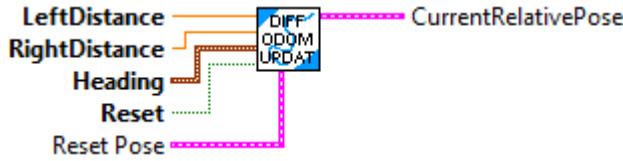
- DiffDriveKinematics - This differential drive kinematics data structure
- chassisSpeeds - The linear and angular (dx and dtheta) components that represent the chassis' speed.

Returns:

- LeftVelocity - Left wheel velocity (meters/sec)
- RightVelocity - Right wheel velocity (meters/sec)

DiffOdometry

DiffOdometry_Update



SubVI for differential drive odometry. Odometry allows you to track the robot's position on the field over the course of a match using readings from 2 encoders and a gyroscope.

Teams can use odometry during the autonomous period for complex tasks like path following. Furthermore, odometry can be used for latency compensation when using computer-vision systems.

Updates the robot position on the field using distance measurements from encoders. This method is more numerically accurate than using velocities to integrate the pose and is also advantageous for teams that are using lower CPR encoders.

This implementation slightly differs from the WPILIB C++/Java. The constructor, Update and Reset functions are incorporated into this single routine. There is no need to reset encoders or gyros. This routine compensates by remembering the values at reset. An optional "Initial POSE" can be specified to allow this routine to calculate absolute position. If not specified the position is relative to the position at reset. More than one instance of this can be used if needed. One instance could track absolute field position while another instance can track relative position for executing a trajectory.

Parameters:

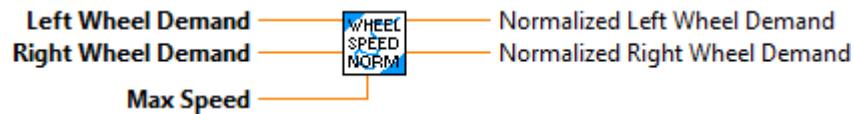
- Left distance -- Left wheel encoder distance (meters)
- Right distance -- Right wheel encoder distance (meters)
- Heading - Robot gyroscope reading
- Reset - When set, this resets the remembered left, right distance and gyroscope values, and sets the current pose to the initial pose.
- Reset Pose - (Optional) When reset is TRUE this pose is used to set the current robot pose. (If the initial pose is zero or not present, the result will be a relative pose. If the reset pose is the absolute field pose, then the result current pose will be absolute.)

Results:

- currentRelativePose - The new pose of the robot. (If the initial pose is absolute, this will be an absolute pose.)

DiffWheel

DiffWheel_Normalize



Normalizes the wheel speeds using some max attainable speed. Sometimes, after inverse kinematics, the requested speed from a/several modules may be above the max attainable speed for the driving motor on that module. To fix this issue, one can "normalize" all the wheel speeds to make sure that all requested module speeds are below the absolute threshold, while maintaining the ratio of speeds between modules.

Parameters:

- Left Wheel Demand - Desired left wheel speed demand.
- Right Wheel Demand - Desired right wheel speed demand.
- Max Speed - The absolute max speed that a wheel can reach.

Returns:

- Normalized Left Wheel Demand
- Normalized Right Wheel Demand

DigSeqLogic

DigSeqLogic_Delay



This VI implements a digital "time delay". The output is the input delayed by "Delay" seconds. If the "Delay" time does not exactly match the sample period, the boolean value prior to the delay time is returned.

Inputs:

- Input -- Boolean input value
- Delay Time -- Time (seconds) to delay the output value. If time delay is increased, then the buffer is re-created. FALSE will be returned until the buffer has sufficient data to return a delayed value.
- Time -- Continuously counting system time, Seconds. If not wired the FPGA time will be used.

Outputs:

- Output -- Boolean output value with delay applied
- IsPresent -- Set to TRUE if the buffer has enough data to return a delayed value.

DigSeqLogic_Edge_Change



DigSeqLogic_Edge_Off



DigSeqLogic_Edge_On



DigSeqLogic_Off_Delay



This VI implements a digital "off delay". The output becomes TRUE when the input goes TRUE. The output remains TRUE for a specified time after the input goes FALSE.

Inputs:

- Input -- Boolean input value
- Delay -- Off delay time in seconds.
- Time -- Continuously counting system time, Seconds. If not wired the FPGA time will be used.

Outputs:

- Output -- Boolean output value with the off delay applied
- Remain -- Time remaining for this value to be on, seconds. Value will be 0 to Delay input value.

DigSeqLogic_On_Delay



This VI implements a digital "on delay". When input is zero, output is zero. When output is true, input goes true after the designated time delay. Once the delay counter has started, the time delay value cannot be changed.

Inputs:

- Input -- Boolean input value
- Delay -- On delay time in seconds.

- Time -- Continuously counting system time, Seconds. If not wired the FPGA time will be used.

Outputs:

- Output -- Boolean output value with the on delay applied
 - Remain -- Time remaining for this value to be off in, seconds. Value will be 0 to Delay input value.
-

DigSeqLogic_One_Shot



This VI implements a digital "one shot". The output is true for a specific period of time after the input transitions from FALSE to TRUE.

Inputs:

- Input -- Boolean input to use for one shot output
- OneShot -- Length of one shot output in seconds.
- Time -- Current system time. If not wired, FPGA time is used.

Output:

- Output -- Boolean output one-shot
 - Remain -- Number of seconds remaining in one-shot.
-

DigSeqLogic_SR_FlipFlop



This VI implements a standard Set-Reset (SR) Flip Flop. A flip flop is sometimes called "digital memory" because when both inputs are off, it remembers the state of the last input.

Reset takes precedence over Set. The initial state is Reset.

Inputs:

- Set -- Boolean, when TRUE indicates the flip-flop should be SET.
- Reset -- Boolean, when TRUE indicates the flip-flop should be RESET. (Reset overrides set.)
- InitialValue -- Value of the flip-flop during the first execution. TRUE = SET, FALSE = RESET. (Default: False)

Output:

- Output -- Output state of the flip-flop
- Inverse Output -- Inverse (NOT) output of the flip-flop.

Discretization

Discretization_DiscretizeA



Discretizes the given continuous A matrix.

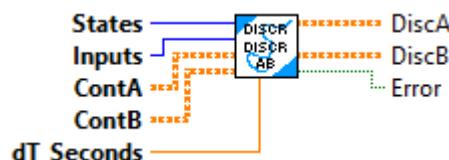
Inputs:

- States -- Num representing the number of states.
- contA -- Continuous system matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- discA -- the discrete matrix system.
- error -- If TRUE, an error occurred.

Discretization_DiscretizeAB



Discretizes the given continuous A and B matrices.

Inputs:

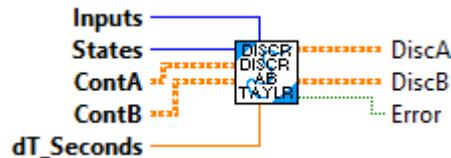
- States -- Nat representing the states of the system.
- Inputs -- Nat representing the inputs to the system.
- contA -- Continuous system matrix.

- contB -- Continuous input matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- discA -- Discretized A matrix
- discB -- Discretized B matrix
- error -- If TRUE, an error occurred.

Discretization_DiscretizeABTaylor



Discretizes the given continuous A and B matrices.

Rather than solving a $(\text{States} + \text{Inputs}) \times (\text{States} + \text{Inputs})$ matrix exponential like in `DiscretizeAB()`, we take advantage of the structure of the block matrix of A and B.

- 1) The exponential of A^*t , which is only $N \times N$, is relatively cheap.
- 2) The upper-right quarter of the $(\text{States} + \text{Inputs}) \times (\text{States} + \text{Inputs})$ matrix, which we can approximate using a taylor series to several terms and still be substantially cheaper than taking the big exponential.

Inputs:

- states -- the states of the system.
- contA -- Continuous system matrix.
- contB -- Continuous input matrix.
- dtseconds -- Discretization timestep.

Outputs:

- DiscA -- Discretized A matrix
- DiscB -- Discretized B matrix
- Error -- If TRUE, an error occurred.

Discretization_DiscretizeAQ



Discretizes the given continuous A and Q matrices.

Inputs:

- States -- the number of states.
- contA -- Continuous system matrix.
- contQ -- Continuous process noise covariance matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- DiscA -- the discrete system matrix
- DiscQ -- process noise covariance matrix.
- error -- If TRUE, an error occurred.

Discretization_DiscretizeAQTaylor



Discretizes the given continuous A and Q matrices.

Rather than solving a $2N \times 2N$ matrix exponential like in `DiscretizeQ()` (which is expensive), we take advantage of the structure of the block matrix of A and Q.

The exponential of $A*t$, which is only $N \times N$, is relatively cheap. 2) The upper-right quarter of the $2N \times 2N$ matrix, which we can approximate using a taylor series to several terms and still be substantially cheaper than taking the big exponential.

Inputs:

- States -- the number of states.
- contA -- Continuous system matrix.
- contQ -- Continuous process noise covariance matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- DiscA -- the discrete system matrix
- DiscQ -- process noise covariance matrix.
- error -- If TRUE, an error occurred.

Discretization_DiscretizeR



Returns a discretized version of the provided continuous measurement noise covariance matrix. Note that $dt=0.0$ divides R by zero.

Inputs:

- Outputs -- the number of outputs.

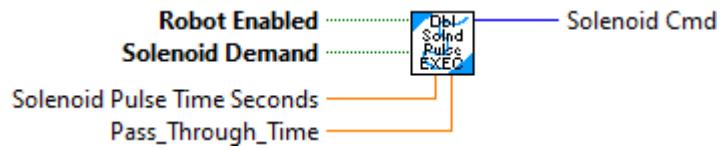
- R -- Continuous measurement noise covariance matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- DiscR -- Discretized version of the provided continuous measurement noise covariance matrix.
- SizeCoerced -- If TRUE, an error occurred.

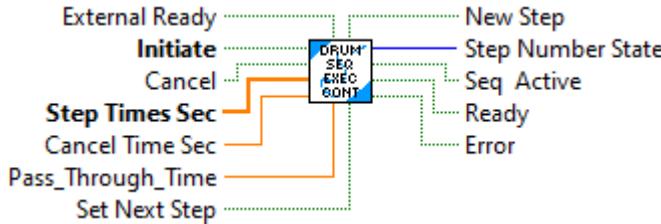
DoubleSolenoid

DoubleSolenoid_Pulse_Execute



DrumSequence

DrumSequence_Cont_Execute



Implements a simplified state machine similar to a mechanical / electrical drum sequencer. When the Drum Sequencer is "Ready" and the "Initiate" input becomes TRUE,, the drum sequences through its set of steps from 1 to N. After each Step is executed it is followed by a wait. During the wait the Step output indicates "Do Nothing". The number of Steps is determined by the number of elements in the "Step Times" input array. Once all the steps have been executed, the Drum Sequencer cycles bac to wait for another initiate.

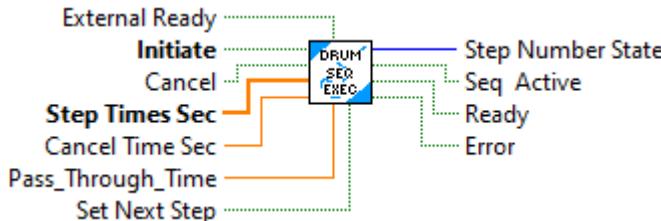
There is a Cancel input that interrupts the sequence, executes the Cancel Step, performs the cancel wait, and then returns to wait for an Initiate action.

If any of the Step Times, or Cancel times are negative, or there more than 32 steps, or less than 1 step an error is signaled. The drum sequencer selects the error state and stays there.

Inputs:

Outputs:

DrumSequence_Pulse_Execute



Implements a simplified state machine similar to a mechanical / electrical drum sequencer. When the Drum Sequencer is "Ready" and the "Initiate" input becomes TRUE,, the drum sequences through its set of steps from 1 to N. After each Step is executed it is followed by a wait. During the wait the Step output indicates "Do Nothing". The number of Steps is determined by the number of elements in the "Step Times" input array. Once all the steps have been executed, the Drum Sequencer cycles bac to wait for another initiate.

There is a Cancel input that interrupts the sequence, executes the Cancel Step, performs the cancel wait, and then returns to wait for an Initiate action.

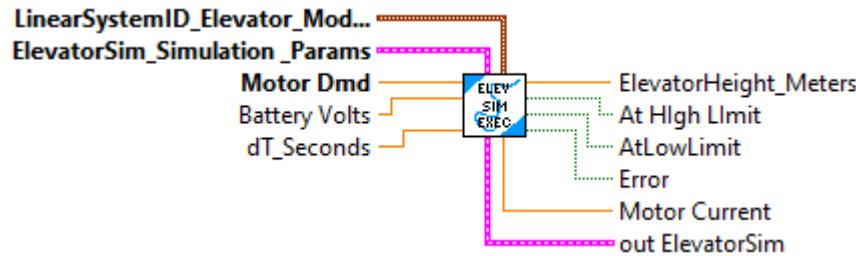
If any of the Step Times, or Cancel times are negative, or there more than 32 steps, or less than 1 step an error is signaled. The drum sequencer selects the error state and stays there.

Inputs:

Outputs:

ElevatorSim

ElevatorSim_Execute



Single call LabVIEW function to simulate an Elevator system.

The Elevator linear system definition is:

- States
 - position (meters)
 - velocity (meters/sec)
- Inputs:
 - motor voltage (volts)
- Outputs:
 - position (meters)

Inputs:

- LinearSystem Elevator Model Params -- cluster -- Contains physical configuration for Elevator system.
- Elevator Sim Simulation Params -- cluster -- Contains siulation configuration.
- Motor Dmd (CO) -- double -- motor demand value (+/- 1.0)
- Battery Voltage -- double -- Current simulated battery voltage (Volts)
- dT Sec -- double -- Update period (Seconds, Default: 0.02)

Outputs:

- ElevatorHeight -- double -- Current elevator height (meters)

- At High Limit -- boolean -- Elevator has reached high height limit
 - At Low Limit -- boolean -- Elevator has reached low height limit
 - Error -- boolean -- If TRUE an error occurred.
 - Motor Current -- double -- Current motor current (Amps)
 - outElevatorSim -- cluster -- Current internal data cluster.
-

ElevatorSim_GetCurrentDraw



Returns the elevator current draw.

Inputs:

- ElevatorSim -- Data cluster

Outputs:

- Current_Amps -- The elevator current draw. (Amps)
-

ElevatorSimGetPositionMeters



Returns the position of the elevator.

Inputs:

- ElevatorSim -- Data cluster

Outputs:

- Position_Meters -- The position of the elevator. (Meters)
-
-

ElevatorSim_GetVelocityMetersPerSecond



Returns the velocity of the elevator.

Inputs:

- ElevatorSim -- Data cluster

Outputs:

- Velocity_MetersPerSec -- The velocity of the elevator. (M/S)
-
-

ElevatorSim_HasHitLowerLimit



Returns whether the elevator has hit the lower limit.

Inputs:

- ElevatorSim -- Data cluster

Outputs:

- AtLowLimit -- Whether the elevator has hit the lower limit.
-
-

ElevatorSim_HasHitUpperLimit



Returns whether the elevator has hit the upper limit.

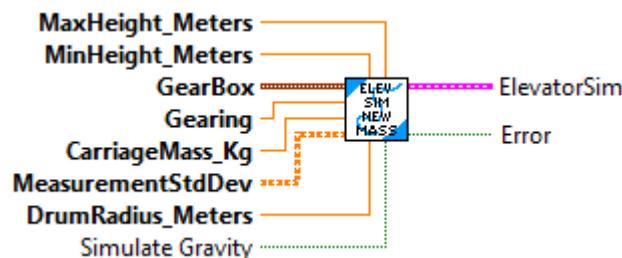
Inputs:

- ElevatorSim -- Data cluster

Outputs:

- AtHighLimit -- Whether the elevator has hit the upper limit.

ElevatorSim_New



Creates a simulated elevator mechanism.

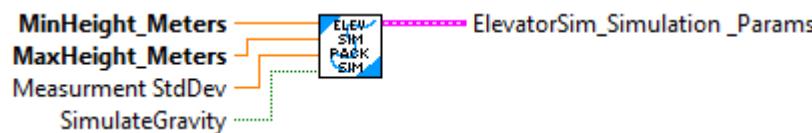
Inputs:

- maxHeightMeters -- The max allowable height of the elevator.
- minHeightMeters -- The min allowable height of the elevator.
- gearbox -- The type of and number of motors in the elevator gearbox.
- gearing -- The gearing of the elevator (numbers greater than 1 represent reductions).
- carriageMassKg -- The mass of the elevator carriage.
- MeasurementStdDev -- Vector matrix of std deviations for measurements.
- drumRadiusMeters -- The radius of the drum that the elevator spool is wrapped around.
- SimulateGravity -- Simulate gravity (optional) Default: True

Outputs:

- OutElevatorSim -- Updated data cluster
 - Error -- If TRUE, an error occurred.
-

ElevatorSim_Pack_Simulation_Params



Pack simulation parameters used by the Elevator Simulation Execute routine.

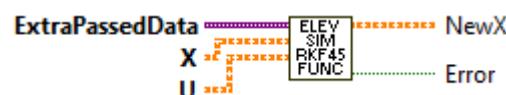
Inputs:

- Min Height -- double -- Minimum physical elevator height (meters)
- Max Height -- double -- Maximum physical elevator height (meters)
- Pos Sim Std Dev -- double -- Standard deviation for the motor position (meters Default: 0.09)
- SimulateGravity -- boolean -- Simulate gravity (Default: True)

Outputs:

- Elevator Sim Simulation -- cluster -- Packed data for Execute function
-

ElevatorSim_Rkf45_Func



Callback function to pass as a strict reference to numerical integration routine to calculate $F(X, U, \text{extra})$

This is an internal function.

The calculation performed is:

$$\text{NewX} = A \times X + B \times U + K$$

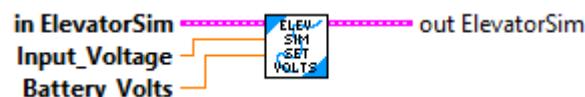
Inputs:

- Variant -- Extra data to pass to the function. The format of the data is specific to this routine. In this case the data contains:
 - Current A matrix
 - Current B matrix
 - K matrix
- X -- X matrix
- U -- U matrix

Output:

- NewX -- Calculated new value of X
- Error -- If TRUE, an error occurred.

ElevatorSim_SetInputVoltage



Sets the input voltage for the elevator.

Inputs:

- ElevatorSim -- Data cluster
- volts -- The input voltage. (volts)
- BatteryVoltage -- The current battery voltage (volts)

Outputs:

- OutElevatorSim -- Updated data cluster

ElevatorSim_SetState



Sets the system state.

Inputs:

- ElevatorSim -- Data cluster
- state -- The new state.

Outputs:

- OutElevatorSim -- Updated data cluster
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

ElevatorSim_Update



Updates the simulation.

Inputs:

- ElevatorSim -- Data cluster
- dtSeconds -- The time between updates.

Outputs:

- OutElevatorSim -- Updated data cluster
- Error -- If TRUE, an error occurred.

ElevatorSim_UpdateX



Updates the state of the elevator.

Inputs:

- ElevatorSim -- Data cluster
- currentXhat -- The current state estimate.
- u -- The system inputs (voltage).
- dtSeconds -- he time difference between controller updates.

Outputs:

- OutElevatorSim -- Updated data cluster
- Error -- If TRUE, an error occurred.

ElevatorSim_WouldHitLowerLimit



Returns whether the elevator would hit the lower limit.

Inputs:

- ElevatorSim -- Data cluster
- elevatorHeightMeters -- The elevator height.

Outputs:

- WouldHitLowLimit -- Whether the elevator would hit the lower limit.

ElevatorSim_WouldHitUpperLimit



Returns whether the elevator would hit the upper limit.

Inputs:

- ElevatorSim -- Data cluster
- elevatorHeightMeters -- The elevator height.

Outputs:

- OutElevatorSim -- Updated data cluster
- WouldHitUpperLimit -- Whether the elevator would hit the upper limit.

ElevFF

ElevFF_Calculate



Calculates the feedforward from the gains and setpoints.

Inputs:

- ElevFF -- The ElevFF data cluster
- velocity -- The velocity setpoint.
- acceleration -- The acceleration setpoint.

Outputs:

- feedforward -- The computed feedforward.

ElevFF_CalculateVelocityOnly



Calculates the feedforward from the gains and velocity setpoint (acceleration is assumed to be zero).

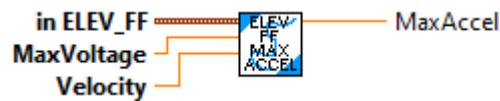
Inputs:

- ElevFF -- The ElevFF data cluster
- velocity -- The velocity setpoint.

Outputs:

- feedforward -- The computed feedforward.

ElevFF_MaxAchieveAccel



Calculates the maximum achievable acceleration given a maximum voltage supply and a velocity. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the velocity constraint, and this will give you a simultaneously-achievable acceleration constraint.

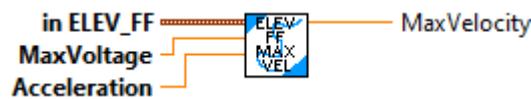
Inputs:

- ElevFF -- The ElevFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the elevator.
- velocity -- The velocity of the elevator.

Outputs:

- MaxAccel -- The maximum possible acceleration at the given velocity.

ElevFF_MaxAchieveVelocity



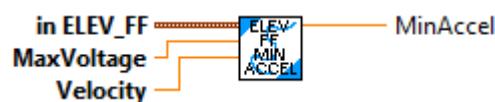
Calculates the maximum achievable velocity given a maximum voltage supply and an acceleration. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the acceleration constraint, and this will give you a simultaneously-achievable velocity constraint.

Inputs:

- ElevFF -- The ElevFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the elevator.
- acceleration -- The acceleration of the elevator.

Outputs:

- MaxVelocity -- The maximum possible velocity at the given acceleration.

ElevFF_MinAchieveAccel

Calculates the minimum achievable acceleration given a maximum voltage supply and a velocity. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the velocity constraint, and this will give you a simultaneously-achievable acceleration constraint.

Inputs:

- ElevFF -- The ElevFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the elevator.
- velocity -- The velocity of the elevator.

Outputs:

- MinAccel -- The minimum possible acceleration at the given velocity.

ElevFF_MinAchieveVelocity



Calculates the minimum achievable velocity given a maximum voltage supply and an acceleration. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable
- enter the acceleration constraint, and this will give you a simultaneously-achievable velocity constraint.

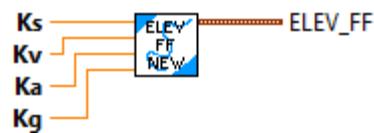
Inputs:

- ElevFF -- The ElevFF data cluster
- maxVoltage -- The maximum voltage that can be supplied to the elevator.
- acceleration -- The acceleration of the elevator.

Outputs:

- MinVelocity -- The minimum possible velocity at the given acceleration.

ElevFF_New



Creates a new ElevatorFeedforward data cluster with the specified gains. Units of the gain values will dictate units of the computed feedforward.

This is a helper set of subVIs that computes feedforward outputs for a simple elevator (modeled as a motor acting against the force of gravity).

Inputs:

- ks -- The static gain.

- kg -- The gravity gain.
- kv -- The velocity gain.
- ka -- The acceleration gain.

Outputs:

- ElevFF -- The initialized ElevFF data cluster
-
-

ElevFF_New_ZeroAccel



Creates a new ElevatorFeedforward data cluster with the specified gains. Acceleration gain is defaulted to zero. Units of the gain values will dictate units of the computed feedforward.

This is a helper set of subVIs that computes feedforward outputs for a simple elevator (modeled as a motor acting against the force of gravity).

Inputs:

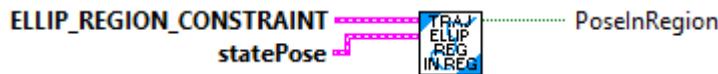
- ks -- The static gain.
- kg -- The gravity gain.
- kv -- The velocity gain.

Outputs:

- ElevFF -- The initialized ElevFF data cluster

EllipRegionConstraint

EllipRegionConstraint_IsPoseInRegion



Determines if the robot pose is within the defined region.

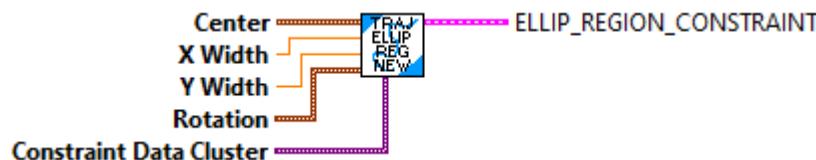
Parameters:

- Elliptical Region Constraint -- Constraint data cluster.
- statePose - current traj state Pose

Returns

- PoseInRegion -- TRUE if pose is within the region.

EllipRegionConstraint_New



Constructs a new EllipticalRegionConstraint.

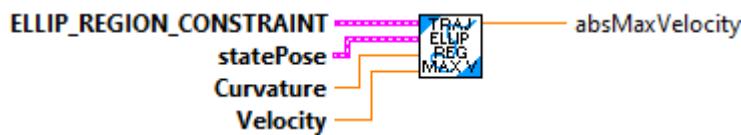
Parameters:

- center -- The center of the ellipse in which to enforce the constraint.
- xWidth -- The width of the ellipse in which to enforce the constraint.
- yWidth -- The height of the ellipse in which to enforce the constraint.
- rotation -- The rotation to apply to all radii around the origin.
- constraint -- The constraint to enforce when the robot is within the region.

Returns

- Ellip Region Constraint - Constraint data structure.

EllipRegionConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

Ensure each individual normalized wheel speed is within the defined maximum velocity.

Parameters:

- EllipRegionConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

EllipRegionConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values.

Parameters:

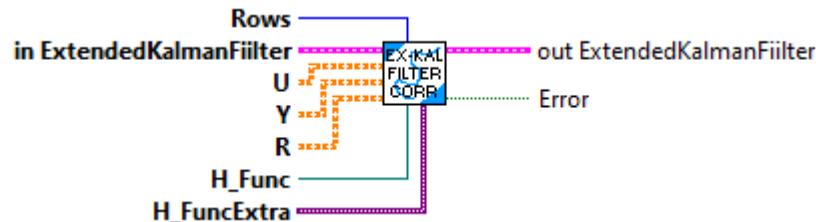
- EllipRegionConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

ExtendedKalmanFilter

ExtendedKalmanFilter_Correct



Correct the state estimate \hat{x} using the measurements in y .

This is useful for when the measurements available during a timestep's `Correct()` call vary.

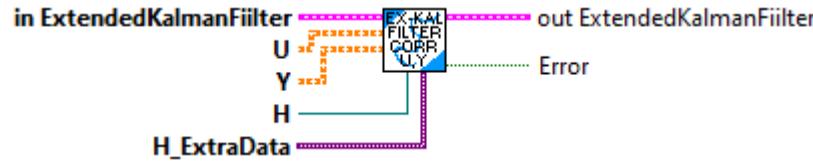
Inputs:

- `inExtendedKalmanFilter` -- filter data cluster
- `rows` -- Number of rows in the result of $f(x, u)$.
- `u` -- Same control input used in the predict step.
- `y` -- Measurement vector.
- `R` -- Discrete measurement noise covariance matrix.
- `h_Func` -- A vector-valued function reference of x and u that returns the measurement vector.
- `H_func_extra` -- extra data, if any, used by `H_Func`

Returns:

- `outExtendedKalmanFilter` -- updated filter data cluster
- `error` -- If TRUE, an error occurred.

ExtendedKalmanFilter_Correct_OnlyUY



Correct the state estimate \hat{x} using the measurements in y .

Inputs:

- inExtendedKalmanFIltter -- filter data cluster
- u -- Same control input used in the predict step.
- y -- Measurement vector.
- h_Func -- A vector-valued function reference of x and u that returns the measurement vector.
- H_func_extra -- extra data, if any, used by H_Func

Returns:

- outExtendedKalmanFIltter -- updated filter data cluster
- error -- If TRUE, an error occurred

ExtendedKalmanFilter_GetP



Returns the error covariance matrix P .

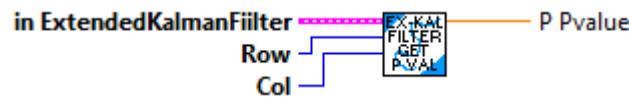
Inputs:

- inExtendedKalmanFIltter -- filter data cluster

Output:

- P -- the error covariance matrix P .

ExtendedKalmanFilter_GetP_Single



Returns an element of the error covariance matrix P.

Inputs:

- inExtendedKalmanFIltter -- filter data cluster
- row Row of P.
- col Column of P.

Returns:

- outExtendedKalmanFIltter -- updated filter data cluster
 - P_Value -- the value of the error covariance matrix P at (i, j).
-
-

ExtendedKalmanFilter_GetXHat



Returns the state estimate x-hat.

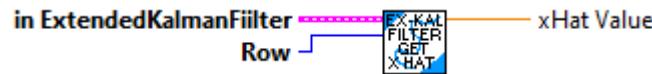
Inputs:

- inExtendedKalmanFIltter -- filter data cluster

Returns:

- xHat -- the state estimate x-hat.
-
-

ExtendedKalmanFilter_GetXHat_Single



Returns an element of the state estimate x-hat.

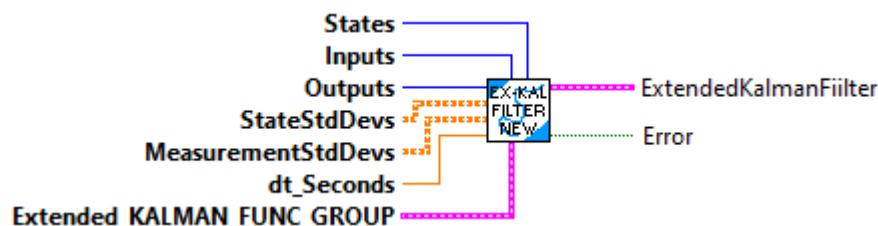
Inputs:

- inExtendedKalmanFIltter -- filter data cluster
- row Row of x-hat.

Returns:

- xHat_value -- the value of the state estimate x-hat at i.

ExtendedKalmanFilter_New



Kalman filters combine predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

The Extended Kalman filter is just like the KalmanFilter Kalman filter, but we make a linear approximation of nonlinear dynamics and/or nonlinear measurement models. This means that the EKF works with nonlinear systems.

Constructs an extended Kalman filter.

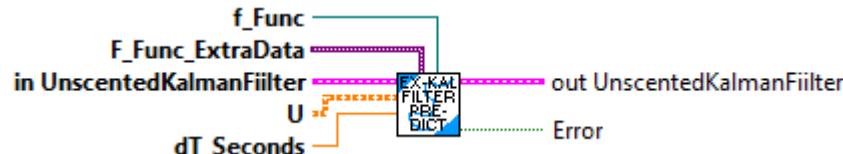
- states -- the number of states.
- inputs -- the number of inputs.
- outputs -- the number of outputs.

- ExtendedKalmanFuncGroup -- definition of external functions used by this filter. it includes:
 - f_func -- A vector-valued function of x and u that returns the derivative of the state vector.
 - f_FuncExtra -- extra data, if any, used by the F_Func
 - h_func -- A vector-valued function of x and u that returns the measurement vector.
 - h_FuncExtra -- extra data, if any, used by the H_Func
- stateStdDevs -- Standard deviations of model states.
- measurementStdDevs -- Standard deviations of measurements.
- dtSeconds -- Nominal discretization timestep.

Outputs:

- ExtendedKalmanFilter -- filter data cluster

ExtendedKalmanFilter_Predict



Project the model into the future with a new control input u.

Inputs:

- inExtendedKalmanFilter -- filter data cluster
- u New control input from controller.
- F_Func -- reference a function used to linearize the model.
- F_FuncExtraData -- extra data, if any, for the F_Func.
- dtSeconds Timestep for prediction.

Returns:

- outExtendedKalmanFilter -- updated filter data cluster
 - Error -- If TRUE, an error occurred.
-

ExtendedKalmanFilter_Reset



Resets the filter.

Inputs:

- inExtendedKalmanFilter -- filter data cluster

Returns:

- outExtendedKalmanFilter -- updated filter data cluster
-

ExtendedKalmanFilter_SetP



Sets the entire error covariance matrix P.

Inputs:

- inExtendedKalmanFilter -- filter data cluster
- newP The new value of P to use.

Returns:

- outExtendedKalmanFilter -- updated filter data cluster
- sizeCoerced -- If TRUE, an unexpected error occurred.

ExtendedKalmanFilter_SetXHat



Set initial state estimate x-hat.

Inputs:

- inExtendedKalmanFIltter -- filter data cluster
- xHat The state estimate x-hat.

Returns:

- outExtendedKalmanFIltter -- updated filter data cluster
- sizeCoerced -- If TRUE, an unexpected error occurred.

ExtendedKalmanFilter_SetXHat_Single



Set an element of the initial state estimate x-hat.

Inputs:

- inExtendedKalmanFIltter -- filter data cluster
- row Row of x-hat.
- value Value for element of x-hat.

Returns:

- outExtendedKalmanFIltter -- updated filter data cluster
- Error -- If TRUE, an error occurred.

FieldDisp

FieldDisp_Convert_IMAQ_Image_to_Picture



Internal function -- should not be called directly by users.

Complete documentation TBD.

FieldDisp_Element_Dispatch



Displays, moves and rotates the field element on the display.

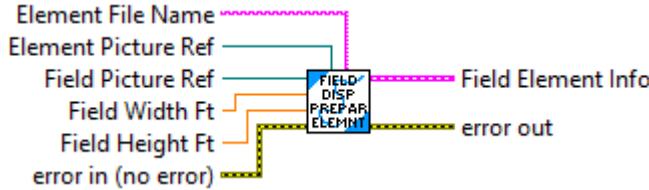
Inputs:

- Element Info -- cluster -- information defining the field element.
- Element Pose -- Pose2d -- Location and orientation of the field element.
- Visible -- boolean -- The field element is visible. This allows tabbed controls to make elements invisible when the field tab is not shown. (Optional. Default: TRUE)

Outputs:

-- NONE --

FieldDisp_Element_Prep



Prepare a field element (robot, game piece, etc.) for display on a field. This routine opens the image file and prepares the image for display, movement, and rotation.

Image files can be PNG, BMP, or JPG.

Inputs:

- Element File Name -- String -- File name of element to display. Relative paths start in the root directory of the application.
- Element Picture Ref -- Picture Reference -- Reference to the control used to display the field element.
- Field Picture Ref -- Picture Reference -- Reference to the control used to display the field.
- Field Width -- double -- Width of the field (feet). This is used to initialize element positioning.
- Field Height -- double -- Height of the field (feet). This is used to initialize element positioning.
- Error in -- error cluster -- input error. Often used to sequence VIs.

Outputs:

- Field Element Info -- cluster -- Contains information used for display of element.
- Error out -- error cluster -- contains error information.

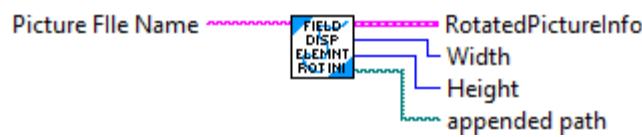
FieldDisp_Element_Rotate



Internal function -- should not be called directly by users.

Complete documentation TBD.

FieldDisp_Element_Rotate_Init

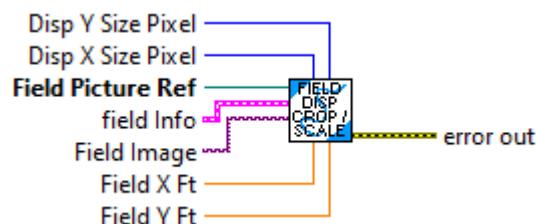


Internal function -- should not be called directly by users.

Complete documentation TBD.

Initialize a PNG image to be rotated.

FieldDisp_Field_Crop_and_Scale



Internal function -- should not be called directly by users.

Complete documentation TBD.

FieldDisp_Field_Disp



Read the specified field image file, scale and crop the image, then update the picture control with the image.

The field information INI file must be in a FieldInfo subdirectory under the application's main directory. The field image files are also located in this subdirectory.

Inputs:

- Picture In -- Picture Control Reference -- Reference to the picture control to display the field..
- Field Selector -- String -- String containing the name of the field to display. This can be the output of a combo box control.

Outputs:

- Error out -- error cluster -- If an error occurred, contains error information.

FieldDisp_Field_Selector_Prep



Read the field information INI file and update a ComboBox control with the list of available fields.

The field information INI file must be in a FieldInfo subdirectory under the application's main directory. The field image files are also located in this subdirectory.

Inputs:

- ComboBox Ref -- ComboBox Control Reference -- Reference to the control to be updated.

Outputs:

- Section Names -- String array -- List of field names.
- Error out -- error cluster -- If an error occurred, contains error information.

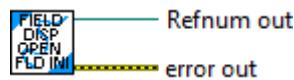
FieldDisp_Get_Field_Info



Internal function -- should not be called directly by users.

Complete documentation TBD.

FieldDisp_Open_Field_Info_File



Internal function -- should not be called directly by users.

Complete documentation TBD.

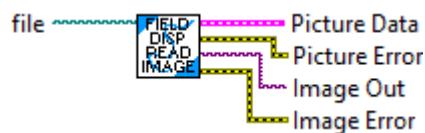
FieldDisp_Read_Field_Pic



Internal function -- should not be called directly by users.

Complete documentation TBD.

FieldDisp_Read_Image_File

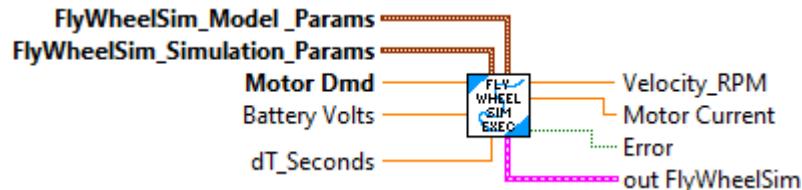


Internal function -- should not be called directly by users.

Complete documentation TBD.

FlyWheelSim

FlyWheelSim_Execute



Single call LabVIEW function to simulate a Flywheel system.

The flywheel linear system definition is:

- States
 - angular velocity (Rad/Sec)
- Inputs:
 - motor voltage
- Outputs:
 - angular velocity (rad/sec)

Inputs:

- LinearSystem Flywheel Model Params -- cluster -- Contains physical configuration for Flywheel system.
- Flywheel Sim Simulation Params -- cluster -- Contains siulation configuration.
- Motor Dmd (CO) -- double -- motor demand value (+/- 1.0)
- Battery Voltage -- double -- Current simulated battery voltage (Volts)
- dT Sec -- double -- Update period (Seconds, Default: 0.02)

Outputs:

- Velocity RPM -- double -- Current flywheel velocity (RPM)
- Motor Current -- double -- Current motor current (Amps)
- Error -- boolean -- If TRUE an error occured.

- outFlyWheelSim -- cluster -- Current internal data cluster.
-
-

FlyWheelSim_GetCurrentDrawAmps



Returns the flywheel current draw.

Inputs:

- FlyWheelSim -- Data cluster

Outputs:

- Current_Amps -- The flywheel current draw. (Amps)
-
-

FlyWheelSim_New_MOI



Creates a simulated flywheel mechanism.

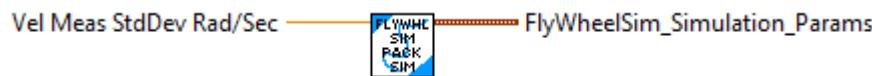
Inputs:

- gearbox -- The type of and number of motors in the flywheel gearbox.
- gearing -- The gearing of the flywheel (numbers greater than 1 represent reductions).
- jKgMetersSquared -- The moment of inertia of the flywheel.
- measurementStdDevs -- The standard deviations of the measurements.

Outputs:

- outFlyWheelSim -- Updated data cluster
 - Error -- If TRUE, an error occurred.
-

FlyWheelSim_Pack_Simulation_Params



Pack simulation parameters used by the Flywheel Simulation Execute routine.

Inputs:

- Vel Sim Std Dev -- double -- Standard deviation for the flywheel velocity (rad/sec Default: 1)

Outputs:

- Flywheel Sim Simulation -- cluster -- Packed data for Execute function
-

FlyWheelSim_SetInput



Sets the input voltage for the flywheel.

Inputs:

- FlyWheelSim -- Data cluster
- volts -- The input voltage.
- BatteryVolts -- Current battery voltage.

Outputs:

- outFlyWheelSim -- Updated data cluster

FlyWheelSim_SetState



Sets the system state.

Inputs:

- FlyWheelSim -- Data cluster
- state -- The new state.

Outputs:

- OutFlyWheelSim -- Updated data cluster
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

FlyWheelSim_Update



Updates the simulation.

Inputs:

- FlyWheelSim -- Data cluster
- dtSeconds -- The time between updates.

Outputs:

- OutFlyWheelSim -- Updated data cluster
- Error -- If TRUE, an error occurred.

FlyWheelSim_getAngularVelocityRPM



Returns the flywheel velocity in RPM.

Inputs:

- FlyWheelSim -- Data cluster

Outputs:

- Velocity_RPM -- The flywheel velocity in RPM.

FlyWheelSim_getAngularVelocityRadPerSec



Returns the flywheel velocity.

Inputs:

- FlyWheelSim -- Data cluster

Outputs:

- Velocity_RadPerSec -- The flywheel velocity. (Rad/Sec)

FunctionGenerator

FunctionGenerator_Add_Value



Insert new X, Y pair into an existing Function Generator data cluster. If the X values matches an existing X value, the new X and Y values replaces the old values.

Inputs:

- In Function Generator -- Function generator data cluster
- X -- X value
- Y -- Y value

Outputs:

- out Function Generator -- updated data structure

FunctionGenerator_Add_XY



Insert new X, Y pair into an existing Function Generator data cluster. If the X values matches an existing X value, the new X and Y values replaces the old values.

Inputs:

- In Function Generator -- Function generator data cluster
- XY -- Pair of X, Y value

Outputs:

- out Function Generator -- updated data structure

FunctionGenerator_Calculate



Calculate the output Y value for the provided X. This is done by interpolating through an array of X and Y pairs. The pairs must be ordered in increasing X value.

If there's no matching key, the value returned will be a linear interpolation between the keys before and after the provided one.

Inputs:

- Function_Generator -- function generator data structure
- Input Value -- The input value.

Outputs:

- Output Value -- Output value calculated from the input value.

FunctionGenerator_Clear



Clear out all X, Y entries

Inputs:

- in Function Generator -- Data cluster

Outputs:

- Function Generator -- Updated data cluster

FunctionGenerator_Execute



Convenience, single call, LabVIEW function. Creates and calculates the function generator.

If there's no matching key, the value returned will be a linear interpolation between the keys before and after the provided one.

Inputs:

- X Y Pairs -- Array of pairs of X and Y values
- Input Value -- Input "X" value

Outputs:

- Output Value -- Calculated value "Y" for the input "X" value.

FunctionGenerator_New



Create a new Function Generator. Function Generators (Interpolating Tree Maps) are used to get values at points that are not defined by making a guess from points that are defined. This uses linear interpolation.

Inputs:

- X Y Pairs -- (Optional) Array of pairs of X and Y values used to define this function generator. Additional X Y pairs can be added later.

Outputs:

- Function Generator -- Created data structure

FunctionGeneratorMatrix

FunctionGeneratorMatrix_Add_Value



Insert new X, Y pair into an existing Function Generator Matrix data cluster.

Inputs:

- Input Funct Gen Matrix -- Data cluster
- X Value -- New X value
- Y Value -- New Y matrix

Outputs:

- Function Generator Matrix -- Updated data cluster
- Error -- Set true if an error occurred. This is set when the size of the Y matrix does not match the definition of this function generator.

FunctionGeneratorMatrix_Calculate



Calculate the output Y value for the provided X. This is done by interpolating through an array of X and Y pairs. The pairs must be ordered in increasing X value. (The Add function ensures that the X values are sorted properly.)

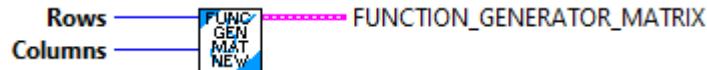
Inputs:

- Function_Generator Matrix -- function generator data structure
- Input Value -- The input value.

Outputs:

- Output Value -- Output matrix calculated from the input value.
-

FunctionGeneratorMatrix_New



Function generator matrix (Interpolating Tree Maps) are used to get values at points that are not defined by making a guess from points that are defined. This uses linear interpolation. This type returns a Y matrix as a function of a provided double X value.

Inputs:

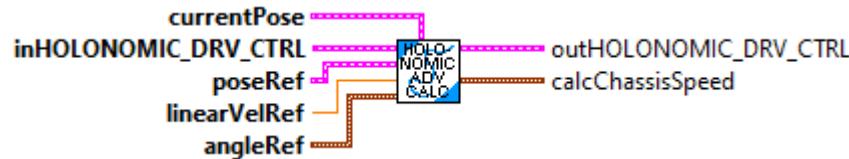
- Rows -- Number of rows in the Y matrix
- Columns -- Number of columns in the Y matrix.

Outputs:

- Function Generator Matrix -- Created data cluster

HolDrvCtrl

HolDrvCtrl_AdvCalculate



Returns the next output of the holonomic drive controller.

This version uses the Advanced PID instead of the standard PID for X and Y control.

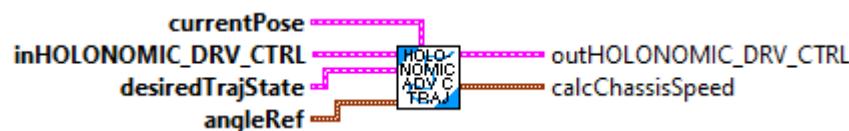
Inputs:

- inHolonomicDrvCtrl -- controller datat structure
- currentPose -- The current pose.
- poseRef -- The desired pose.
- linearVelocityRefMeters -- The linear velocity reference.
- angleRef -- The angular reference.

Outputs:

- outHolonomicDrvCtrl -- controller datat structure
- calcChassisSpeed -- The next output of the holonomic drive controller.

HolDrvCtrl_AdvCalculate_Trajectory



Returns the next output of the holonomic drive controller.

This version uses the Advanced PID instead of the standard PID for X and Y control.

Inputs:

- inHolonomicDrvCtrl -- controller data structure
- currentPose -- The current pose.
- desiredState -- The desired trajectory state.
- angleRef -- The desired end-angle.

Outputs:

- outHolonomicDrvCtrl -- updated controller data structure
- calcChassisSpeed -- The next output of the holonomic drive controller.

HolDrvCtrl_AtReference



Returns true if the pose error is within tolerance of the reference.

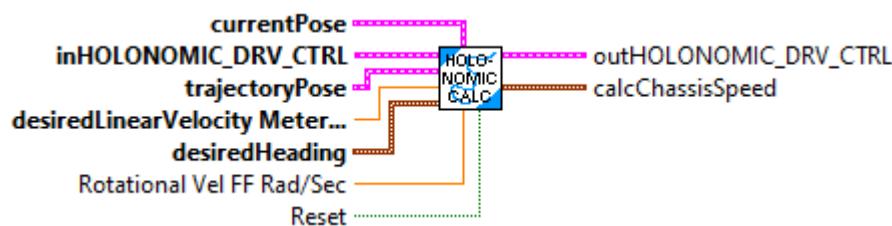
Parameters:

- Holonomic_Drv_Ctrl - Holonomic_Drv_Ctrl data structure

Returns:

- At Reference - Return value

HolDrvCtrl_Calculate



Returns the next output of the holonomic drive controller.

`@return The next output of the holonomic drive controller.Returns the next output of the holonomic drive controller.`

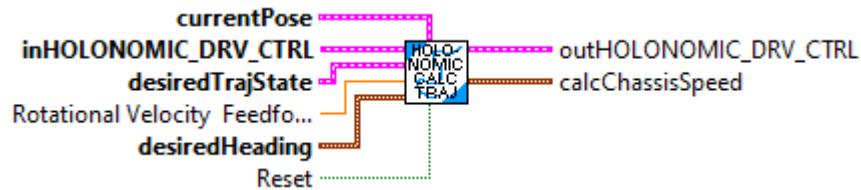
Inputs:

- `inHolonomicDrvCtrl` -- cluster -- controller data structure
- `currentPose` -- pose2d -- The current pose, as measured by odometry or pose estimator.
- `trajectoryPose` -- pose2d -- The desired trajectory pose, as sampled for the current timestep.
- `desiredLinearVelocityMetersPerSecond` -- double -- The desired linear velocity.
- `desiredHeading` -- rotation2d -- The desired heading.

Outputs:

- `outHolonomicDrvCtrl` -- controller data structure
- `calcChassisSpeed` -- ChassisSpeeds -- The next output of the holonomic drive controller.

HolDrvCtrl_Calculate_Trajectory



Returns the next output of the holonomic drive controller.

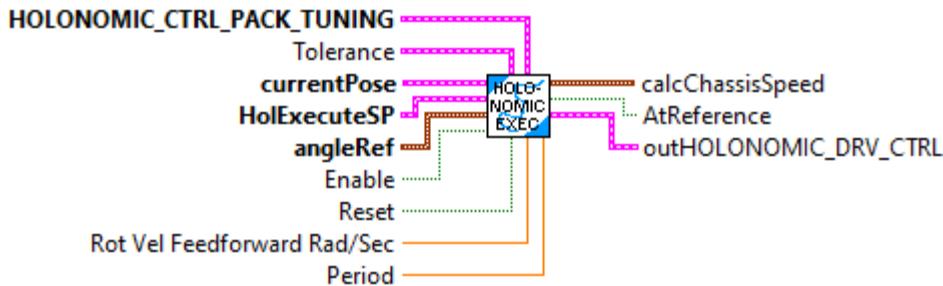
Inputs:

- `inHolonomicDrvCtrl` -- controller data structure
- `currentPose` -- The current pose, as measured by odometry or pose estimator.
- `desiredState` -- The desired trajectory pose, as sampled for the current timestep.
- `desiredHeading` -- The desired heading..

Outputs:

- outHolonomicDrvCtrl -- updated controller data structure
 - calcChassisSpeed -- The next output of the holonomic drive controller.
-

HolDrvCtrl_Execute



This holonomic drive controller can be used to follow trajectories using a holonomic drivetrain (i.e. swerve or mecanum). Holonomic trajectory following is a much simpler problem to solve compared to skid-steer style drivetrains because it is possible to individually control forward, sideways, and angular velocity.

The holonomic drive controller takes in one PID controller for each direction, forward and sideways, and one profiled PID controller for the angular direction. Because the heading dynamics are decoupled from translations, users can specify a custom heading that the drivetrain should point toward. This heading reference is profiled for smoothness.

This convenience function creates and executes a Holonomic Controller. It uses a packed Setpoint (SP), created by the HolDrvCtrl_PackSP.vi, as the setpoint and the current pose as the process variable. It returns a chassis speed as the control output.

NOTE: This version does not support dynamic tuning. All the uint inputs and Tolerance are used only on the first call.

Inputs:

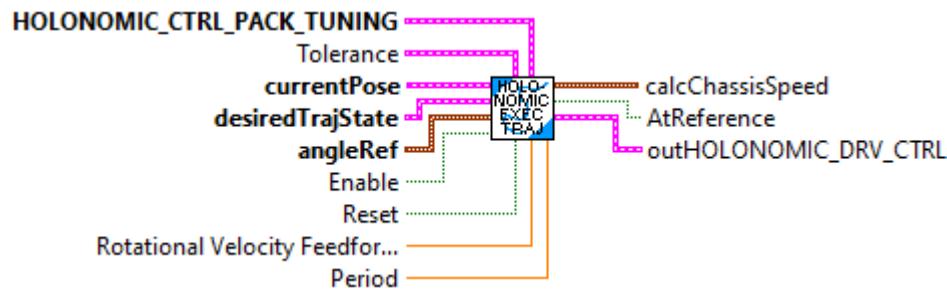
- X PID Tuning - Input from PACK PROF PID VI
- Y PID Tuning - Input from PACK PROF PID VI
- thetaController -- A profiled PID controller to respond to error in angle.
- CurrentPose -- Current location
- HolExecuteSP -- Desired location and speed cluster. This contains:

- Desired Pose -- X,Y,Heading
- Linear Velocity -- Desired velocity
- AngleRef -- Current angle of orientation.
- Enable -- Enable close loop control. If disabled, the feedforwards from the trajectory state are used.
- Period -- Execution period (Default 0.020) Seconds.
- Tolerance -- Amount of position and heading error allowed to be considered on target. (Default 0.04, 0.04, 0.034)

Outputs:

- ChassisSpeed -- ChassisSpeed setpoint calculated from the inputs.
- AtTarget -- Boolean indicating the position and heading are within the provided tolerance.
- HolonomicDriveController -- Data structure (can be used for debug or extracting information)

HolDrvCtrl_Execute_Trajectory



This holonomic drive controller can be used to follow trajectories using a holonomic drivetrain (i.e. swerve or mecanum). Holonomic trajectory following is a much simpler problem to solve compared to skid-steer style drivetrains because it is possible to individually control forward, sideways, and angular velocity.

The holonomic drive controller takes in one PID controller for each direction, forward and sideways, and one profiled PID controller for the angular direction. Because the heading dynamics are decoupled from translations, users can specify a custom heading that the drivetrain should point toward. This heading reference is profiled for smoothness.

This convenience function creates and executes a Holonomic Controller. It uses a Trajectory state as the setpoint and the current pose as the process variable. It returns a chassis speed as the control output.

NOTE: This version does not support dynamic tuning. All the uint inputs and Tolerance are used only on the first call.

Inputs:

- X PID Tuning - Input from PACK PROF PID VI
- Y PID Tuning - Input from PACK PROF PID VI
- thetaController -- A profiled PID controller to respond to error in angle.
- CurrentPose -- Current location
- DesiredTrajectoryState - Desired heading, speed, and location.
- DesiredAngle -- Desired angle of orientation.
- RotationalVelocityFeedforward -- Desired rotational velocity Rad/Sec
- Enable -- Enable close loop control. If disabled, the feedforwards from the trajectory state are used.
- Reset -- Reset controllers. Do this at the start of following a new trajectory.
- Period -- Execution period (Default 0.020) Seconds.
- Tolerance -- Amount of position and heading error allowed to be considered on target. (Default 0.04, 0.04, 0.034)

Outputs:

- ChassisSpeed -- ChassisSpeed setpoint calculated from the inputs.
- AtTarget -- Boolean indicating the position and heading are within the provided tolerance.
- HolonomicDriveController -- Data structure (can be used for debug or extracting information)

HolDrvCtrl_New



This holonomic drive controller can be used to follow trajectories using a holonomic drivetrain (i.e. swerve or mecanum). Holonomic trajectory following is a much simpler problem to solve compared to skid-steer style drivetrains because it is possible to individually control forward, sideways, and angular velocity.

The holonomic drive controller takes in one PID controller for each direction, forward and sideways, and one profiled PID controller for the angular direction. Because the heading dynamics are decoupled from translations, users can specify a custom heading that the drivetrain should point toward. This heading reference is profiled for smoothness.

This VI constructs a holonomic drive controller data structure.

Inputs:

- xController -- A PID Controller to respond to error in the field-relative x direction.
- yController -- A PID Controller to respond to error in the field-relative y direction.
- thetaController -- A profiled PID controller to respond to error in angle.

Output:

- outHolonomicDrvCtrl -- controller data structure.

HolDrvCtrl_PackControllers



Pack controller tuning configuration for Holonomic Drive Controller.

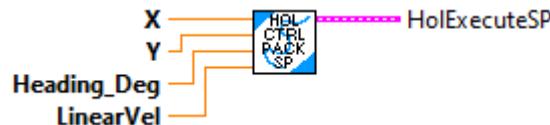
Inputs:

- X PID Tuning - Input from PACK PROF PID VI
- Y PID Tuning - Input from PACK PROF PID VI
- thetaController -- A profiled PID controller to respond to error in angle.

Outputs:

- Holonomic_Ctrl_Packed -- cluster - packed controller configuration.

HolDrvCtrl_PackExecuteSP



Convenience, single call, LabVIEW function. Packs individual values into a cluster to feed to the Holonomic controller execute function.

Inputs:

- X -- X part of desired position
- Y -- Y part f desired position
- Heading_Deg -- Travel heading angle part of desired position (Degrees)
- LinearVel -- Velocity in the direction of heading.

Outputs:

- HolExecuteSP -- Setpoint (desired position and velocity) to feed to Holonomic Execute function.

HolDrvCtrl_PackPID



Convenience, single call, LabVIEW function. Packs "standard" constants for the X and Y PID controllers used by the holonomic controller.

NOTE -- This version does NOT support dynamic tuning. The constants are read only used during the first execution cycle.

Inputs:

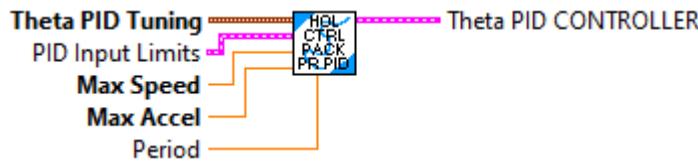
- X PID Tuning -- PID Tuning parameters containing:
 - Kp -- The proportional coefficient.
 - Ki -- The integral coefficient.

- Kd -- The derivative coefficient.
- Maximum Integral -- The maximum value of the integrator.
- Minimum Integral -- The minimum value of the integrator.
- Y PID Tuning -- PID Tuning parameters containing:
 - Kp -- The proportional coefficient.
 - Ki -- The integral coefficient.
 - Kd -- The derivative coefficient.
 - Maximum Integral -- The maximum value of the integrator.
 - Minimum Integral -- The minimum value of the integrator.
- Period -- Period of repeated calls in seconds. (Default: 0.020)

Outputs:

- X PID_Controller -- Created X PID_Controller data cluster
- Y PID_Controller -- Created Y PID_Controller data cluster

HolDrvCtrl_PackProfPID



Convenience, single call, LabVIEW function. Packs the tuning parameters of the Theta Profiled PID controller used by the Holonomic Controller.

NOTE -- This version does NOT support dynamic tuning. The constants are read only used during the first execution cycle.

Inputs:

- PID Tuning -- PID Tuning parameters containing:
 - Kp -- The proportional coefficient.
 - Ki -- The integral coefficient.
 - Kd -- The derivative coefficient.
 - Maximum Integral -- The maximum value of the integrator.
 - Minimum Integral -- The minimum value of the integrator.
- PID Input Limits -- Cluster containing:
 - MaxInput -- (Default: 0)
 - MinInput -- (Default: 0)
 - Continous -- When True indicates that the input is continuous. (Default: False)
- Max_Speed -- Maximum speed robot rotation
- Max_Accel -- Maximum acceleration of robot rotation.
- Period -- Period of repeated calls in seconds. (Default: 0.020)

Outputs:

- Theta Profiled_PID_Controller -- Created Theta Profiled PID_Controller data cluster

HolDrvCtrl_SetEnabled



Enables and disables the controller for troubleshooting problems. When calculate() is called on a disabled controller, only feedforward values are returned.

Input:

- inHolonomicDrvCtrl -- controller data cluster
- enabled -- If the controller is enabled or not.

Outputs:

- outHolonomicDrvCtrl -- updated controller data cluster
-

HolDrvCtrl_SetTolerance



Sets the pose error which is considered tolerance for use with AtReference function.

Inputs:

- inHolonomicDrvCtrl -- controller data cluster
- tolerance -- The pose error which is tolerable.

Outputs:

- outHolonomicDrvCtrl -- updated controller data cluster

ImplModelFollow

ImplModelFollow_Calculate



Returns the next output of the controller.

Inputs:

- in Implicit Model Follower -- Input data cluster
- x -- The current state x.
- u -- The current input for the original model.

Outputs:

- out Implicit Model Follower -- Updated data cluster
- U Out -- The next controller output.
- Error -- Returns TRUE if an error occurred.

ImplModelFollow_GetU



Returns the control input vector u.

Inputs:

- Implicit Model Follower -- Input data cluster

Outputs:

- U -- The control input.

ImplModelFollow_GetU_Single



Returns an element of the control input vector u.

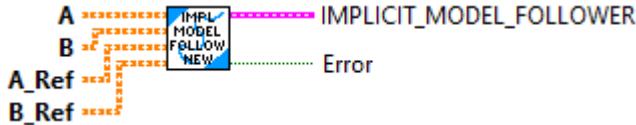
Inputs:

- Implicit Model Follower -- Input data cluster

Outputs:

- Index -- Row of u.
- U -- The control input value.

ImplModelFollow_New



Implicit model following lets us design a feedback controller that erases the dynamics of our system and makes it behave like some other system. This can be used to make a drivetrain more controllable during teleop driving by making it behave like a slower or more benign drivetrain.

Create implicit model follower

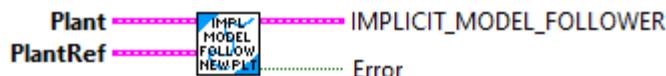
Inputs:

- A < states, states > -- Continuous system matrix of the plant being controlled.
- B < states, inputs > -- Continuous input matrix of the plant being controlled.
- A_Ref < states, states > -- Continuous system matrix whose dynamics should be followed.
- B_Ref < states, inputs > -- Continuous input matrix x whose dynamics should be followed.

Outputs:

- Implicit Model Follower -- Created data cluster
 - Error -- If TRUE, an error occurred.
-

ImplModelFollow_New_Plant



Implicit model following lets us design a feedback controller that erases the dynamics of our system and makes it behave like some other system. This can be used to make a drivetrain more controllable during teleop driving by making it behave like a slower or more benign drivetrain.

Create implicit model follower

Inputs:

- Plant -- The plant being controlled.
- PlantRef -- The plant whose dynamics should be followed.

Outputs:

- Implicit Model Follower -- Created data cluster
 - Error -- If TRUE, an error occurred.
-

ImplModelFollow_Reset



Resets the controller.

Inputs:

- in Implicit Model Follower -- Input data cluster

Outputs:

- out Implicit Model Follower -- Updated data cluster

JerkConstraint

JerkConstraint_New



Constructs a centripetal acceleration constraint.

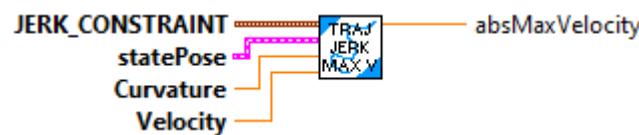
Parameters:

- MaxCentripitalAccel - Maximum Centripetal acceleration (meters/sec²)

Returns

- CentripetalAccelConstraint - Constraint data structure.

JerkConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

Calculation comments.

```

// ac = v^2 / r
// k (curvature) = 1 / r
// therefore, ac = v^2 * k
// ac / k = v^2
// v = std::sqrt(ac / k)
  
```

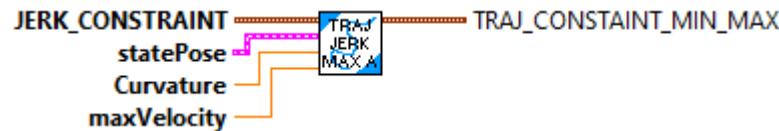
Parameters:

- CentripitalAccelConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

JerkConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values. Note says. The acceleration of the robot has no impact on the centripetal acceleration of the robot.

Parameters:

- CentripitalAccelConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

KalmanFilter

KalmanFilter_Correct



Correct the state estimate \hat{x} using the measurements in y .

Inputs:

- inKalmanFIltter -- filter data cluster
- u -- Same control input used in the last predict step.
- y -- Measurement vector.

Outputs:

- outKalmanFilter -- updated filter data cluster
- error -- If true, an error occurred

KalmanFilter_GetK



Returns the steady-state Kalman gain matrix K.

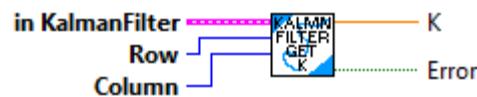
Inputs:

- KalmanFilter -- filter data cluster

Outputs:

- K - The steady-state Kalman gain matrix K.

KalmanFilter_GetK_Single



Returns an element of the steady-state Kalman gain matrix K.

Inputs:

- KalmanFilter -- filter data cluster
- row -- matrix row index (0-n)
- column -- matrix column index (0-n)

Outputs:

- K - The steady-state Kalman gain matrix K.
 - Error -- value is TRUE if an error occurred.
-

KalmanFilter_GetXHat



Returns the state estimate x-hat.

Inputs:

- KalmanFilter -- filter data cluster

Outputs:

- xHat - The state estimate x-hat.
-

KalmanFilter_GetXHat_Single



Returns a single element of the state estimate x-hat.

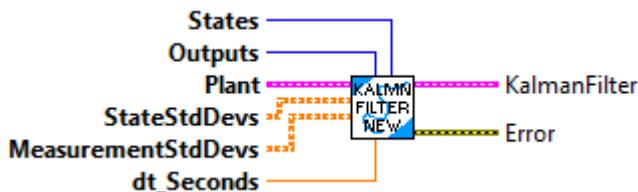
Inputs:

- KalmanFilter -- filter data cluster
- row -- matrix row number (0-n)

Outputs:

- xHat - The single state estimate x-hat value.

KalmanFilter_New



A Kalman filter combines predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

Kalman filters use a K gain matrix to determine whether to trust the model or measurements more. Kalman filter theory uses statistics to compute an optimal K gain which minimizes the sum of squares error in the state estimate. This K gain is used to correct the state estimate by some amount of the difference between the actual measurements and the measurements predicted by the model.

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf> chapter 9 "Stochastic control theory".

Constructs a state-space observer with the given plant.

Inputs:

- states -- A Nat representing the states of the system.
- outputs -- A Nat representing the outputs of the system.
- plant -- The plant used for the prediction step.
- stateStdDevs -- Standard deviations of model states.
- measurementStdDevs -- Standard deviations of measurements.
- dtSeconds -- Nominal discretization timestep.

Outputs:

- KalmanFilter -- The kalman filter data cluster
- Error -- TRUE indicates an error has occurred.

KalmanFilter_Predict



Project the model into the future with a new control input u.

Inputs:

- inKalmanFilter -- filter data cluster
- u -- New control input from controller.
- dtSeconds -- Timestep for prediction.

Outputs:

- outKalmanFilter -- updated filter data cluster
- error -- If value is TRUE, an error occurred.

KalmanFilter_Reset



Resets xHAT matrix to all zero

Inputs:

- inKalmanFilter -- filter data cluster

Outputs:

- outKalmanFilter -- filter data cluster

KalmanFilter_SetXHat



Set initial state estimate x-hat.

Inputs:

- inKalmanFilter -- filter data cluster
- xhat -- The state estimate x-hat.

Outputs:

- outKalmanFilter -- updated filter data cluster
- sizeCoerced -- If TRUE, an unexpected error occurred with the size of the input xHAT

KalmanFilter_SetXHat_Single



Sets a single row of the initial state estimate x-hat.

Inputs:

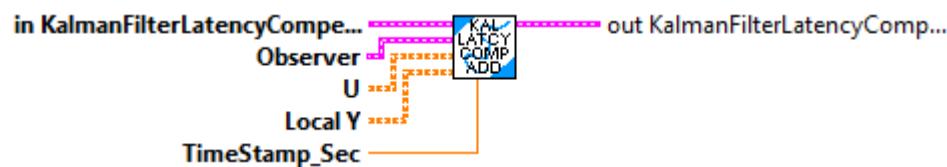
- inKalmanFilter -- filter data cluster
- xhat -- The state estimate x-hat.
- row -- matrix row number (0-n) to set.

Outputs:

- outKalmanFilter -- updated filter data cluster
- error -- If TRUE, an unexpected error occurred

KalmanFilterLatencyComp

KalmanFilterLatencyComp_AddObserverState



Add past observer states to the observer snapshots list.

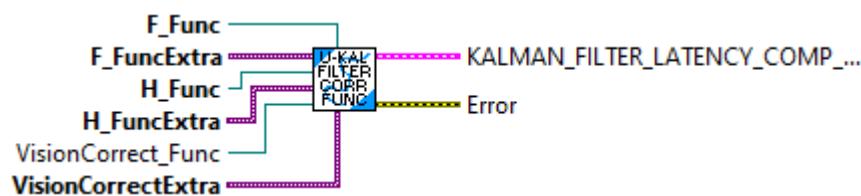
Inputs:

- KalmanLatencyComp -- Data cluster
- observer -- The observer. (Uncented Kalman Filter)
- u -- The input matrix at the timestamp.
- localY -- The local output matrix at the timestamp
- timestampSeconds -- The timesnap of the state.

Outputs:

- outKalmanLatencyComp -- updated data cluster

KalmanFilterLatencyComp_ApplyPastGlobalMeas_FuncGroup



Packs the individual function references and extra data into a cluster to pass to ApplyPastGlobalMeasurements.

Inputs:

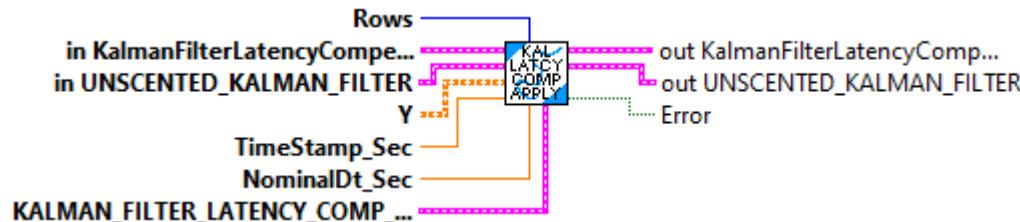
- F_Func --

- F_FuncExtra -- variant containing the extra data, if any, required by the F_Func
- H_Func --
- H_FuncExtra -- variant containing the extra data, if any, required by the F_Func
- VisionCorrect_Func --
- VisionCorrectExtra -- variant containing the extra data, if any, required by the VIisionCorrect_Func

Outputs:

- Comp_Func_Group -- Packed data cluster
- Error -- If TRUE, an error occurred.

KalmanFilterLatencyComp_ApplyPastGlobalMeasurement_UKF



Add past global measurements (such as from vision) to the estimator. This routine is particular for a system that uses an Unscented Kalman Filter.

Inputs:

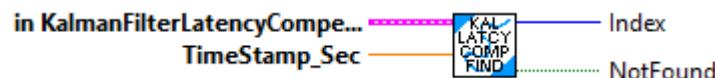
- rows -- The rows in the global measurement vector.
- KalmanLatencyComp -- Data cluster
- Unscented Kalman Filter -- Data cluster for the unscented kalman filter (observer). This is used to recalculate the estimated position from the newly applied measurements.
- y -- The measurement.
- timestampSeconds -- The timestamp of the measurement.
- nominalDtSeconds -- The nominal timestep.

- KALMAN_FILTER_LATENCY_COMP_FUNC_GROUP -- The function references that take calls correct() on the observer.

Outputs:

- outKalmanLatencyComp -- updated data cluster
 - outUnscented Kalman Filter -- updated cluster for the unscented kalman filter (observer).
 - Error -- Set to TRUE if an error occurred.
-

KalmanFilterLatencyComp_FindClosestMeasurement



An internal function that finds the item in the Past Observer Snapshot list with the timestamp closest to the one passed as an input.

Inputs:

- inKalmanFilterLatencyCompensator -- Data cluster
- TimeStamp_Sec -- Relative robot timestamp of the item to find (Seconds)

Outputs:

- Index -- Integer index to the closest entry
 - NotFound -- Boolean, whose value is TRUE if the closest entry was not found.
-

KalmanFilterLatencyComp_New



Creates a new, empty, Kalman Filter Latency Compensator data cluster. The Observer Snapshot list is created empty.

Inputs:

Outputs:

- KalmanFilterLatencyCompensator -- Newly created data cluster.

KalmanFilterLatencyComp_Observer_New



Creates the Observer List Item data cluster from individual input parameters.

Inputs:

- observer -- The observer. (Uncentred Kalman Filter) data cluster
- u -- The input matrix at the timestamp.
- localY -- The local output matrix at the timestamp
- timestampSeconds -- The timesnap of the state.

Outputs:

- ObserverListItem -- Observer List Item data cluster (ready to push onto the list of stored items)

KalmanFilterLatencyComp_Reset



Clears the observer snapshot buffer.

Inputs:

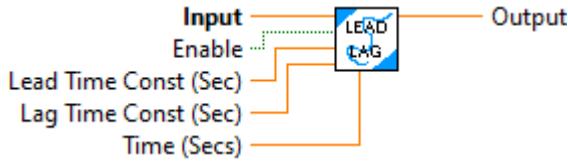
- KalmanLatencyComp -- Data cluster

Outputs:

- outKalmanLatencyComp -- Updated data cluster

LeadLag

LeadLag_Execute



This function provides a classical control lead / lag calculation (or filter). This can be used as a lag only, lead only, or lead / lag function.

Inputs:

- Input value -- double float -- input value
- Enable -- boolean -- When true the lead / lag calculation is performed. When false, the output = the input. (optional. Default: TRUE).
- Lead Time Const -- double float -- Lead time constant (seconds). If the value is zero (or missig), lead fuction function is not performed. (optional. Default: 0.0)
- Lag Time Const -- double float -- Lag time constant (seconds). If the value is zero (or missig), lag fuction function is not performed. (optional. Default: 0.0)
- Time -- double float -- Elapsed time (seconds). (Optional, Default: read FPGA time)

OutputsL

- Output -- double float -- The result of the leag / lag calculation. If Enable is False, the output will track (equal) the input.

LinearFilter

LinearFilter_BackwardFiniteDifference



Creates a backward finite difference filter that computes the nth derivative of the input given the specified number of samples.

For example, a first derivative filter that uses two samples and a sample period of 20 ms would be

`LinearFilter_BackwardFiniteDifference(`

```

1,
2,
0.02);
  
```

For additional information see:

https://en.wikipedia.org/wiki/Finite_difference_coefficient#Arbitrary_stencil_points

For a given list of stencil points s of length n and the order of derivative $d < n$, the finite difference coefficients can be obtained by solving the following linear system for the vector a .

$$\begin{aligned} [s_1^{\circ} \ ? \ s_n^{\circ}] [a_1] &= [d_0, d_1, \dots, d_d] \\ [? \ ? \ ?] [a_1] &= d! [? \ ? \ ?] \\ [s_1 n_1^{\circ} \ ? \ s_n n_n^{\circ}] [a_1] &= [d_0, d_1, \dots, d_d] \end{aligned}$$

where d_i are the Kronecker delta. The FIR gains are the elements of the vector a in reverse order divided by n_i .

The order of accuracy of the approximation is of the form $O(h^n)$.

Inputs:

- derivative -- The order of the derivative to compute.
- samples -- The number of samples to use to compute the given derivative. This must be one more than the order of derivative or higher.
- period -- The period in seconds between samples taken by the user.

Outputs:

- LINEAR FILTER - cluster containing:
 - InputGains - Array of feedforward or FIR gain factors (bx)
 - OutputGains - Array of feedback or IIR gain factors for feedback terms (ax)
 - Inputs - Array of the last n saved inputs
 - Outputs - Array of the last n saved outputs
 - InGainCount - Number of input gain terms
 - OutGainCount - number of output gain terms
- Error -- Returns TRUE if an error occurred.

LinearFilter_Calculate

Calculates the next value of the filter. To work correctly this must be called periodically

Inputs:

- inLinearfilter - Linear filter data structure
- input - Current input value.

Outputs:

- outLinearfilter - Updated linear filter data structure

- output - The filtered value at this step

LinearFilter_CutoffFrequency



Calculate the cutoff frequency from the time constant.

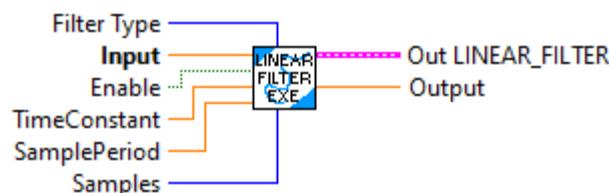
Inputs:

- time Constant - seconds

Outputs:

- cutoff Frequency - hertz

LinearFilter_Execute



This is a LabVIEW "convience" function that combines the various subVI for a linear filter into a single subVI. This subVI creates and executes the linear filter. The filter can be dynamically enabled and disabled. When the filter is disabled, the input is transferred directly to the output. Bumpless transfer is provided when enabling the filter. Bumpless transfer is NOT provided when disabling the filter. The output value jumps directly to the current input value. The filter type and filter constants can also be changed dynamically. When the filter constants are changed, the output may jump to the current input value when the internal buffers are reset to the current input value.

Inputs:

- FilterType - An enumerated value used to select the type of filter to be configured. The possible values are:

- Moving Average - Specify the "sample" input.

- Low pass 1st order Butterworth filter
 - Low pass 2nd order Butterworth filter
 - High pass 1st order Butterworth filter
 - High pass 2nd order Butterworth filter
- Input - Input value to be filtered (Required)
- Enable - Boolean indicating filtering should be performed (Default = True)
- TimeConstant - Filtering time constant seconds. Used for all types except Moving Average (Default = 0.020)
- SamplePeriod - Execution period seconds. Used for all types except Moving Average. (Default = 0.020)
- Samples - Number of samples for the moving average filter. (Default = 3)

Outputs:

- OutLinearFilter - Current value of the LinearFilter data structure.
- Output - Current output value.

LinearFilter_Factorial



Computes the factorial of "n". This is an internal function.

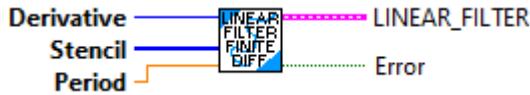
Inputs:

- n - Integer value ≥ 1 .

Outputs:

- factorial - The calculated factorial value, as an integer.
-

LinearFilter_FiniteDifference



Creates a finite difference filter that computes the nth derivative of the input given the specified stencil points.

Stencil points are the indices of the samples to use in the finite difference. 0 is the current sample, -1 is the previous sample, -2 is the sample before that, etc. Don't use positive stencil points (samples from the future) if the LinearFilter will be used for stream-based online filtering.

For additional information see:

https://en.wikipedia.org/wiki/Finite_difference_coefficient#Arbitrary_stencil_points

For a given list of stencil points s of length n and the order of derivative $d < n$, the finite difference coefficients can be obtained by solving the following linear system for the vector a .

$$[s_1 \ ? \ s_n][a_1] = [d_0, d]$$

$$[? \ ? \ ?][?] = d! [?]$$

$$[s_1 n^{-1} \ ? \ s_n n^{-1}][a?] = [d_1, d]$$

where d_1, d are the Kronecker delta. The FIR gains are the elements of the vector a in reverse order divided by h .

The order of accuracy of the approximation is of the form $O(h^n)$.

Inputs:

- derivative -- The order of the derivative to compute.

- stencil -- List of stencil points. Its length is the number of samples to use to compute the given derivative. This must be one more than the order of the derivative or higher.
- period -- The period in seconds between samples taken by the user.

Outputs:

- LINEAR FILTER - cluster containing:
 - InputGains - Array of feedforward or FIR gain factors (bx)
 - OutputGains - Array of feedback or IIR gain factors for feedback terms (ax)
 - Inputs - Array of the last n saved inputs
 - Outputs - Array of the last n saved outputs
 - InGainCount - Number of input gain terms
 - OutGainCount - number of output gain terms
- Error -- Returns TRUE if an error occurred.

LinearFilter_HighPass



Creates a first-order high-pass filter of the form:

$$y[n] = \text{gain} \cdot x[n] + (-\text{gain}) \cdot x[n-1] + \text{gain} \cdot y[n-1]$$

where

$$\text{gain} = e^{(-dt / T)},$$

T is the time constant in seconds.

This filter is stable for time constants greater than zero.

Inputs:

- timeConstant The discrete-time time constant in seconds.
- samplePeriod The period in seconds between samples taken by the user.

Outputs:

- Linearfilter - Created datat structure
-

LinearFilter_HighPassBW1

Creates a 1st order Butterworth high pass filter.

More information on this type of filter can be found at:

https://en.wikipedia.org/wiki/Butterworth_filter

Inputs:

- timeConstant - The discrete-time time constant in seconds.
- samplePeriod - The period in seconds between samples taken by the user.

Outputs:

- Linearfilter - Created datat structure
-

LinearFilter_HighPassBW2

Creates a 2nd order Butterworth high pass filter.

A 2nd order filter provides more filtering, however it also has a larger phase shift (time delay).

More information on this type of filter can be found at:

https://en.wikipedia.org/wiki/Butterworth_filter

Inputs:

- timeConstant - The discrete-time time constant in seconds.
- samplePeriod - The period in seconds between samples taken by the user.

Outputs:

- Linearfilter - Created datat structure

LinearFilter_LowPassBW1



Creates a 1st order Butterworth low pass filter.

More information on this type of filter can be found at:

https://en.wikipedia.org/wiki/Butterworth_filter

Inputs:

- timeConstant - The discrete-time time constant in seconds.
- samplePeriod - The period in seconds between samples taken by the user.

Outputs:

- LinearFilter - Created datat structure

LinearFilter_LowPassBW2



Creates a 2nd order Butterworth low pass filter.

A 2nd order filter provides more filtering, however it also has a larger phase shift (time delay).

More information on this type of filter can be found at:

https://en.wikipedia.org/wiki/Butterworth_filter

Inputs:

- timeConstant - The discrete-time time constant in seconds.
- samplePeriod - The period in seconds between samples taken by the user.

Outputs:

- Linearfilter - Created datat structure

LinearFilter_MovingAverage



Creates a K-tap FIR moving average filter of the form:

$$y[n] = 1/k * (x[k] + x[k-1] + \dots + x[0]).$$

This filter is always stable.

Inputs:

- Samples - The number of samples to average over. Higher = smoother but slower.

The number of samples must be ≥ 1

Outputs:

- Linearfilter - Created datat structure
- Error -- If TRUE, an error occured.

LinearFilter_New



This VI creates a new data structure (cluster) that stores the data for a set of VIs that implement a linear, digital filter. All types of FIR and IIR filters are supported. A set of VIs are provided to create commonly used types of filters.

Filters are of the form:

$$y[n] = (b_0*x[n] + b_1*x[n-1] + \dots + b_P*x[n-P]) - (a_0*y[n-1] + a_2*y[n-2] + \dots + a_Q*y[n-Q])$$

Where:

$y[n]$ is the output at time "n"

$x[n]$ is the input at time "n"

$y[n-1]$ is the output from the LAST time step ("n-1")

$x[n-1]$ is the input from the LAST time step ("n-1")

$b_0 \dots b_P$ are the "feedforward" (FIR) gains

$a_0 \dots a_Q$ are the "feedback" (IIR) gains

IMPORTANT! Note the "-" sign in front of the feedback term! This is a common convention in signal processing.

What can linear filters do? Basically, they can filter, or diminish, the effects of undesirable input frequencies. High frequencies, or rapid changes, can be indicative of sensor noise or be otherwise undesirable. A "low

"low pass" filter smooths out the signal, reducing the impact of these high frequency components. Likewise, a "high pass" filter gets rid of slow-moving signal components, letting you detect large changes more easily.

Example FRC applications of filters:

- Getting rid of noise from an analog sensor input (note: the roboRIO's FPGA can do this faster in hardware)
- Smoothing out joystick input to prevent the wheels from slipping or the robot from tipping
- Smoothing motor commands so that unnecessary strain isn't put on electrical or mechanical components
- If you use clever gains, you can make a PID controller out of this class! (Use the PID set of VI's instead...)

For more on filters, we highly recommend the following articles:

https://en.wikipedia.org/wiki/Linear_filter

https://en.wikipedia.org/wiki/Iir_filter

https://en.wikipedia.org/wiki/Fir_filter

Note 1: calculate() should be called by the user on a known, regular period. You can use code in a periodic function.

Note 2: For ALL filters, gains are necessarily a function of frequency. If you make a filter that works well for you at, say, 100Hz, (executing every 10 milliseconds), you will most definitely need to adjust the gains if you then want to run it at 200Hz, (executing every 5 milliseconds)! Combining this with Note 1, the impetus is on YOU as a developer to make sure calculate() gets called at the desired, constant frequency!

Inputs:

- InputGains - Array of feedforward or FIR gain factors (bx)
- OutputGains - Array of feedback or IIR gain factors for feedback terms (ax)

Outputs:

- LINEAR FILTER - cluster containing:
 - InputGains - Array of feedforward or FIR gain factors (bx)
 - OutputGains - Array of feedback or IIR gain factors for feedback terms (ax)

- Inputs - Array of the last n saved inputs
 - Outputs - Array of the last n saved outputs
 - InGainCount - Number of input gain terms
 - OutGainCount - number of output gain terms
-
-

LinearFilter_Reset



Reset the filter state. Sets the saved inputs and outputs to zero.

Inputs:

- inLinearfilter - Linear filter data structure

Outputs:

- outLinearfilter - Updated linear filter data structure
-
-

LinearFilter_ResetToValue



Reset the filter state. Sets the saved inputs and outputs to "InputValue:"

Inputs:

- inLinearfilter - Linear filter data structure
- InputValue - This value is used to fill the saved input and output values

Outputs:

- outLinearfilter - Updated linear filter data structure

LinearFilter_SinglePoleIIR



Creates a one-pole IIR low-pass filter of the form:

$$y[n] = (1-\text{gain})x[n] + \text{gain}y[n-1]$$

where

$$\text{gain} = e^{(-dt / T)},$$

T is the time constant in seconds.

This filter is stable for time constants greater than zero.

Inputs:

- timeConstant - The discrete-time time constant in seconds.
- samplePeriod - The period in seconds between samples taken by the user.

Outputs:

- Linearfilter - Created datat structure
- Cutoff Frequency - Frequency (Hz)

LinearFilter_TimeConst



Calculate the time constant from the cutoff frequency

Inputs:

- cutoff Frequency - hertz

Outputs:

- time Constant - seconds

LinearPlntInvFF

LinearPlntInvFF_Calculate



Calculate the feedforward with current and future reference vectors.

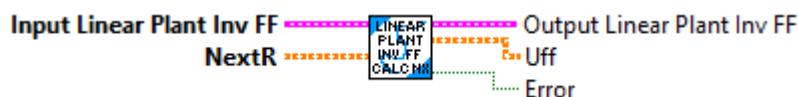
Inputs:

- LinearPlntInversionFF -- system data cluster
- r -- The reference state of the current timestep (k).
- nextR -- The reference state of the future timestep (k + dt).

Outputs:

- outLinearPlntInversionFF -- updated system data cluster
- Uff -- The calculated feedforward.
- error -- If TRUE, an error occurred.

LinearPlntInvFF_Calculate_NextR



Calculate the feedforward with only the desired future reference. This uses the internally stored "current" reference.

If this method is used the initial state of the system is the one set using {@link LinearPlantInversionFeedforward#reset(Matrix)}. If the initial state is not set it defaults to a zero vector.

Inputs:

- LinearPlntInversionFF -- system data cluster
- nextR -- The reference state of the future timestep ($k + dt$).

Outputs:

- outLinearPlntInversionFF -- updated system data cluster
- Uff -- The calculated feedforward.
- error -- If TRUE, an error occurred.

LinearPlntInvFF_GetR



Returns the current reference vector r.

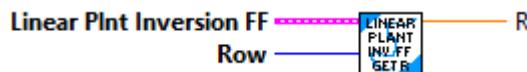
Inputs:

- LinearPlntInversionFF -- system data cluster

Outputs:

- R -- The current reference vector.

LinearPlntInvFF_GetR_Single



Returns an element of the current reference vector r.

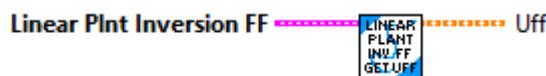
Inputs:

- LinearPlntInversionFF -- system data cluster
- row -- Row of r.

Outputs:

- R -- The row of the current reference vector.

LinearPlntInvFF_GetUff



Returns the previously calculated feedforward as an input vector.

Inputs:

- LinearPlntInversionFF -- system data cluster

Outputs:

- Uff -- The calculated feedforward.

LinearPlntInvFF_GetUff_Single



Returns an element of the previously calculated feedforward.

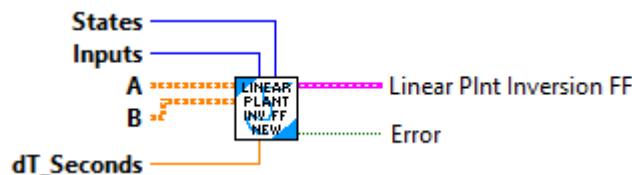
Inputs:

- LinearPlntInversionFF -- system data cluster
- row -- Row of uff.

Outputs:

- Uff -- The row of the calculated feedforward.

LinearPlntInvFF_New



Constructs a plant inversion model-based feedforward from a LinearSystem.

The feedforward is calculated as

$$u_{ff} = Binv(r_{k+1} - Ar_k),$$

where

$Binv$

is the pseudoinverse of B .

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a feedforward with the given coefficients.

Inputs:

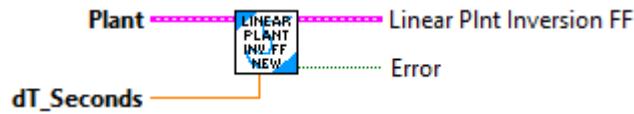
- A -- Continuous system matrix of the plant being controlled.
- B -- Continuous input matrix of the plant being controlled.
- dtSeconds -- Discretization timestep.

Outputs:

- outLinearPlntInversionFF -- updated system data cluster

- error -- If TRUE, an error occurred.

LinearPlntInvFF_New_Plant



Constructs a plant inversion model-based feedforward from a LinearSystem.

The feedforward is calculated as

$$u_{ff} = \text{Binv} (r_{k+1} - A r_k),$$

where

Binv

is the pseudoinverse of B .

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a feedforward with the given plant.

Inputs:

- plant -- The plant being controlled.
- dtSeconds -- Discretization timestep.

Outputs:

- outLinearPlntInversionFF -- updated system data cluster
- error -- If TRUE, an error occurred.

LinearPlntInvFF_Reset_Initial



Resets the feedforward with a specified initial state vector.

Inputs:

- LinearPlntInversionFF -- system data cluster
- initialState -- The initial state vector.

Outputs:

- outLinearPlntInversionFF -- updated system data cluster
- sizeCoerced -- If TRUE, an error occurred.

LinearPlntInvFF_Reset_Zero



Resets the feedforward with a zero initial state vector.

Inputs:

- LinearPlntInversionFF -- system data cluster

Outputs:

- outLinearPlntInversionFF -- updated system data cluster

LinearQuadraticRegulator

LinearQuadraticRegulator_Calculate



Returns the next output of the controller.

Inputs:

- LinearQuadraticRegulator -- controller data cluster
- x -- The current state x.

Output:

- outLinearQuadraticRegulator -- updated controller data cluster
- U -- The next controller output.
- error -- If TRUE, an error occurred.

LinearQuadraticRegulator_Calculate_NextR



Returns the next output of the controller.

Inputs:

- LinearQuadraticRegulator -- controller data cluster
- x -- The current state x.
- nextR -- the next reference vector r.

Output:

- outLinearQuadraticRegulator -- updated controller data cluster
 - U -- The next controller output.
 - error -- If TRUE, an error occurred.
-

LinearQuadraticRegulator_GetK



Returns the controller matrix K.

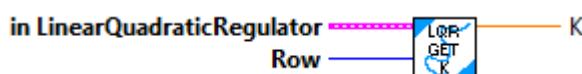
Input:

- LinearQuadraticRegulator -- controller data cluster

Outputs:

- K -- the controller matrix K.
-

LinearQuadraticRegulator_GetK_Single



Returns a single element of the controller matrix K.

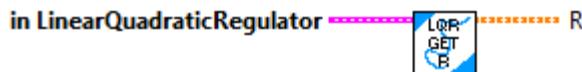
Input:

- LinearQuadraticRegulator -- controller data cluster
- row -- index into the row of K (0-n)

Outputs:

- K -- the controller single value K.

LinearQuadraticRegulator_GetR



Returns the reference vector r.

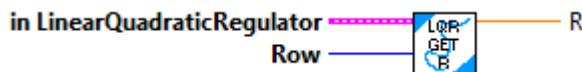
Inputs:

- LinearQuadraticRegulator -- controller data cluster

Outputs:

- R -- The reference vector.

LinearQuadraticRegulator_GetR_Single



Returns a single element of the reference vector r.

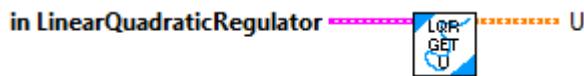
Inputs:

- LinearQuadraticRegulator -- controller data cluster
- row -- index into the R matrix (0-n)

Outputs:

- R -- The single requested reference value.

LinearQuadraticRegulator_GetU



Returns the control input vector u.

Inputs:

- LinearQuadraticRegulator -- controller data cluster

Outputs:

- U -- The control input.

LinearQuadraticRegulator_GetU_Single



Returns a single element of the control input vector u.

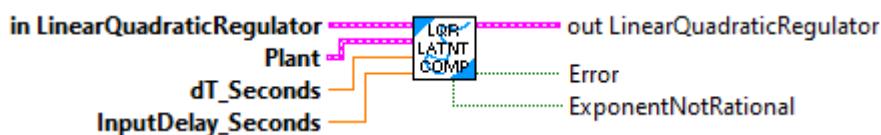
Inputs:

- LinearQuadraticRegulator -- controller data cluster
- row -- Index of U to be returned (0-n)

Outputs:

- U -- The control input single value.

LinearQuadraticRegulator_LatencyCompensate



Adjusts LQR controller gain to compensate for a pure time delay in the input.

Linear-Quadratic regulator controller gains tend to be aggressive. If sensor measurements are time-delayed too long, the LQR may be unstable. However, if we know the amount of delay, we can compute the control based on where the system will be after the time delay.

See <https://file.tavsys.net/control/controls-engineering-in-frc.pdf> appendix C.4 for a derivation.

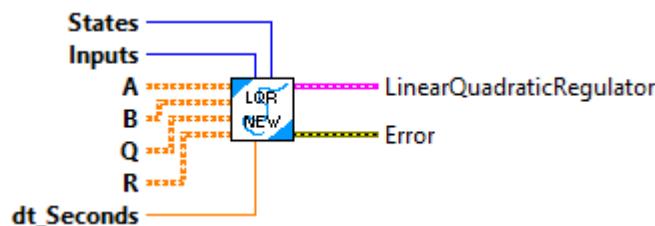
Inputs:

- inLinearQuadraticRegulator -- controller data cluster
- plant -- The plant being controlled.
- dtSeconds -- Discretization timestep in seconds.
- inputDelaySeconds -- Input time delay in seconds.

Output:

- outLinearQuadraticRegulator -- updated data cluster
- error -- If TRUE, an error occurred.
- exponentNotRational -- if TRUE, an error occurred.

LinearQuadraticRegulator_New



Contains the controller coefficients and logic for a linear-quadratic regulator (LQR). LQRs use the control law $u = K(r - x)$.

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a controller with the given coefficients and plant.

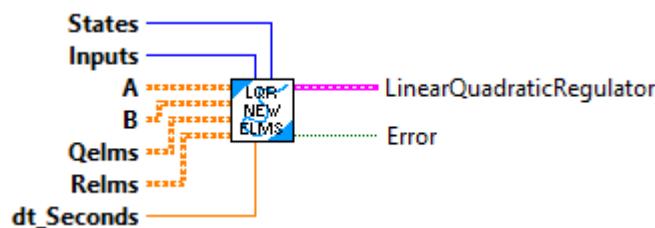
Inputs:

- A -- Continuous system matrix of the plant being controlled.
- B -- Continuous input matrix of the plant being controlled.
- Q -- The state cost matrix.
- R -- The input cost matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- LinearQuadraticRegulator -- controller data cluster
- Error -- if TRUE, and error occurred

LinearQuadraticRegulator_New_ELMS



Contains the controller coefficients and logic for a linear-quadratic regulator (LQR). LQRs use the control law $u = K(r - x)$.

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a controller with the given coefficients and plant.

Inputs:

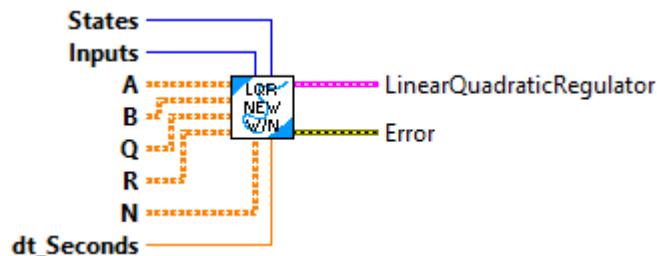
- A -- Continuous system matrix of the plant being controlled.
- B -- Continuous input matrix of the plant being controlled.
- qelms -- The maximum desired error tolerance for each state.

- relms -- The maximum desired control effort for each input.
- dtSeconds -- Discretization timestep.

Outputs:

- LinearQuadraticRegulator -- controller data cluster
- Error -- If TRUE, an error occurred.

LinearQuadraticRegulator_New_N



Contains the controller coefficients and logic for a linear-quadratic regulator (LQR). LQRs use the control law $u = K(r - x)$.

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a controller with the given coefficients and plant.

Inputs:

- A -- Continuous system matrix of the plant being controlled.
- B -- Continuous input matrix of the plant being controlled.
- Q -- The state cost matrix.
- R -- The input cost matrix.
- N -- The state-input cross-term cost matrix.
- dtSeconds -- Discretization timestep.

Outputs:

- LinearQuadraticRegulator -- controller data cluster
 - Error -- if TRUE, and error occurred
-

LinearQuadraticRegulator_New_SystemELMS



Contains the controller coefficients and logic for a linear-quadratic regulator (LQR). LQRs use the control law $u = K(r - x)$.

For more on the underlying math, read

<https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a controller with the given coefficients and plant. Rho is defaulted to 1.

Inputs:

- LinearSystem - The plant being controlled.
- qelms -- The maximum desired error tolerance for each state.
- relms -- The maximum desired control effort for each input.
- dtSeconds -- Discretization timestep.

Outputs:

- LinearQuadraticRegulator -- controller data cluster
 - Error -- If TRUE, and error occurred.
-

LinearQuadraticRegulator_Reset



Resets the controller.

Inputs:

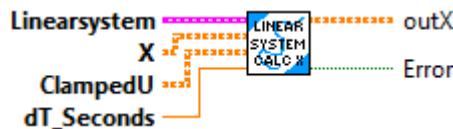
-- LinearQuadraticRegulator -- controller data cluster

Outputs:

-- outLinearQuadraticRegulator -- updated controller data cluster

LinearSystem

LinearSystem_CalculateX



Computes the new x given the old x and the control input.

This is used by state observers directly to run updates based on state estimate.

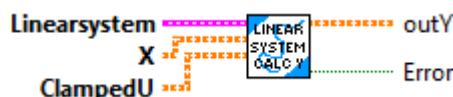
Inputs:

- LinearSystem -- system data cluster
- x -- The current state.
- clampedU -- The control input.
- dtSeconds -- Timestep for model update.

Outputs:

- outX -- the updated x.
- error -- If TRUE, an error occurred

LinearSystem_CalculateY



Computes the new y given the control input.

This is used by state observers directly to run updates based on state estimate.

Inputs:

- LinearSystem -- system data cluster
- x -- The current state.
- clampedU -- The control input.

Outputs:

- outY -- the updated output matrix Y.
 - error -- If TRUE, an error occurred.
-
-

LinearSystem_GetA



Returns the system matrix A.

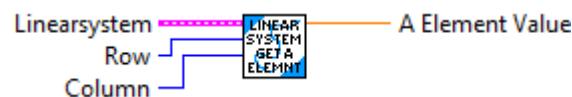
Inputs:

- LinearSystem -- system data cluster

Outputs:

- A -- the system matrix A.
-
-

LinearSystem_GetAElement



Returns an element of the system matrix A.

Inputs:

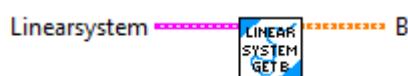
- LinearSystem -- system data cluster

- row -- Row of A.
- col -- Column of A.

Outputs:

- A_element_Value -- the system matrix A at (i, j).
-
-

LinearSystem_GetB



Returns the input matrix B.

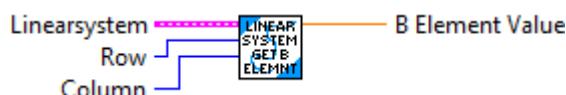
Inputs:

- LinearSystem -- system data cluster

Outputs:

- B -- the input matrix B.
-
-

LinearSystem_GetBElement



Returns an element of the input matrix B.

Inputs:

- LinearSystem -- system data cluster
- row -- Row of B.
- col -- Column of B.

Outputs:

- B_Value -- The value of the input matrix B at (i, j).
-

LinearSystem_GetC



Returns the output matrix C.

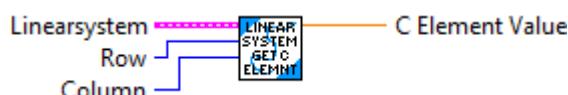
Inputs:

- LinearSystem -- system data cluster

Outputs:

- C -- Output matrix C.
-

LinearSystem_GetCElement



Returns an element of the output matrix C.

Inputs:

- LinearSystem -- system data cluster
- row -- Row of C.
- col -- Column of C.

Outputs:

- C_Value -- the double value of C at the given position.

LinearSystem_GetD



Returns the feedthrough matrix D.

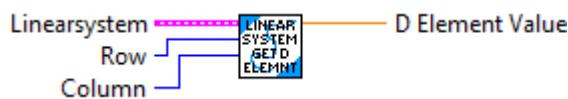
Inputs:

- LinearSystem -- system data cluster

Outputs:

- D -- the feedthrough matrix D.

LinearSystem_GetDElement



Returns an element of the feedthrough matrix D.

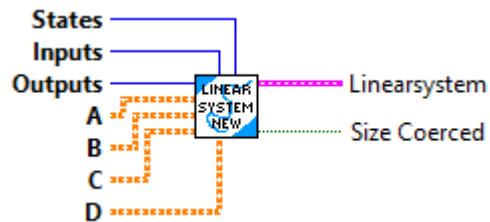
Inputs:

- LinearSystem -- system data cluster
- row -- Row of D.
- col -- Column of D.

Outputs:

- D_Value -- The feedthrough matrix D at (i, j).

LinearSystem_New



Construct a new LinearSystem from the four system matrices.

Inputs:

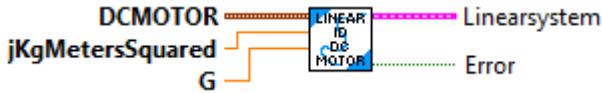
- states -- number of states
- inputs -- number of inputs
- outputs -- number of outputs
- a -- The system matrix A.
- b -- The input matrix B.
- c -- The output matrix C.
- d -- The feedthrough matrix D.

Outputs:

- outLinearSystem -- updated system data cluster
- error -- If TRUE, an error occurred.

LinearSystemId

LinearSystemId_CreateDCMotorSystem



Create a state-space model of a DC motor system. The states of the system are [angular position, angular velocity], inputs are [voltage], and outputs are [angular position, angular velocity].

The DC Motor linear system is defined as:

- states:
 - angular position (rad)
 - angular velocity (rad/sec)
- inputs:
 - motor voltage
- outputs:
 - angular position (rad)
 - angular velocity (rad/sec)

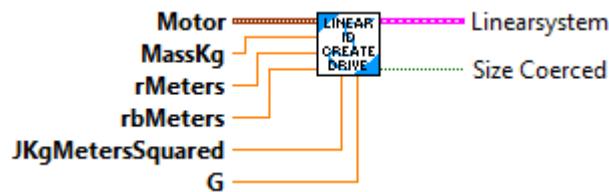
Inputs:

- motor -- The motor (or gearbox)
- jKgMetersSquared -- The moment of inertia J of the motor and attached load. (Must be > 0)
- G -- The reduction between motor and drum, as a ratio of output to input. (Must be > 0)

Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- Error -- A boolean indicating an error occurred creating the system. A value of TRUE indicates an error.

LinearSystemId_CreateDriveTrainVelocitySystem



Create a state-space model of a differential drive drivetrain. This linear system is defined as follows:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

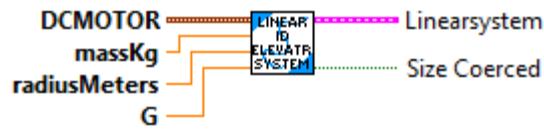
Inputs:

- motor -- the gearbox representing the motors driving the drivetrain.
- massKg -- the mass of the robot.
- rMeters -- the radius of the wheels in meters.
- rbMeters -- the radius of the base (half the track width) in meters.
- JKgMetersSquared -- the moment of inertia of the robot.
- G -- the gearing reduction as output over input.

Outputs:

- LinearSystem -- A LinearSystem cluster representing a differential drivetrain.
- SizeCoerced -- A boolean indicating that sizes had to be adjusted. A value of TRUE indicates an error.

LinearSystemId_CreateElevatorSystem



Create a state-space model of an elevator system. The states of the system are [position, velocity], inputs are [voltage], and outputs are [position].

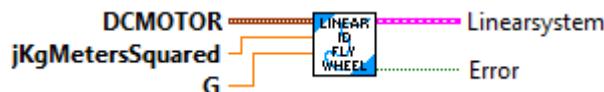
Inputs:

- motor -- The motor (or gearbox) attached to the arm.
- massKg -- The mass of the elevator carriage, in kilograms.
- radiusMeters -- The radius of the driving drum of the elevator, in meters.
- G -- The reduction between motor and drum, as a ratio of output to input.

Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- SizeCoerced -- A boolean indicating that sizes had to be adjusted. A value of TRUE indicates an error.

LinearSystemId_CreateFlywheelSystem



Create a state-space model of a flywheel system. The states of the system are [angular velocity], inputs are [voltage], and outputs are [angular velocity].

The flywheel linear system definition is:

- States
 - angular velocity (Rad/Sec)
- Inputs:

- motor voltage
- Outputs:
 - angular velocity (rad/sec)

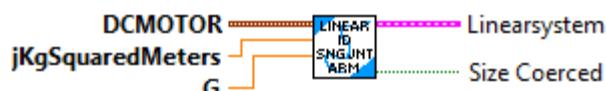
Inputs:

- motor -- The motor (or gearbox) attached to the arm.
- jKgMetersSquared -- The moment of inertia J of the flywheel.
- G -- The reduction between motor and drum, as a ratio of output to input.

Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- Error -- A boolean indicating an error occurred creating the system. A value of TRUE indicates an error.

LinearSystemId_CreateSingleJointedArmSystem



Create a state-space model of a single jointed arm system. The states of the system are [angle, angular velocity], inputs are [voltage], and outputs are [angle].

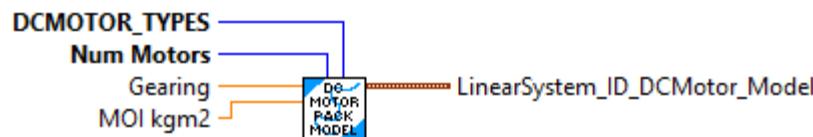
Inputs:

- motor -- The motor (or gearbox) attached to the arm.
- jKgSquaredMeters -- The moment of inertia J of the arm.
- G -- The gearing between the motor and arm, in output over input. Most of the time this will be greater than 1.

Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- Error -- A boolean indicating an error occurred creating the system. A value of TRUE indicates an error.

LinearSystemId_DCMotor_Pack_Model_Params



Pack the model parameters used to create a DC Motor Linear System.

Inputs

- DCMOTOR_TYPE -- enum -- Model of motor being used.
- Num Numtors -- integer -- Number of motors used.
- Gearing -- double -- Gear ratio. (Input speed / Output Speed)
- MOI -- double -- Moment of inertia - kg m²

Outputs:

- LinearSystemId DC Motor Model -- cluster -- packed model parameters.

LinearSystemId_DiffDrv_ID_Pack_Model_Params



Creates a Differential Drive Train linear system from the drive train characterization (identification) data.

The linear system is define as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs

- left voltage (volts)
- right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

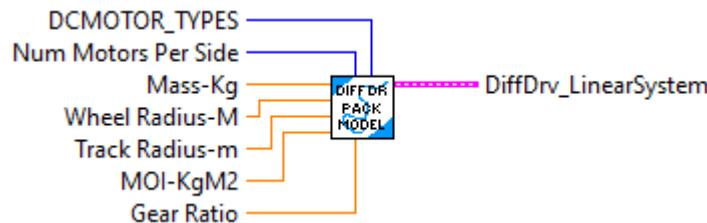
Inputs:

- Kv Linear -- double -- Linear velocity constant (volts/(meter/sec))
- Ka Linear -- double -- Linear acceleration constant (volts/(meter/sec²))
- Kv Angular -- double -- Rotational velocity constant (volts/(meter/sec))
- Ka Angular -- double -- Rotational acceleration constant (volts/(meter/sec²))

Outputs:

- DiffDrive Linear Syste -- Linear System -- The resulting differential drive linear system.

LinearSystemId_DiffDrv_Pack_Model_Params



Creates a Differential Drive Train linear system from the physical parameters of the robot..

The linear system is defined as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs

- left voltage (volts)
- right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

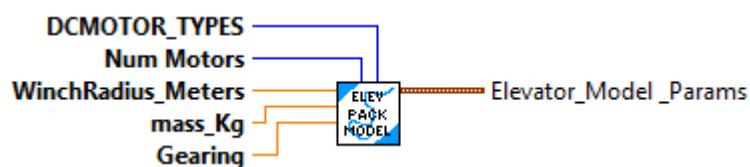
Inputs:

- DC Motor Type -- enum -- Model of the drive motor
- Num Motors Per Side -- integer -- Number of motoes on each side of the drive
- Mass -- double -- Robot mass (kg)
- Wheel Radius -- double -- Radius of drive wheel (meters) Note this is radius, not diameter.
- Track Radius -- double -- 1/2 of the distance between wheels (meters). This is effective radius which may be more than the physical radius.
- MOI -- double -- Moment of inertia (kg m^2)
- Gear Ratio -- double -- Drive gear ratio (Input/Output)

Output:

- DiffDrive Linear System -- Linear System -- Created differential drive linear system

LinearSystemId_Elevator_Pack_Model_Params



Pack the physical parameters describing an elevator linear system.

Inputs:

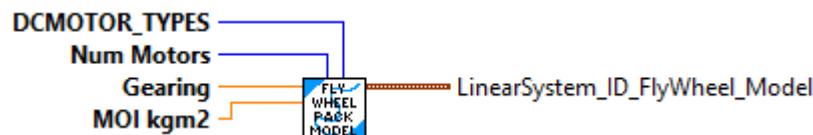
- DC Motor -- enum -- Model of the motor used for the flywheel

- Num Motors -- integer -- Number of flywheel drive motors
- Winch Radius -- double -- Radius of the winch (meters). Note this is radius, not diameter.
- Mass -- double -- Elevator mass (kg)
- Gearing -- double -- Gear ratio (Input/Output)

Outputs:

- Elevator Model Params -- cluster -- Packed elevator physical modeling parameters

LinearSystemId_FlyWheel_Pack_Model_Params



Pack the physical parameters describing a FlyWheel system.

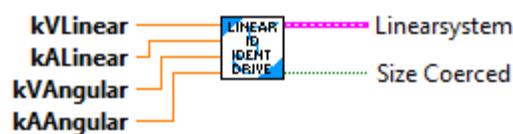
Inputs:

- DC Motor -- enum -- Model of the motor used for the flywheel
- Num Motors -- integer -- Number of flywheel drive motors
- Gearing -- double -- Gear ratio (Input/Output)
- MOI -- double -- Moment of inertia (kg m^2)

Outputs:

- FlyWheel Model Params -- cluster -- Packed flywheel physical modeling parameters

LinearSystemId_IdentifyDriveTrainSystem



Identify a standard differential drive drivetrain, given the drivetrain's kV and kA in both linear (volts/(meter/sec) and volts/(meter/sec²) and angular (volts/(meter/sec) and volts/(meter/sec²) cases. This can be found using frc-characterization. The linear system is defined as follows:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

Inputs:

- kVLinear -- The linear velocity gain, volts per (meter per second).
- kALinear -- The linear acceleration gain, volts per (meter per second squared).
- kVAngular -- The angular velocity gain, volts per (meter per second).
- kAAngular -- The angular acceleration gain, volts per (meter per second squared).

Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- SizeCoerced -- A boolean indicating an error occurred creating the system. A value of TRUE indicates an error.

Additional information

<https://github.com/wpilibsuite/frc-characterization>

LinearSystemId_IdentifyPositionSystem



Identify a position system from it's kV (volts/(unit/sec)) and kA (volts/(unit/sec²)). These constants cam be found using frc-characterization. The states of the system are [position, velocity]?, inputs are [voltage], and outputs are [position].

The distance unit you choose MUST be an SI unit (i.e. meters or radians). You can use the Util.Units subVI for converting between unit types.

Inputs:

- kV -- The velocity gain, in volts per (units per second)
- kA -- The acceleration gain, in volts per (units per second squared)

Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- SizeCoerced -- A value of TRUE indicates an unexpected error occured.

Additional information:

See [@SuppressWarnings\("ParameterName"\)](https://github.com/wpilibsuite/frc-characterization)

LinearSystemId_IdentifyVelocitySystem



Identify a velocity system from it's kV (volts/(unit/sec)) and kA (volts/(unit/sec²)). These constants cam be found using frc-characterization. The states of the system are [velocity], inputs are [voltage], and outputs are [velocity].

The distance unit you choose MUST be an SI unit (i.e. meters or radians). You can use the Util.Units subVI for converting between unit types.

Inputs:

- kV -- The velocity gain, in volts per (units per second)
- kA -- The acceleration gain, in volts per (units per second squared)

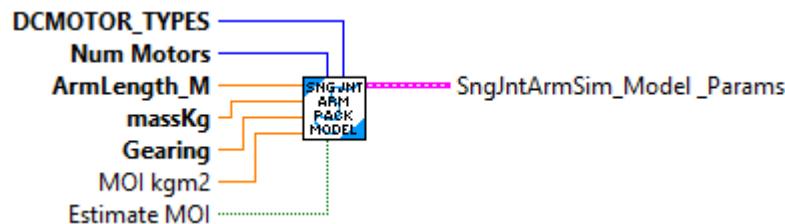
Outputs:

- LinearSystem -- A LinearSystem representing the given characterized constants.
- SizeCoerced -- A value of TRUE indicates an unexpected error occurred.

Additional information:

See <https://github.com/wpilibsuite/frc-characterization>

LinearSystemId_SngJntArm_Pack_Model_Params



Pack the physical parameters describing a single jointed arm linear system.

Inputs:

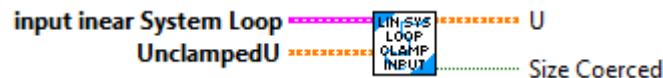
- DC Motor -- enum -- Model of the motor used for the single jointed arm
- Num Motors -- integer -- Number of drive motors
- Arm Length -- double -- Length of arm (meters).
- Mass -- double -- Elevator mass (kg)
- Gearing -- double -- Gear ratio (Input/Output)
- MOI -- double -- Moment of inertia (kg m^2)
- Estimate MOI -- boolean -- If TRUE, estimate MOI from mass and arm length. (Default: TRUE)

Outputs:

- Single Jointed Arm Model Params -- cluster -- Packed single jointed arm physical modeling parameters

LinearSystemLoop

LinearSystemLoop_ClampInput



Clamp the input u to the min and max.

Inputs:

- LinearSystemLoop -- system data cluster
- unclampedU -- The input to clamp.

Outputs:

- U -- The clamped input.
- SizeCoerced -- If TRUE, an error occurred.

LinearSystemLoop_Correct



Correct the state estimate \hat{x} using the measurements in y.

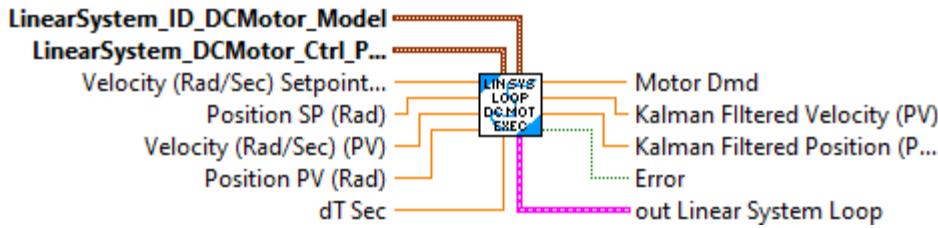
Inputs:

- LinearSystemLoop -- system data cluster
- y -- Measurement vector.

Outputs:

- outLinearSystemLoop -- updated system data cluster
- error -- If TRUE, an error occurred.

LinearSystemLoop_DCMotor_Execute



Model and control a DC Motor system using a Kalman Filter and LQR controller. This is a single call LabVIEW function.

The DC Motor linear system is defined as:

- states:
 - angular position (rad)
 - angular velocity (rad/sec)
- inputs:
 - motor voltage
- outputs:
 - angular position (rad)
 - angular velocity (rad/sec)

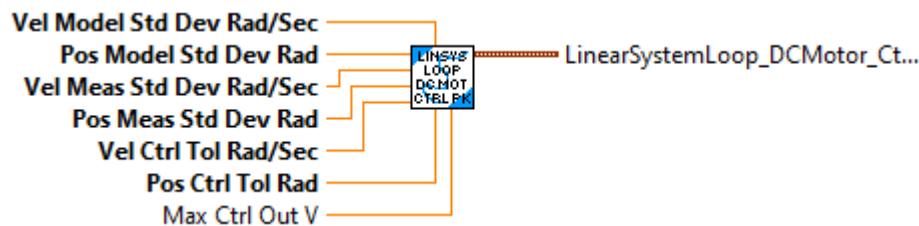
Inputs:

- LinearSystemId DC Motor Model -- cluster -- Packed parameters used to create DC Motor linear system.
- LineaarSystem DC Motor Ctrl -- cluster -- Packed parameters used by Kalman Filter and LQR to control the system.
- Velocity Setpoint -- double -- Desired velocity (Rad/Sec)
- Position Setpiont -- double -- Desired position (Radians, Default: 0)
- Velocity -- double -- Measured velocity (Rad/Sec)
- Position -- double -- Measured position (Radians)
- dT -- double -- Update time (Seconds, Default: 0.02)

Outputs:

- Motor Dmd - double -- Motor demand (+/- 1.0)
 - Kalman Filtered Velocity -- double -- Filtered velocity (Rad/Sec)
 - Kalman Filtered Position -- double -- Filtered position (Radians)
 - Error -- boolean -- If TRUE an error occurred.
-
-

LinearSystemLoop_DCMotor_Pack_Ctrl



Pack the control parameters for a DC motor linear system.

The DC Motor linear system is defined as:

- states:
 - angular position (rad)
 - angular velocity (rad/sec)
- inputs:
 - motor voltage
- outputs:
 - angular position (rad)
 - angular velocity (rad/sec)

Inputs:

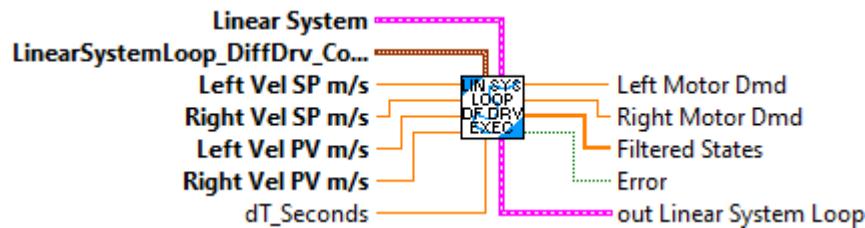
- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (rad/sec)
- Pos Model Std Dev -- double -- Standard deviation for modeled position (rad)

- Vel Meas Std Dev -- double -- Standard deviation for measured velocity (rad/sec)
- Pos Meas Std Dev -- double -- Standard deviation for measured position (rad)
- Vel Ctrl Tol Std Dev -- double -- Control tolerance for velocity (rad/sec)
- Pos Ctrl Tol Std Dev -- double -- Control tolerance for position (rad)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)

Outputs:

- LinearSystemLoop DCMotor Ctrl Params -- cluster -- Packed control parameters

LinearSystemLoop_DiffDrv_Execute



Model and control a Diff Drive system using a Kalman Filter and LQR controller. This is a single call LabVIEW function.

The Diff Drive linear system is define as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

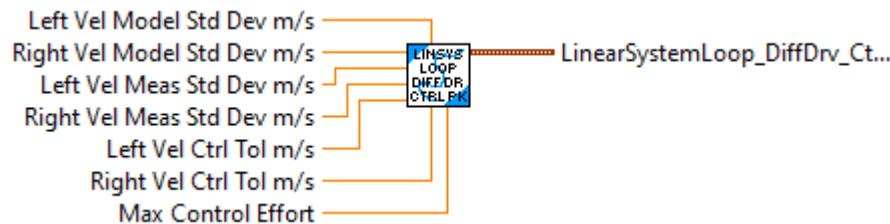
Inputs:

- LinearSystem Diff Drive Model -- Linear System -- Diff Drive linear system.
- LineaarSystem DC Motor Ctrl -- cluster -- Packed parameters used by Kalman Filter and LQR to control the system.
- Left Velocity Setpoint -- double -- Desired left velocity (Meters/Sec)
- Right Velocity Setpoint -- double -- Desired right velocity (Meters/Sec)
- Left Velocity -- double -- Measured left velocity (Meters/Sec)
- Right Velocity -- double -- Measured right velocity (Meters/Sec)
- dT -- double -- Update time (Seconds, Default: 0.02)

Outputs:

- Left Motor Dmd - double -- Left motor demand (+/- 1.0)
- Right Motor Dmd - double -- Right motor demand (+/- 1.0)
- Kalman FIlttered States -- double array -- Filtered velocity states (Meters/Sec)
- out LinearSystemLoop -- cluster -- current internal data cluster
- Error -- boolean -- If TRUE an error occurred.

LinearSystemLoop_DiffDrv_Pack_Ctrl



Pack the control parameters for a Diff Drive linear system.

The Diff Drive linear system is defined as:

- states

- left velocity (m/s)
- right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

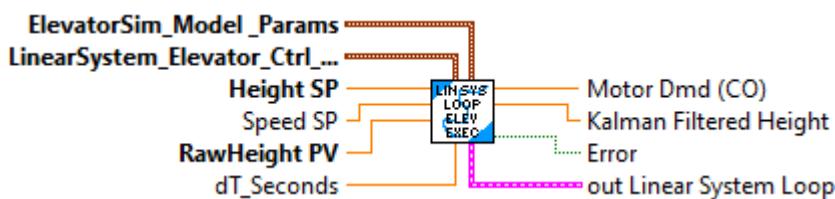
Inputs:

- Left Vel Model Std Dev -- double -- Standard deviation for modeled left velocity (meters/sec)
- Right Vel Model Std Dev -- double -- Standard deviation for modeled right velocity (meters/sec)
- Left Vel Meas Std Dev -- double -- Standard deviation for measured left velocity (meters/sec)
- Right Vel Meas Std Dev -- double -- Standard deviation for measured right velocity (meters/sec)
- Left Vel Ctrl Tol Std Dev -- double -- Control tolerance for left velocity (meters/sec)
- Right Vel Ctrl Tol Std Dev -- double -- Control tolerance for right velocity (meters/sec)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)

Outputs:

- LinearSystemLoop Diff Drive Ctrl Params -- cluster -- Packed control parameters

LinearSystemLoop_Elevator_Execute



Model and control an Elevator system using a Kalman Filter and LQR controller. This is a single call LabVIEW function.

The Elevator linear system definition is:

- States
 - position (meters)
 - velocity (meters/sec)
- Inputs:
 - motor voltage (volts)
- Outputs:
 - position (meters)

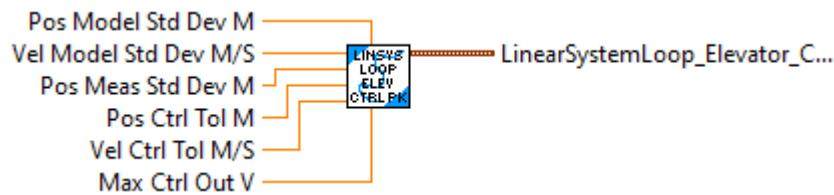
Inputs:

- LinearSystemId Elevator Model -- cluster -- Packed parameters used to create Elevator linear system.
- LineaarSystem Elevator Ctrl -- cluster -- Packed parameters used by Kalman Filter and LQR to control the system.
- Height Setpoint -- double -- Desired height (meters)
- Speed Setpiont -- double -- Desired speed (meters/sec, Default: 0)
- Raw Height -- double -- Measured height (meters)
- dT -- double -- Update time (Seconds, Default: 0.02)

Outputs:

- Motor Dmd - double -- Motor demand (+/- 1.0)
 - Kalman FIlttered Height -- double -- Filtered Height (meters)
 - Error -- boolean -- If TRUE an error occurred.
-

LinearSystemLoop_Elevator_Pack_Ctrl



Pack the control parameters for an elevator linear system.

The Elevator linear system definition is:

- States
 - position (meters)
 - velocity (meters/sec)
- Inputs:
 - motor voltage (volts)
- Outputs:
 - position (meters)

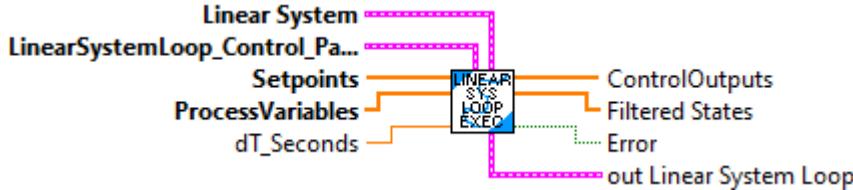
Inputs:

- Pos Model Std Dev -- double -- Standard deviation for modeled position (meters)
- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (meters/sec)
- Pos Meas Std Dev -- double -- Standard deviation for measured position (meters)
- Pos Ctrl Tol Std Dev -- double -- Control tolerance for position (meters)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)

Outputs:

- LinearSystemLoop Elevator Ctrl Params -- cluster -- Packed control parameters

LinearSystemLoop_Execute



This is a single call LabVIEW function to control a linear system using Kalman Filter and LQR controller.

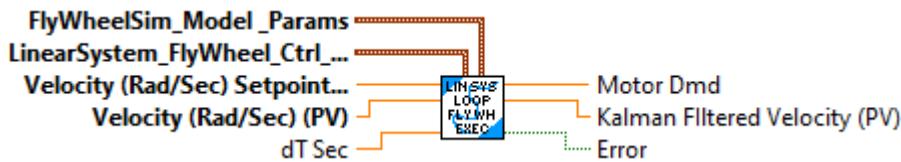
Inputs:

- LinearSystem -- Linear System -- Externally created modeled or characterized (ID'd) state space linear system.
- LineaarSystemLoopControlParams -- cluster -- Packed parameters used by Kalman Filter and LQR to control the system.
- Setpoints -- double array(states) -- Array of desired state setpoints.
- ProcessVariables -- double array(outputs) -- Array of measured process variables (system outputs).
- dT -- double -- Update time (Seconds, Default: 0.02)

Outputs:

- Control Outputs - double array(inputs) -- Array of control outputs (system inputs).
- Filtered States - double array(states) -- Results of the Kalman filtered system states.
- out LinearSystemLoop -- cluster -- current internal data cluster
- Error -- boolean -- If TRUE an error occurred.

LinearSystemLoop_FlyWheel_Execute



Model and control a Flywheel system using a Kalman Filter and LQR controller. This is a single call LabVIEW function.

The flywheel linear system definition is:

- States
 - angular velocity (Rad/Sec)
- Inputs:
 - motor voltage
- Outputs:
 - angular velocity (rad/sec)

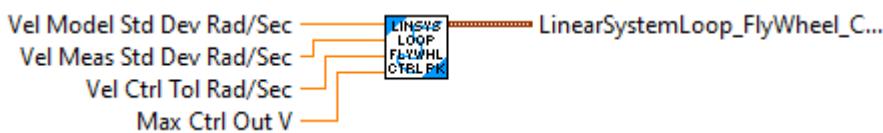
Inputs:

- LinearSystemId Flywheel Model -- cluster -- Packed parameters used to create Flywheel linear system.
- LineaarSystem Flywheel Ctrl -- cluster -- Packed parameters used by Kalman Filter and LQR to control the system.
- Velocity Setpiont -- double -- Desired speed (Rad/sec)
- Velocity -- double -- Measured velocity (Rad/Sec)
- dT -- double -- Update time (Seconds, Default: 0.02)

Outputs:

- Motor Dmd - double -- Motor demand (+/- 1.0)
- Kalman FIlttered Speed -- double -- Filtered Height (Rad/Sec)
- Error -- boolean -- If TRUE an error occured.

LinearSystemLoop_FlyWheel_Pack_Ctrl



Pack the control parameters for a flywheel linear system.

The flywheel linear system definition is:

- States

- angular velocity (Rad/Sec)

- Inputs:

- motor voltage

- Outputs:

- angular velocity (rad/sec)

Inputs:

- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (rad/sec)
- Vel Meas Std Dev -- double -- Standard deviation for measured velocity (rad/sec)
- Vel Ctrl Tol Std Dev -- double -- Control tolerance for velocity (rad/sec)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)

Outputs:

- LinearSystemLoop Flywheel Ctrl Params -- cluster -- Packed control parameters

LinearSystemLoop_GetController



Return the controller used internally.

Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- Controller -- the controller data cluster used internally.

LinearSystemLoop_GetError



Returns difference between reference r and current state x-hat.

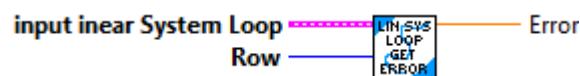
Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- Error -- The state error matrix.

LinearSystemLoop_GetError_Single



Returns difference between reference r and current state x-hat.

Inputs:

- LinearSystemLoop -- system data cluster
- index -- The index of the error matrix to return.

Outputs:

- Error -- The error at that index.

LinearSystemLoop_GetFeedForward



Return the feedforward used internally.

Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- FeedForward -- the feedforward data cluster used internally.
-
-

LinearSystemLoop_GetNextR



Returns the controller's next reference r.

Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- NextR -- the controller's next reference r.
-
-

LinearSystemLoop_GetNextR_Single



Returns an element of the controller's next reference r.

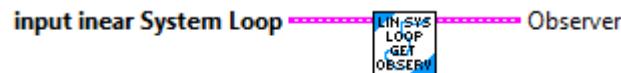
Inputs:

- LinearSystemLoop -- system data cluster
- row -- Row of r.

Outputs:

- NextR -- the element i of the controller's next reference r.
-

LinearSystemLoop_GetObserver



Return the observer used internally.

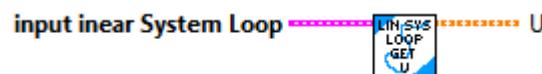
Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- Observer -- the observer data cluster used internally.
-

LinearSystemLoop_GetU



Returns the controller's calculated control input u plus the calculated feedforward u_ff.

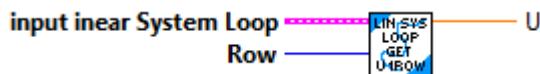
Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- U -- the calculated control input u.
-

LinearSystemLoop_GetU_Row



Returns an element of the controller's calculated control input u.

Inputs:

- LinearSystemLoop -- system data cluster
- row -- Row of u.

Outputs:

- U -- the calculated control input u at the row i.

LinearSystemLoop_GetXHat



Returns the observer's state estimate x-hat.

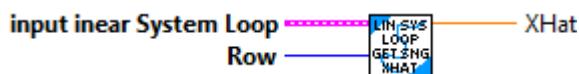
Inputs:

- LinearSystemLoop -- system data cluster

Outputs:

- XHat -- the observer's state estimate x-hat.

LinearSystemLoop_GetXHat_Single



Returns an element of the observer's state estimate x-hat.

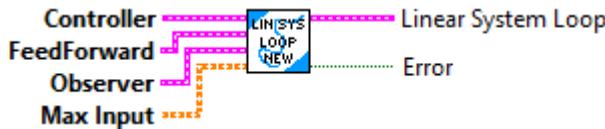
Inputs:

- LinearSystemLoop -- system data cluster
- row -- Row of \hat{x} .

Outputs:

- \hat{X} -- the i-th element of the observer's state estimate \hat{x} .

LinearSystemLoop_New



Combines a controller, feedforward, and observer for controlling a mechanism with full state feedback.

For everything in this file, "inputs" and "outputs" are defined from the perspective of the plant. This means U is an input and Y is an output (because you give the plant U (powers) and it gives you back a Y (sensor values)). This is the opposite of what they mean from the perspective of the controller (U is an output because that's what goes to the motors and Y is an input because that's what comes back from the sensors).

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a state-space loop with the given controller, feedforward and observer. By default, the initial reference is all zeros. Users should call reset with the initial system state before enabling the loop.

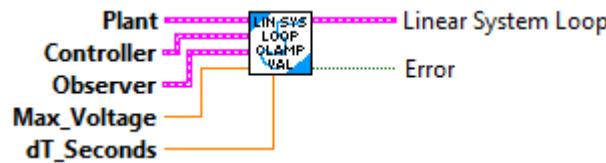
Inputs:

- controller -- State-space controller.
- feedforward -- Plant inversion feedforward.
- observer -- State-space observer.
- maxVoltageVolts -- The maximum voltage that can be applied. Assumes that the inputs are voltages.

Outputs:

- outLinearSystemLoop -- updated system data cluster
- error -- If TRUE, an error occurred.

LinearSystemLoop_New_LinearSystem_ClampVal



Combines a controller, feedforward, and observer for controlling a mechanism with full state feedback.

For everything in this file, "inputs" and "outputs" are defined from the perspective of the plant. This means U is an input and Y is an output (because you give the plant U (powers) and it gives you back a Y (sensor values)). This is the opposite of what they mean from the perspective of the controller (U is an output because that's what goes to the motors and Y is an input because that's what comes back from the sensors).

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf>.

Constructs a state-space loop with the given plant, controller, and observer. By default, the initial reference is all zeros. Users should call reset with the initial system state before enabling the loop. This constructor assumes that the input(s) to this system are voltage.

Inputs:

- plant -- State-space plant.
- controller -- State-space controller.
- observer -- State-space observer.
- maxVoltageVolts -- The maximum voltage that can be applied. Commonly 12.
- dtSeconds -- The nominal timestep.

Outputs:

- outLinearSystemLoop -- updated system data cluster
 - error -- If TRUE, an error occurred.
-

LinearSystemLoop_Pack_Ctrl_Params



Pack control parameters needed for the Linear System Loop Execute function.

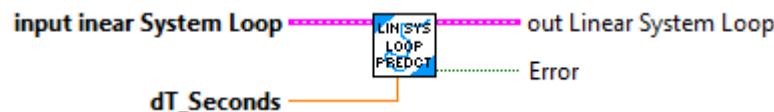
Inputs:

- Model Std Dev -- double array (states) -- Array of standard deviations for model states.
- Measure Std Dev -- double array(outputs) -- Array of standard deviations for model outputs
- Control Tol -- double array(states) -- Array of standard deviations for controlled model states.
- Max Control Effort -- double array(inputs) -- Array of maximum values for inputs (classical control outputs)

Outputs:

- LinearSystemLoopControlParams -- cluster -- Packed values.
-

LinearSystemLoop_Predict



Sets new controller output, projects model forward, and runs observer prediction.

After calling this, the user should send the elements of u to the actuators.

Inputs:

- LinearSystemLoop -- system data cluster
- dtSeconds -- Timestep for model update.

Outputs:

- outLinearSystemLoop -- updated system data cluster
- error -- If TRUE, an error occurred.

LinearSystemLoop_Reset



Zeroes reference r and controller output u. The previous reference of the PlantInversionFeedforward and the initial state estimate of the KalmanFilter are set to the initial state provided.

Inputs:

- LinearSystemLoop -- system data cluster
- initialState -- The initial state.

Outputs:

- outLinearSystemLoop -- updated system data cluster
- sizeCoerced -- If TRUE, an error occurred.

LinearSystemLoop_SetNextR



Set the next reference r.

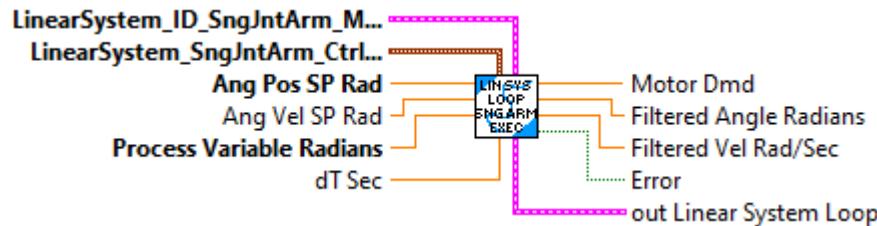
Inputs:

- LinearSystemLoop -- system data cluster
- nextR -- Next reference.

Outputs:

- outLinearSystemLoop -- updated system data cluster
- sizeCoerced -- If TRUE, an error occurred.

LinearSystemLoop_SngJntArm_Execute



Model and control a single jointed arm system using a Kalman Filter and LQR controller. This is a single call LabVIEW function.

The single jointed arm linear system definition is:

- States
 - angle position (Rad)
 - velocity (Rad/Sec)
- Inputs:
 - motor voltage
- Outputs:
 - angle position (rad)

Inputs:

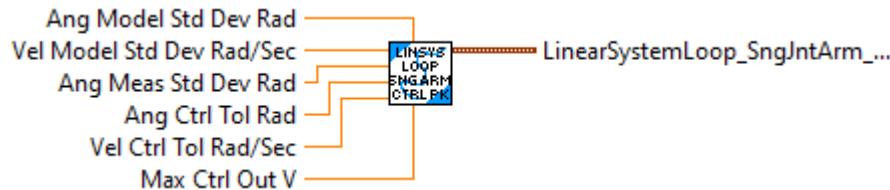
- LinearSystemId SngJntArm Model -- cluster -- Packed parameters used to create Single Jointed Arm linear system.

- LineaarSystem SngJntArm Ctrl -- cluster -- Packed parameters used by Kalman Filter and LQR to control the system.
- Ang Pos Setpoint -- double -- Desired arm angle (radians)
- Ang Vel Setpoint -- double -- Desired arm speed setpoint (Rad/sec, Default: 0)
- PV -- double -- Measured arm angle (radians)
- dT -- double -- Update time (Seconds, Default: 0.02)

Outputs:

- Motor Dmd - double -- Motor demand (+/- 1.0)
- Kalman Filtered angle -- double -- Filtered angle (radians)
- Kalman filtered velocity -- double -- Filtered velocity (Rad/Sec)
- Error -- boolean -- If TRUE an error occurred.

LinearSystemLoop_SngJntArm_Pack_Ctrl



Pack the control parameters for a single jointed arm linear system.

The single jointed arm linear system definition is:

- States
 - angle position (Rad)
 - velocity (Rad/Sec)

- Inputs:

- motor voltage

- Outputs:

- angle position (rad)

Inputs:

- Ang Model Std Dev -- double -- Standard deviation for modeled angle (Rad)
- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (rad/sec)
- Ang Measl Std Dev -- double -- Standard deviation for measured angle (Rad)
- Ang Ctrl Tol Std Dev -- double -- Control tolerance for angle (rad)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)

Outputs:

- LinearSystemLoop Single JJointed Arm Ctrl Params -- cluster -- Packed control parameters

LinearSystemSim

LinearSystemSim_ClampInput



Clamp the input vector such that no element exceeds the given voltage. If any does, the relative magnitudes of the input will be maintained.

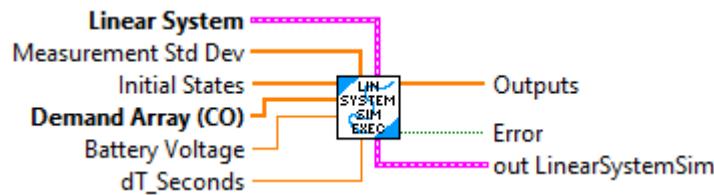
Inputs:

- Input -- The input vector (u).
- BatteryVolts -- The current battery voltage (volts)

Outputs:

- ClampedInput -- The normalized input.
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

LinearSystemSim_Execute



Single call LabVIEW function to simulate a linear system.

Inputs:

- Linear System -- Linear System -- The linear system to be simulated.
- Measurement Std Dev -- double array (outputs) -- Optional, simulation output standard deviations (Default: 0.0)

- Initial States -- double array (states) -- Optional, contains initial state values. (Default: contents of linear system)
- Demand Array (CO) -- double array(inputs) -- demand values. (+/- 1.0) These are turned into voltages by multiplying by the battery voltage.
- Battery Voltage -- double -- Optional, current simulated battery voltage (Volts Default: 12.0)
- dT Sec -- double -- Optional,, update period (Seconds, Default: 0.02)

Outputs:

- Outputs -- double array (outputs) -- Current system outputs.
- Error -- boolean -- If TRUE an error occurred.
- outLinearSystemSim -- cluster -- Current internal data cluster.

LinearSystemSim_GetOutput



Returns the current output of the plant.

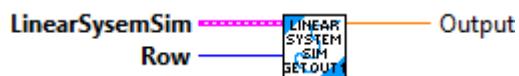
Inputs:

- LinearSystemSim -- Data cluster

Outputs:

- Output -- The current output matrix of the plant.

LinearSystemSim_GetOutput_Single



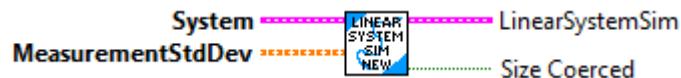
Returns an element of the current output of the plant.

Inputs:

- LinearSystemSim -- Data cluster
- row -- The row to return.

Outputs:

- Output -- An element of the current output of the plant.
-
-

LinearSystemSim_New

Creates a simulated generic linear system with measurement noise.

This class helps simulate linear systems.

Call `setInput(double...)` with the inputs to the system (usually voltage).

Call `update` to update the simulation.

Set simulated sensor readings with the simulated positions in `getOutput()`. (Outputs of the simulation are inputs to the robot -- sensors.)

Inputs:

- system -- The Linear System being controlled.
- measurementStdDevs -- Standard deviations of measurements. Can be null if no noise is desired.

Outputs:

- OutLinearSystemSim -- Updated data cluster
- Error -- If TRUE, an error occurred.

LinearSystemSim_SetInput_Array



Sets the system inputs.

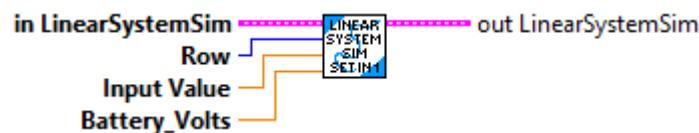
Inputs:

- LinearSystemSim -- Data cluster
- Inputs -- An array of doubles that represent the inputs (u) of the system.

Outputs:

- OutLinearSystemSim -- Updated data cluster
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

LinearSystemSim_SetInput_Single



Sets the system inputs.

Inputs:

- LinearSystemSim -- Data cluster
- row -- The row in the input matrix to set.
- value -- The value to set the row to.
- BatteryVoltage -- The current battery voltage (volts)

Outputs:

- OutLinearSystemSim -- Updated data cluster

LinearSystemSim_Update



Updates the simulation.

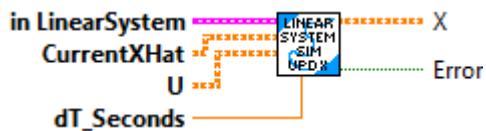
Inputs:

- LinearSystemSim -- Data cluster
- dtSeconds -- The time between updates.

Outputs:

- OutLinearSystemSim -- Updated data cluster
- Error -- If TRUE, an error occurred.

LinearSystemSim_UpdateX



Updates the state estimate of the system.

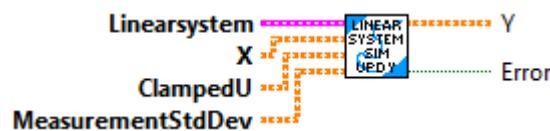
Inputs:

- LinearSystem -- Data cluster
- currentXhat -- The current state estimate.
- u -- The system inputs (usually voltage).
- dtSeconds -- The time difference between controller updates.

Outputs:

- X -- The new state.
 - Error -- If TRUE, an error occurred.
-

LinearSystemSim_UpdateY



Updates the Y matrix of the system.

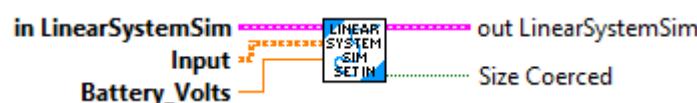
Inputs:

- LinearSystem -- Data cluster
- currentX -- The current state estimate.
- ClampedU -- The system inputs (usually voltage).
- MeasurementStdDev -- Standard deviation vector matrix for measurements

Outputs:

- Y -- The new Y matrix.
 - Error -- If TRUE, an error occurred.
-

LinearSystemSim_SetInput



Sets the system inputs (usually voltages).

Inputs:

- LinearSystemSim -- Data cluster
- Input -- The system inputs (u).
- BatteryVolts -- Current battery voltage (volts)

Outputs:

- OutLinearSystemSim -- Updated data cluster
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

LinearSystemSim_setState



Sets the system state.

Inputs:

- LinearSystemSim -- Data cluster
- SystemState -- The new state.

Outputs:

- OutLinearSystemSim -- Updated data cluster
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

LTVDiffDriveCtrl

LTVDiffDriveCtrl_AtReference



Returns true if the pose error is within tolerance of the reference.

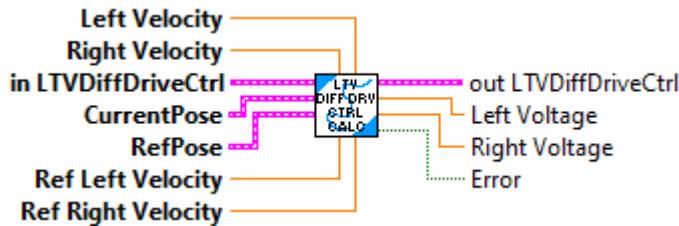
Inputs:

- LTV Diff Drive Ctrl -- Controller data cluster

Outputs:

- At Reference -- True if the pose error is within tolerance of the reference.

LTVDiffDriveCtrl_Calculate



Returns the left and right output voltages of the LTV controller.

The reference pose, linear velocity, and angular velocity should come from a drivetrain trajectory.

Inputs:

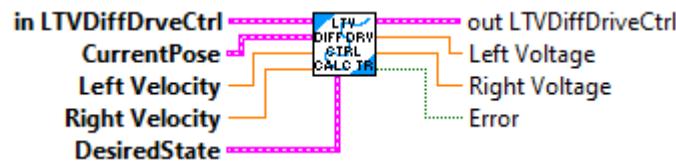
- in LTV Dif Drive Ctrl -- Controller data cluster.
- currentPose -- The current pose.
- leftVelocity -- The current left velocity in meters per second.
- rightVelocity -- The current right velocity in meters per second.
- poseRef -- The desired pose.

- leftVelocityRef -- The desired left velocity in meters per second.
- rightVelocityRef -- The desired right velocity in meters per second.

Outputs:

- out LTV Dif Drive Ctrl -- Updated controller data cluster.
- Left Voltage -- Left wheel output voltages of the LTV controller.
- Right Voltage -- Right wheel output voltages of the LTV controller.
- Error -- Returns TRUE if an error occurred.

LTVDiffDriveCtrl_Calculate_TrajState



Returns the left and right output voltages of the LTV controller.

The reference pose, linear velocity, and angular velocity should come from a drivetrain trajectory.

Inputs:

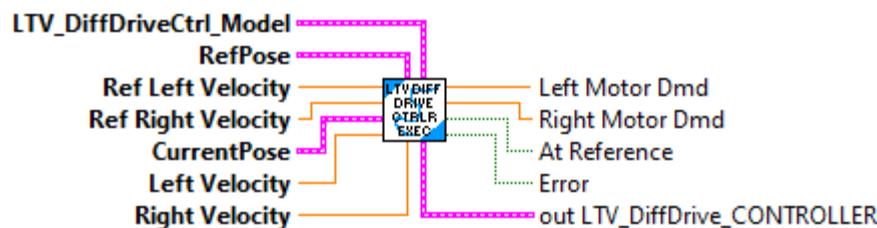
- in LTV Dif Drive Ctrl -- Controller data cluster.
- currentPose -- The current pose.
- leftVelocity -- The current left velocity in meters per second.
- rightVelocity -- The current right velocity in meters per second.
- desiredState -- The desired pose, linear velocity, and angular velocity from a trajectory.

Outputs:

- out LTV Dif Drive Ctrl -- Updated controller data cluster.
- Left Voltage -- Left wheel output voltages of the LTV controller.

- Right Voltage -- Right wheel output voltages of the LTV controller.
- Error -- Returns TRUE if an error occurred.

LTVDiffDriveCtrl_Execute



The linear time-varying (LTV) differential drive controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state. We precomputed gains for important places in our state-space, then interpolated between them with a look up table (LUT) to save computational resources. Filters the provided voltages to limit a differential drive's linear and angular acceleration.

This LabVIEW single call function creates and executes the LTV Diff Drive controller.

The input linear system is defined as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

Inputs:

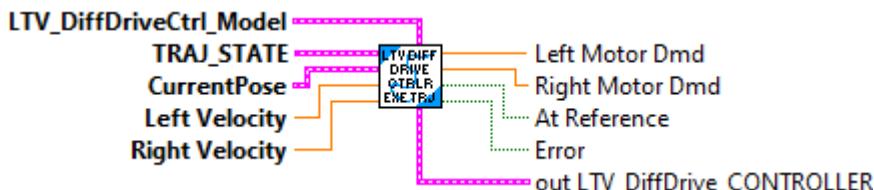
- LTVDiffDriveCtrl Model -- cluster -- Packed data used to construct the LTV Diff Drive control system.

- Ref Pose -- Pose2d -- The desired pose setpoint. (Meters, Radians)
- Ref Left Velocity -- double -- The desired left velocity (Meters/sec)
- Ref Right Velocity -- double -- The desired right velocity (Meters/sec)
- Current Pose -- Pose2d -- The current pose. (Meters, Radians)
- Left Velocity -- double -- The current left velocity (Meters/sec)
- Right Velocity -- double -- The current right velocity (Meters/sec)

Outputs:

- Left Motor Dmd -- double -- Left wheel motor demand (+/- 1.0)
- Right Motor Dmd -- double -- Left wheel motor demand (+/- 1.0)
- At Reference -- boolean -- If TRUE, the current position is within the tolerance.
- Error -- boolean -- If TRUE, an error occurred.
- out LTV DiffDriveCtrl -- cluster -- Internal data structure.

LTVDiffDriveCtrl_Execute_TrajState



The linear time-varying (LTV) differential drive controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state. We precomputed gains for important places in our state-space, then interpolated between them with a look up table (LUT) to save computational resources. Filters the provided voltages to limit a differential drive's linear and angular acceleration.

This LabVIEW single call function creates and executes the LTV Diff Drive controller.

The input linear system is defined as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

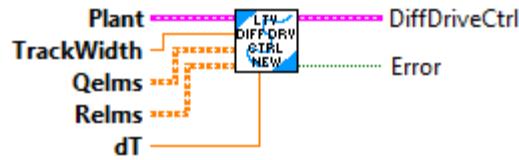
Inputs:

- LTVDiffDriveCtrl Model -- cluster -- Packed data used to construct the LTV Diff Drive control system.
- Traj state -- Traj state -- The desired trajectory state. Contains desired position and velocity values. (Meters, radians)
- Ref Left Velocity -- double -- The desired left velocity (Meters/sec)
- Ref Right Velocity -- double -- The desired right velocity (Meters/sec)
- Current Pose -- Pose2d -- The current pose. (Meters, Radians)
- Left Velocity -- double -- The current left velocity (Meters/sec)
- Right Velocity -- double -- The current right velocity (Meters/sec)

Outputs:

- Left Motor Dmd -- double -- Left wheel motor demand (+/- 1.0)
- Right Motor Dmd -- double -- Left wheel motor demand (+/- 1.0)
- At Reference -- boolean -- If TRUE, the current position is within the tolerance.
- Error -- boolean -- If TRUE, an error occurred.
- out LTV DiffDriveCtrl -- cluster -- Internal data structure.

LTVDiffDriveCtrl_New



The linear time-varying (LTV) differential drive controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state. We precomputed gains for important places in our state-space, then interpolated between them with a LUT to save computational resources. Filters the provided voltages to limit a differential drive's linear and angular acceleration.

The input linear system is defined as:

The linear system is define as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)
 - right velocity (m/s)

Constructs a linear time-varying differential drive controller.

Inputs:

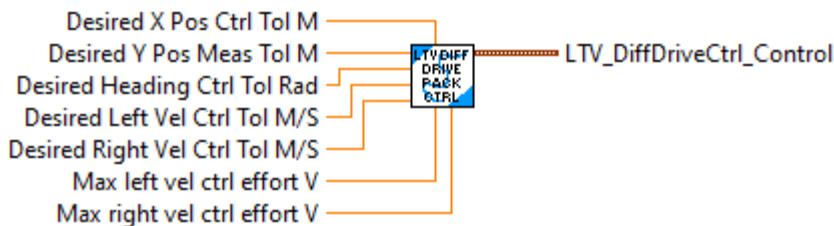
- plant -- The drivetrain velocity plant.
- trackwidth -- The drivetrain's trackwidth in meters.

- qelems -- The maximum desired error tolerance for each state. Matrix (5,1)
- relems -- The maximum desired control effort for each input. Matrix (2,1)
- dt -- Discretization timestep in seconds.

Outputs:

- DiffDriveCtrl -- Created data cluster
- Error -- If TRUE, an error occurred.

LTVDiffDriveCtrl_Pack_Ctrl_Params



Pack control values for LTV Diff Drive Ctrl execute function

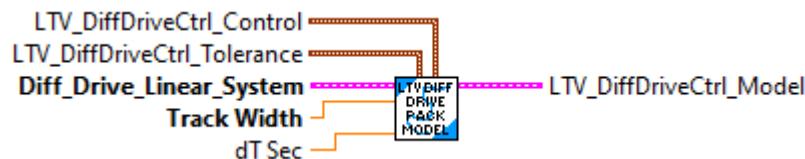
Inputs:

- Desired X Pos Ctrl Tol -- double --- Maximum desired X position control tolerance (meters, Default: 0.0625)
- Desired Y Pos Ctrl Tol -- double --- Maximum desired y position control tolerance (meters, Default: 0.125)
- Desired Heading Ctrl Tol -- double --- Maximum desired heading control tolerance (meters, Default: 0.2)
- Desired Left Vel Ctrl Tol -- double --- Maximum desired left velocity control tolerance (meters, Default: 0.3)
- Desired Right Vel Ctrl Tol -- double --- Maximum desired right velocity control tolerance (meters, Default: 0.3)
- Max left vel ctrl effort V -- double -- Maximum left motor dmd control effort (volts, default 12.0)
- Max right vel ctrl effort V -- double -- Maximum right motor dmd control effort (volts, default 12.0)

Outputs:

- LTV Diff Drive Ctrl Control -- cluster -- Packed values

LTVDiffDriveCtrl_Pack_Model_Params



Pack model, control, and tolerance values for LTV Diff Drive control execute function:

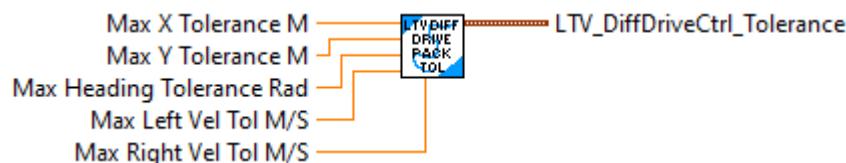
Inputs:

- LTV Diff Drve Ctrl -- cluster -- Packed control parameters
- LTV Diff Drive Tol -- cluster -- Packed tolerance parameters
- Diff Drive Linear System -- Linear system -- Diff Drive linear system
- Track Width -- double -- Track width (meters)
- dT Sec -- double -- Update time (Default 0.02)

Outputs:

- LTVDiffDrveCtrlModel -- cluster -- Packed model, control, tolerance values

LTVDiffDriveCtrl_Pack_Tolerance



Pack on target tolerance values for LTV Diff Drive Ctrl execute function

Inputs:

- Max X tolerance -- double --- Maximum desired X position tolerance (meters, Default: 0.04)
- Max Y tolerance -- double --- Maximum desired Y position tolerance (meters, Default 0.04)
- Max heading tolerance -- double --- Maximum desired heading tolerance (radians, Default 0.06)
- Max left vel tolerance -- double --- Maximum desired left velocity tolerance (meters/sec, Default 0.15)
- Max right vel tolerance -- double --- Maximum desired right velocity tolerance (meters/sec, Default: 0.15)

Outputs:

- LTV Diff Drive Ctrl Tol -- cluster -- Packed values

LTVDiffDriveCtrl_SetTolerance



Sets the pose error which is considered tolerable for use with the 'At Reference' function.

Inputs:

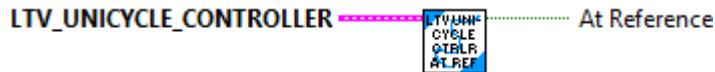
- LTV Diff Drive Ctrl -- Controller data cluster
- poseTolerance -- Pose error which is tolerable.
- leftVelocityTolerance -- Left velocity error which is tolerable in meters per second.
- rightVelocityTolerance -- Right velocity error which is tolerable in meters per second

Outputs:

- out LTV Diff Drive Ctrl -- Updated controller data cluster

LTVUnicycleCtrl

LTVUnicycleCtrl_AtReference



Returns true if the pose error is within tolerance of the reference.

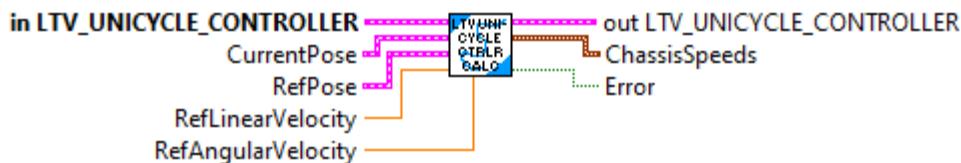
Inputs:

- LTV Unicycle Controller -- Data cluster

Outputs:

- At Reference -- True if the pose error is within tolerance of the reference.

LTVUnicycleCtrl_Calculate



Returns the linear and angular velocity outputs of the LTV controller.

Inputs:

- in LTV Unicycle Controller -- Controller data cluster
- currentPose -- The current pose.
- poseRef -- The desired pose. (Setpoint)
- linearVelocityRef -- The desired linear velocity in meters per second. (Feedforward)
- angularVelocityRef -- The desired angular velocity in radians per second. (Feedforward)

Outputs:

- out LTV Unicycle Controller -- Updated data cluster
 - Chassis Speeds -- The linear and angular velocity outputs of the LTV controller.
 - Error -- Returns TRUE if an error occurred.
-
-

LTVUnicycleCtrl_Calculate_TrajState



Returns the linear and angular velocity outputs of the LTV controller.

The reference pose, linear velocity, and angular velocity should come from a drivetrain trajectory.

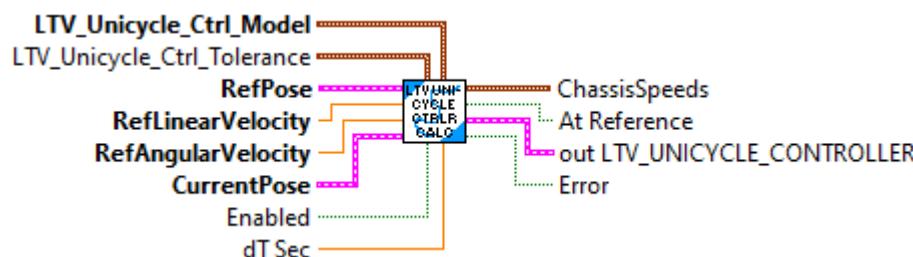
Inputs:

- in LTV Unicycle Controller -- Controller data cluster
 - currentPose -- The current pose.
- Traj State -- The desired pose, linear velocity, and angular velocity from a trajectory.

Outputs:

- out LTV Unicycle Controller -- Updated data cluster
 - Chassis Speeds -- The linear and angular velocity outputs of the LTV controller.
 - Error -- Returns TRUE if an error occurred.
-
-

LTVUnicycleCtrl_Execute



The linear time-varying unicycle controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state.

This LabVIEW single call function creates and executes the LTV Unicycle controller.

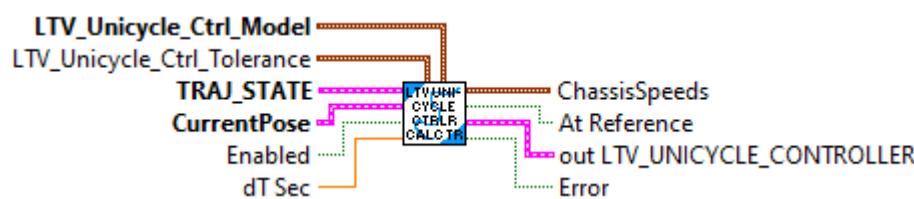
Inputs:

- LTVUnicycleCtrl Model -- cluster -- Packed data used to construct the LTV Unicycle control system.
- LTVUnicycleCtrl Tol -- cluster -- Packed data used to define the tolerance LTV Unicycle control system.
- Ref Pose -- Pose2d -- The desired pose setpoint. (Meters, Radians)
- Ref Linear Velocity -- double -- The desired linear velocity (Meters/sec)
- Ref Angular Velocity -- double -- The desired angular velocity (Radians/sec)
- Current Pose -- Pose2d -- The current pose. (Meters, Radians)
- Enabled -- boolean -- If TRUE use closed loop control to calculate desired chassis speeds. If FALSE, outputs the reference speeds directly.
- dt Sec -- double -- Update time (default: 0.02)

Outputs:

- Chassis Speed -- chassis speed -- Chassis Speed Demand
- At Reference -- boolean -- If TRUE, the current position is within the tolerance.
- Error -- boolean -- If TRUE, an error occurred.
- out LTV UnicycleCtrl -- cluster -- Internal data structure.

LTVUnicycleCtrl_Execute_TrajState



The linear time-varying unicycle controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state.

This LabVIEW single call function creates and executes the LTV Unicycle controller.

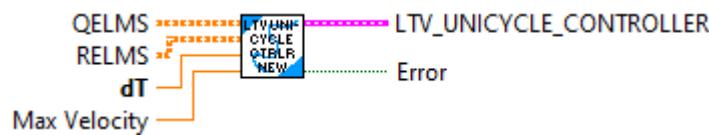
Inputs:

- LTVUnicycleCtrl Model -- cluster -- Packed data used to construct the LTV Unicycle control system.
- LTVUnicycleCtrl Tol -- cluster -- Packed data used to define the tolerance LTV Unicycle control system.
- Traj State -- Traj State -- The desired trajectory state. contains desired position and speeds (Meters, Radians)
- Current Pose -- Pose2d -- The current pose. (Meters, Radians)
- Enabled -- boolean -- If TRUE use closed loop control to calculate desired chassis speeds. If FALSE, outputs the reference speeds directly.
- dt Sec -- double -- Update time (default: 0.02)

Outputs:

- Chassis Speed -- chassis speed -- Chassis Speed Demand
- At Reference -- boolean -- If TRUE, the current position is within the tolerance.
- Error -- boolean -- If TRUE, an error occurred.
- out LTV UnicycleCtrl -- cluster -- Internal data structure.

LTVUnicycleCtrl_New



The linear time-varying unicycle controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state.

Constructs a linear time-varying unicycle controller.

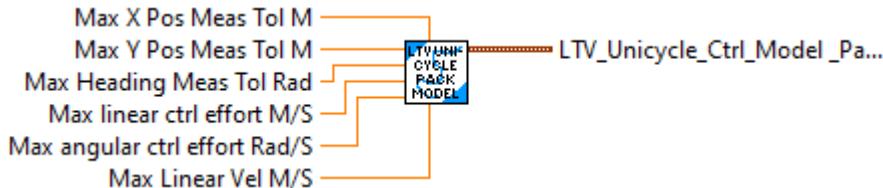
Inputs:

- qelems -- The maximum desired error tolerance for each state. Matrix (3,1) Defaults (0.0625 m, 0.125 m, 0.2 rad)
- relems -- The maximum desired control effort for each input. Matrix (2,1) Defaults (1 m/s, 2 rad/s)
- dt -- Discretization timestep in seconds.
- maxVelocity -- The maximum velocity in meters per second for the controller gain lookup table. (Option) The default is 9 m/s.

Outputs:

- LTV Unicycle Controller -- Created data structure.
- Error -- If TRUE, an error occurred.

LTVUnicycleCtrl_Pack_Model_Parms



Pack model and control values for LTV Unicycle Ctrl execute function

Inputs:

- Max X Pos Meas Tol -- double --- Maximum desired X position measurement tolerance (meters, Default: 0.0625)
- Max Y Pos Meas Tol -- double --- Maximum desired y position measurement tolerance (meters, Default: 0.125)
- Max Heading Meas Tol -- double --- Maximum desired heading measurement tolerance (meters, Default: 0.2)
- Max angular ctrl effort V -- double -- Maximum angular control effort (rad/sec, default 2.0)
- Max linear vel ctrl effort V -- double -- Maximum linearcontrol effort (meters/sec, default 9.0)

Outputs:

- LTV Unicycle Ctrl Control -- cluster -- Packed values

LTVUnicycleCtrl_Pack_Tolerance



Pack on target tolerance values for LTV Unicycle Ctrl execute function

Inputs:

- Max X tolerance -- double --- Maximum desired X position tolerance (meters, Default: 0.03)
- Max Y tolerance -- double --- Maximum desired Y position tolerance (meters, Default 0.03)
- Max heading tolerance -- double --- Maximum desired heading tolerance (radians, Default 0.07)

Outputs:

- LTV Unicycle Ctrl Tol -- cluster -- Packed values

LTVUnicycleCtrl_SetEnabled



Enables and disables the controller for troubleshooting purposes.

When not enabled, the open loop reference values are passed through to the outputs.

Inputs:

- in LTV Unicycle Controller -- Controller data cluster
- Enabled -- If the controller is enabled or not.

Outputs:

- out LTV Unicycle Controller -- Updated data cluster
-
-

LTVUnicycleCtrl_SetTolerance



Sets the pose error which is considered tolerable for use with AtReference.

The tolerance doesn't play a role in the control function. It is only used by At Reference.

Inputs:

- in LTV Unicycle Controller -- Data cluster
- Tolerance -- Pose error which is tolerable.

Outputs:

- out LTV Unicycle Controller -- Updated data cluster

MatBuilder

MatBuilder_Create



Creates a new matrix with the given dimensions.

Inputs:

- Rows -- Number of rows in the matrix
- Columns -- Number of columns in the matrix

Outputs:

- Output Matrix -- The resulting matrix. All values are zero.

MatBuilder_Fill



Fills the matrix with the given data, encoded in row major form. (The matrix is filled row by row, left to right with the given data).

Inputs:

- Value List -- Array of values used to fill the matrix
- Rows -- The number of rows in the resulting matrix
- Columns -- The number of columns in the resulting matrix

Output:

- Output Matrix -- The resulting filled matrix.

MathUtil

MathUtil_AngleModulus



Wraps an angle to the range -pi to pi radians.

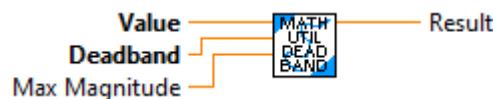
Inputs:

- angleRadians -- Angle to wrap in radians.

Outputs:

- modulusAngle -- The wrapped angle.

MathUtil_ApplyDeadband



Returns 0.0 if the given value is within the specified range around zero. The remaining range between the deadband and the maximum magnitude is scaled from 0.0 to the maximum magnitude.

Inputs:

- value -- Value to clip.
- deadband -- Range around zero.
- max magnitude -- The maximum magnitude of the input. Can be infinite. (Default: 1.0)

Outputs:

- Result -- The value after the deadband is applied.

MathUtil_Clamp



Returns value clamped between low and high boundaries.

Inputs:

- value -- Value to clamp.
- low limit -- The lower boundary to which to clamp value.
- high limit -- The higher boundary to which to clamp value.

Outputs:

- Result -- The clamped value.

MathUtil_Clamp_Int



Returns value clamped between low and high boundaries.

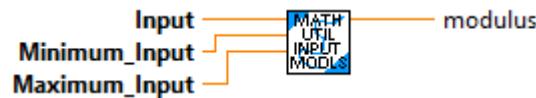
Inputs:

- value -- Value to clamp.
- low -- The lower boundary to which to clamp value.
- high -- The higher boundary to which to clamp value.

Outputs:

- Result -- The clamped value.

MathUtil_InputModulus



Returns modulus of input. This is used for continuous systems, for example rotational systems.

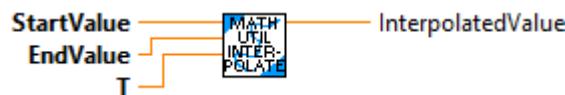
Inputs:

- input -- Input value to wrap.
- minimumInput -- The minimum value expected from the input.
- maximumInput -- The maximum value expected from the input.

Outputs:

- modulus -- The wrapped value.
- AltModulus -- The wrapped value calculated a different way (used only for debugging).
- Different -- If TRUE, then the modulus and AltModulus are different... (used only for debugging)

MathUtil_Interpolate



Perform linear interpolation between two values.

Inputs:

- StartValue -- The value to start at.
- EndValue -- The value to end at.
- t -- How far between the two values to interpolate. This is clamped to [0, 1].

Outputs:

- interpolatedValue -- The interpolated value.

Matrix

Matrix_AssignBlock



Assign a matrix of a given size and start position.

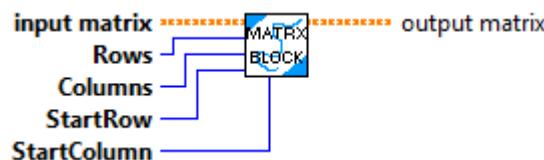
Inputs:

- input matrix -- The input matrix
- other -- The matrix to assign the block to.
- startingRow -- The row to start at.
- startingCol -- The column to start at.

Outputs:

- output matrix -- Resulting matrix

Matrix_Block



Extracts a matrix of a given size and start position with new underlying storage.

Inputs:

- input matrix -- The input matrix
- Rows -- Number of rows to extract.

- Columns -- Number of columns to extract.
- startingRow -- The starting row of the extracted matrix.
- startingCol -- The starting column of the extracted matrix.

Outputs:

- output matrix -- The extracted matrix.
-

Matrix_Create



Constructs an empty zero matrix of the given dimensions.

Inputs:

- rows -- The number of rows of the matrix.
- columns -- The number of columns of the matrix.

Outputs:

- output matrix -- Newly created matrix
-

Matrix_Diag



Returns the diagonal elements inside a vector or square matrix.

If "this" Matrix is a vector then a square matrix is returned. If a "this" Matrix}is a matrix then a vector of diagonal elements is returned.

Inputs:

- input matrix -- The input matrix

Outputs:

- output matrix -- The diagonal elements inside a vector or a square matrix.
-
-

Matrix_ElementSum



Computes the sum of all the elements in the matrix.

Inputs:

- input matrix -- The input matrix

Outputs:

- Sum -- Sum of all the elements.
-
-

Matrix_Exp



Computes the matrix exponential using Eigen's solver. This method only works for square matrices, and will otherwise throw an MatrixDimensionException.

Inputs:

- input matrix -- The input matrix

Outputs:

- output matrix -- The exponential of A.
 - error -- If TRUE, an error occurred.
-

Matrix_ExtractColumnVector



Extracts a given column into a column vector with new underlying storage.

Inputs:

- input matrix -- The input matrix
- column -- The column to extract a vector from.

Outputs:

- output matrix -- A column vector from the given column.
 - error -- If TRUE, an error occurred.
-

Matrix_ExtractFrom



Extracts a submatrix from the supplied matrix and inserts it in a submatrix in "this". The shape of "this" is used to determine the size of the matrix extracted.

Inputs:

- input matrix -- The input matrix
- startingRow -- The starting row in the supplied matrix to extract the submatrix.

- startingCol -- The starting column in the supplied matrix to extract the submatrix.
- other -- The matrix to extract the submatrix from.

Outputs:

- output column -- the column matrix resulting from the extraction

Matrix_ExtractRowVector



Extracts a given row into a row vector with new underlying storage.

Inputs:

- input matrix -- The input matrix
- row -- The row to extract a vector from.

Outputs:

- output row -- A row vector from the given row.
- error -- If TRUE, an error occurred.

Matrix_Fill



Sets all the elements in "this" matrix equal to the specified value.

Inputs:

- input matrix -- The input matrix
- value -- The value each element is set to.

Outputs:

- output matrix -- resulting matrix

Matrix_Ident



Creates the identity matrix of the given dimension.

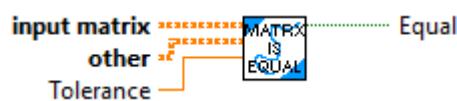
Inputs:

- size -- The dimension of the desired matrix.

Outputs:

- output matrix -- The DxD identity matrix.
- error -- If TRUE, an error occurred.

Matrix_IsEqual



Checks if another {@link Matrix} is equal to "this" within a specified tolerance.

This will check if each element is in tolerance of the corresponding element from the other Matrix.

Inputs:

- input matrix -- The input matrix
- other -- The matrix to check against this one.
- tolerance -- The tolerance to check equality with. (Default is 1E-6)

Outputs:

- equal -- true if this matrix is equal to the one supplied.

Matrix_LDLT_Decomposition



This VI computes the square root free Cholesky factorization

$$A = L \cdot D \cdot L'$$

Where L is a lower triangular matrix with ones on the diagonal, and D is a diagonal matrix.

It is assumed that A is symmetric and positive definite.

Reference: Golub and Van Loan, "Matrix Computations", second edition, p 137.

it is based in the work made by Brian Borchers (borchers@nmt.edu) in MATLAB.

Created by Davis Montenegro

10-05-2013

Inputs:

- Matrix -- Matrix to decompose

Outputs:

- L -- L matrix

- D -- D matrix
-

Matrix_LltDecompose



Decompose "this" matrix using Cholesky Decomposition. If the "this" matrix is zeros, it will return the zero matrix.

Inputs:

- input matrix -- The input matrix
- lowerTriangular -- Whether or not we want to decompose to the lower triangular Cholesky matrix.

Outputs:

- output Cholesky Decomp -- The decomposed matrix.
 - error -- If TRUE, an error occurred. For example if the matrix could not be decomposed(ie. is not positive semidefinite).
-

Matrix_NormF



Computes the Frobenius normal of the matrix.

$$\text{normF} = \text{Sqrt}\{ \text{sum; } i=1: m \text{ sum; } j=1:n (a[i,j]^2) \}$$

Inputs:

- input matrix -- The input matrix

Outputs:

- NormF -- Double containing the frobenius normal value of this matrix
 - Error -- If TRUE, an error occurred.
-

Matrix_Pow



Computes the matrix power using Eigen's solver. This method only works for square matrices, and will otherwise throw an MatrixDimensionException.

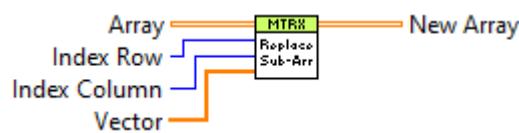
Inputs:

- input matrix -- The input matrix
- exponent -- The exponent.

Outputs:

- output matrix -- The exponential of A.
 - error -- If TRUE, an error occurred.
 - ExponentNotRational -- TRUE if exponent is not a rational number.
-

Matrix_Replace_Sub_array



This VI is used to replace an array inside a matrix where the array has a smaller size than the rows of the matrix.

Created by Davis Montenegro

10-05-2013

Matrix_SetColumn



Sets a column to a given column vector.

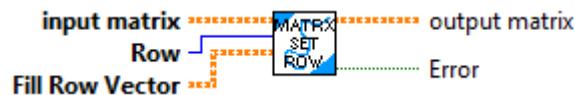
Inputs:

- input matrix -- The input matrix
- column -- The column to set.
- val -- The column vector to set the given row to.

Outputs:

- output matrix -- the resulting matrix
- error -- If TRUE, an error occurred.

Matrix_SetRow



Sets a row to a given row vector.

Inputs:

- input matrix -- The input matrix
- row -- The row to set.

- val -- The row vector to set the given row to.

Outputs:

- output matrix -- resulting matrix
- error -- If TRUE, an error occurred.

Matrix_Transpose



Transpose matrix

Inputs:

- input matrix -- The input matrix

Outputs:

- output matrix -- The transposed matrix

Matrix_WithinTolerance



Determines if all the values are within the given tolerance.

Tolerance is defined as:

```

IF ( MAX( ABS(A), ABS(B) ) ) > 1E-20 THEN
    ABS( A - B ) / MAX( ABS(A), ABS(B) )
ELSE
  
```

TRUE

END IF

Inputs:

- Matrix A -- First input matrix
- Matrix B -- Second input matrix
- Tolerance -- Tolerance value, optional (default: 1.0E-9)

Outputs:

- Within Tolerance -- Set to true if all values are within tolerance.
- Error -- Returns TRUE if an error occurred (matrices are not the same size.)

MatrixHelper

MatrixHelper_CoerceSize



Ensure a matrix is the size that is needed.

Inputs:

- Input Matrix -- The matrix to be checked and resized if needed
- Rows -- The expected number of rows
- Columns -- The expected number of columns

Outputs:

- Output Matrix -- The potentially resized matrix
- Coerced -- If TRUE, the matrix size was modified.

MatrixHelper_MultCoerceBSize



Ensure that the B matrix is the correct size before performing the matrix multiplication A x B.

Inputs:

- A -- A matrix
- B -- B matrix

Outputs:

- Updated B -- B matrix, resized as necessary to multiply with A.
 - Coerced -- If TRUE, the size of B was changed.
-

MatrixHelper_Zero



Zero the contents a a matrix

Inputs:

- Input Matrix -- The matrix to zero.

Outputs:

- Output Matrix -- The zeroed matrix.

MaxVelocityConstraint

MaxVelocityConstraint_New



Constructs a maximum velocity constraint.

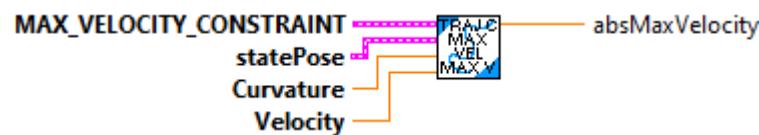
Parameters:

- MaxVelocity - Maximum velocity

Returns

- MaxVelocityConstraint - Constraint data structure.

MaxVelocityConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

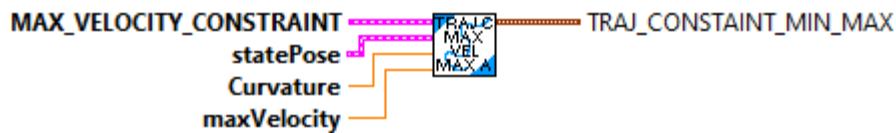
Parameters:

- MaxVelocityConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

MaxVelocityConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

Parameters:

- MaxVelocityConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

MecaDriveKinematicsConstraint

MecaDriveKinematicsConstraint_New



Constructs a mecanum drive dynamics constraint.

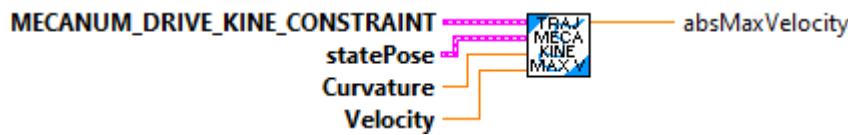
Parameters:

- maxVelocity - Maximum allowed velocity.
- MecanumDriveKinematics - Data structure

Returns

- MecanumDriveKinematicsConstraint - Constraint data structure

MecaDriveKinematicsConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

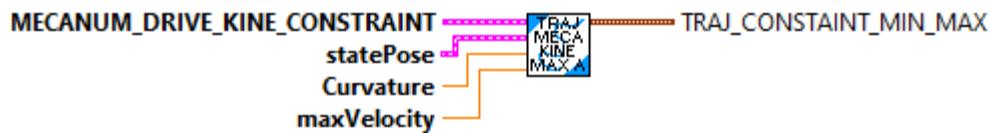
Parameters:

- MecanumDriveKinematicsConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

MecaDriveKinematicsConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values.

Parameters:

- MecanumDriveKinematicsConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

MecaDrivePoseEst

MecaDrivePoseEst_AddVisionMeasurement



Add a vision measurement to the Unscented Kalman Filter. This will correct the odometry pose estimate while still accounting for measurement noise.

This method can be called as infrequently as you want, as long as you are calling `MecaDrivePoseEstimator_update` every loop.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

Inputs:

- `MecaPoseEst` -- Data cluster containing `SwervePoseEst` data
- `visionRobotPoseMeters` -- The pose of the robot as measured by the vision camera.
- `timestampSeconds` -- The timestamp of the vision measurement in seconds. Note that if you don't use your own time source by calling `SwerveDrivePoseEstimator_updateWithTime` then you must use a timestamp with an epoch since FPGA startup (i.e. the epoch of this timestamp is the same epoch as `Timer.getFPGATimestamp()`) This means that you should use `Timer.getFPGATimestamp` as your time source or sync the epochs.

Outputs:

- `outMecaPoseEst` -- Updated data cluster

- Error -- If TRUE, an error occurred.

MecaDrivePoseEst_GetEstimatedPosition



Gets the pose of the robot at the current time as estimated by the Unscented Kalman Filter.

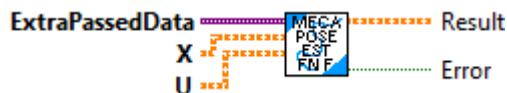
Inputs:

- MecaPoseEst -- Data cluster containing SwervePoseEst data

Outputs:

- EstimatedPose - The estimated robot pose in meters.

MecaDrivePoseEst_Kalman_F_Callback



Meca Drive Pose Estimator, Kalman Filter F function.

This function returns the U matrix.

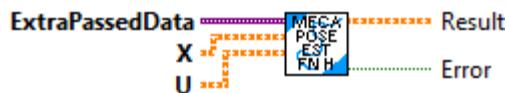
Input:

- ExtraPassedData -- Variant containing extra data for the callback. For this function the extra data should be empty..
- X Matrix -- X matrix
- U Matrix -- U matrix

Output:

- Result -- Matrix resulting from calculation
 - Error -- If TRUE an error occurred
-

MecaDrivePoseEst_Kalman_H_Callback



Mecanum Drive Pose Estimator, Kalman Filter H function.

This function returns row 2 of the X matrix.

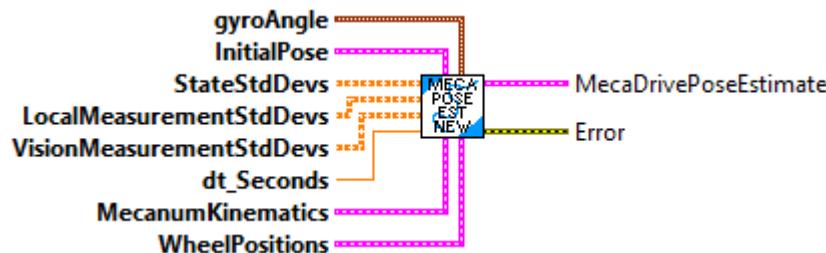
Input:

- ExtraPassedData -- Variant containing extra data for the callback. For this function the extra data should be empty..
- X Matrix -- X matrix
- U Matrix -- U matrix

Output:

- Result -- Matrix resulting from calculation
 - Error -- If TRUE an error occurred
-

MecaDrivePoseEst_New



Constructs a MecaDrivePoseEstimator.

This data cluster and its associated function blocks wrap an UnscentedKalmanFilter Unscented Kalman Filter to fuse latency-compensated vision measurements with mecanum drive encoder velocity measurements. It will correct for noisy measurements and encoder drift. It is intended to be an easy but more accurate drop-in for MecanumDriveOdometry.

MecanumDrivePoseEstimator_update should be called every robot loop. If your loops are faster or slower than the default of 20 ms, then you should change the nominal delta time using the secondary constructor: MecanumDrivePoseEstimator_MecanumDrivePoseEstimator(Rotation2d, Pose2d MecanumDriveWheelPositions, MecanumDriveKinematics, Matrix, Matrix, Matrix, double).

MecanumDrivePoseEstimator_addVisionMeasurement can be called as infrequently as you want; if you never call it, then this data cluster will behave mostly like regular encoder odometry.

The state-space system used internally has the following states (x), inputs (u), and outputs (y):

$$x = [x, y, \theta, s_{fl}, s_{fr}, s_{rl}, s_{rr}]^T$$

in the field coordinate system containing x position, y position, and heading, followed by the distance driven by the front left, front right, rear left, and rear right wheels.

$$u = [v_x, v_y, \omega, v_{fl}, v_{fr}, v_{rl}, v_{rr}]^T$$

containing x velocity, y velocity, and angular rate in the field coordinate system, followed by the velocity of the front left, front right, rear left, and rear right wheels.

$$y = [x, y, \theta]^T$$

from vision containing x position, y position, and heading; or

$$y = [\theta, s_{fl}, s_{fr}, s_{rl}, s_{rr}]^T$$

containing gyro heading, followed by the distance driven by the front left, front right, rear left, and rear right wheels.

Inputs:

- gyroAngle -- The current gyro angle.
- initialPoseMeters -- The starting pose estimate.
- wheelPositions -- The distances driven by each wheel.
- kinematics -- A correctly-configured kinematics object for your drivetrain.
- stateStdDevs -- Standard deviations of model states. Increase these numbers
 - to trust your model's state estimates less. This matrix is in the form $[x, y, \theta, s_{fl}, s_{fr}, s_{rl}, s_{rr}]^T$, with units in meters and radians, followed by meters.
- localMeasurementStdDevs -- Standard deviation of the gyro measurement.
 - Increase this number to trust sensor readings from the gyro less.
 - This matrix is in the form $[\theta, s_{fl}, s_{fr}, s_{rl}, s_{rr}]$, with units in radians, followed by meters.
- visionMeasurementStdDevs -- Standard deviations of the vision measurements.
 - Increase these numbers to trust global measurements from vision less.
 - This matrix is in the form $[x, y, \theta]^T$, with units in meters and radians.
- nominalDtSeconds -- The time in seconds between each robot loop.

Outputs:

- outMecaPoseEst -- Updated data cluster
 - Error -- If TRUE, an error occurred.
-

MecaDrivePoseEst_ResetPosition



Resets the robot's position on the field.

(NOTE -- For LabVIEW version, this may not be needed.) You NEED to reset your encoders (to zero) when calling this method.

The gyroscope angle does not need to be reset in the user's robot code. The library automatically takes care of offsetting the gyro angle.

Inputs:

- MecaPoseEst -- Data cluster containing SwervePoseEst data
- poseMeters -- The position on the field that your robot is at.
- gyroAngle -- The angle reported by the gyroscope.
- wheelPositions -- The distances driven by each wheel.

Outputs:

- outMecaPoseEst -- Updated data cluster

MecaDrivePoseEst_SetVisionMeasurementStdDevs



Sets the pose estimator's trust of global measurements. This might be used to change trust in vision measurements after the autonomous period, or to change trust as distance to a vision target increases.

Inputs:

- MecaPoseEst -- Data cluster containing MecaPoseEst data

- visionMeasurementStdDevs -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form

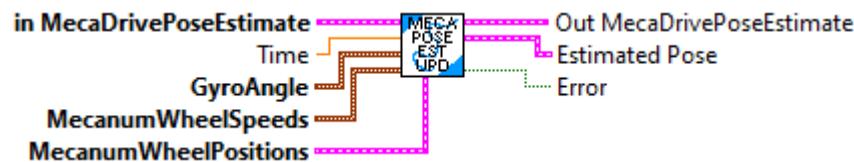
[x, y, theta]?,

with units in meters and radians.

Outputs:

- outMecaPoseEst -- Updated data cluster
- SizeCoerced -- If TRUE, an unexpected error occurred. Execution may continue.

MecaDrivePoseEst_Update



Updates the the Unscented Kalman Filter using only wheel encoder information. This should be called every loop, and the correct loop period must be passed into the constructor of this class.

Inputs:

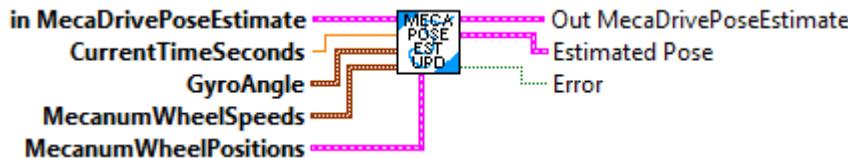
- In Meca Pose Est -- Input data cluster.
- gyroAngle -- The current gyro angle.
- wheelSpeeds -- The current velocities wheels.
- wheelPositions -- The distances driven by each wheel.
- currentTimeSeconds -- (Optional) Time at which this method was called, in seconds.

Outputs:

- outMecaPoseEst -- Updated data cluster
- EstimatedPose -- The estimated pose of the robot in meters.

- Error -- If TRUE, an error occurred.

MecaDrivePoseEst_UpdateWithTime



Updates the the Unscented Kalman Filter using only wheel encoder information. This should be called every loop, and the correct loop period must be passed into the constructor of this class.

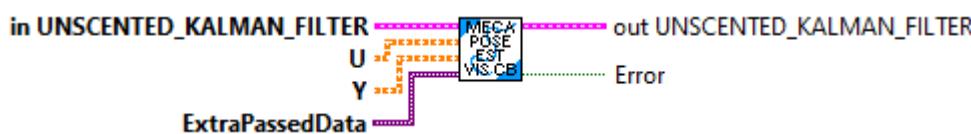
Inputs:

- MecaPoseEst -- Data cluster containing SwervePoseEst data
- currentTimeSeconds -- Time at which this method was called, in seconds.
- gyroAngle -- The current gyroscope angle.
- WheelSpeeds -- The current wheel velocities.
- wheelPositions -- The distances driven by each wheel.

Outputs:

- outMecaPoseEst -- Updated data cluster
- EstimatedPose -- The estimated pose of the robot in meters.
- Error -- If TRUE, an error occurred.

MecaDrivePoseEst_VisionCorrect_Callback



Mecanum Drive Pose Estimator function used to update Kalman Filter.

This function calls the UnscentedKalmanFIIter Correct function with U, Y, VisionContR matrices.

The parameters of this function are fixed since it's reference is passed to other functions. This is why the VisionContR matrix needs to be passed as extra data.

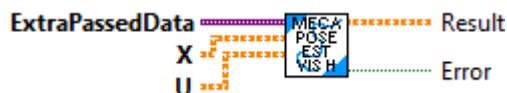
Inputs:

- Unscented Kalman FIIter -- Input data cluster
- U -- U matrix
- Y -- Y matrix
- ExtraData -- Variant containing extra data. For this call the extra data must contain the VisionContR Matrix.

Outputs:

- Out Unscented Kalman FIIter -- Updated data cluster

MecaDrivePoseEst_VisionCorrect_Kalman_H_Callback



Mecanum Drive Pose Estimator, Kalman Filter H function for vision correction. This function passes the X matrix to the resulting matrix. No calculations are done.

Input:

- ExtraPassedData -- Variant containing extra data for the callback. For this function the extra data should be empty..

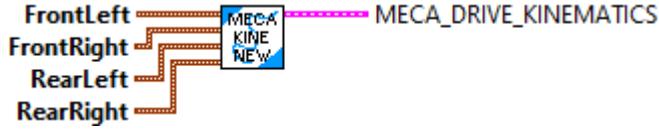
- X Matrix -- X matrix
- U Matrix -- U matrix

Output:

- Result -- Matrix resulting from calculation
- Error -- If TRUE an error occurred

MecaKinematics

MecaKinematics_New



Constructs a MecanumDriveKinematics data structure.

Helper class that converts a chassis velocity (dx , dy , and $d\theta$ components) into individual wheel speeds.

The inverse kinematics (converting from a desired chassis velocity to individual wheel speeds) uses the relative locations of the wheels with respect to the center of rotation. The center of rotation for inverse kinematics is also variable. This means that you can set your center of rotation in a corner of the robot to perform special evasion manuevers.

Forward kinematics (converting an array of wheel speeds into the overall chassis motion) is performs the exact opposite of what inverse kinematics does. Since this is an overdetermined system (more equations than variables), we use a least-squares approximation.

The inverse kinematics: $[wheelSpeeds] = [wheelLocations][chassisSpeeds]$ We take the Moore-Penrose pseudoinverse of $[wheelLocations]$ and then multiply by $[wheelSpeeds]$ to get our chassis speeds.

Forward kinematics is also used for odometry -- determining the position of the robot on the field using encoders and a gyro.

Parameters:

- Left Front - The location of the front-left wheel relative to the physical center of the robot. (Translation)
- Right Front - The location of the front-right wheel relative to the physical center of the robot. (Translation)
- Left Rear - The location of the rear-left wheel relative to the physical center of the robot. (Translation)
- Right Rear - The location of the rear-right wheel relative to the physical center of the robot. (Translation)

Returns:

- Meca_Drive_Kinematics - Data structure

MecaKinematics_SetInverseKinematics



Construct inverse kinematics matrix from wheel locations.

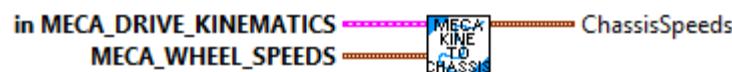
Parameters:

- FrontLeft - The location of the front-left wheel relative to the physical center of the robot.
- FrontRight - The location of the front-right wheel relative to the physical center of the robot.
- RearLeft - The location of the rear-left wheel relative to the physical center of the robot.
- RearRight - The location of the rear-right wheel relative to the physical center of the robot.

Returns:

- InverseKinematics - Return matix
-

MecaKinematics_ToChassisSpeeds



Performs forward kinematics to return the resulting chassis state from the given wheel speeds. This method is often used for odometry -- determining the robot's position on the field using data from the real-world speed of each wheel on the robot.

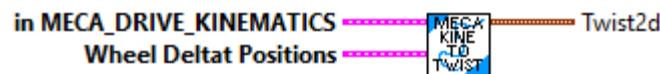
Parametetrs:

- MecaDriveKinematics - Data structure
- WheelSpeeds - The current mecanum drive wheel speeds.

Returns:

- ChassisSpeeds - The resulting chassis speed.

MecaKinematics_ToTwist2d



Performs forward kinematics to return the resulting Twist2d from the given wheel deltas. This method is often used for odometry -- determining the robot's position on the field using changes in the distance driven by each wheel on the robot.

- * @param
- * @return The resulting Twist2d.

Parameters:

- MecaDriveKinematics -- Data structure
- wheelDeltas -- The distances driven by each wheel.

Returns:

- Twist2d - The resulting twist.

MecaKinematics_ToWheelSpeeds



Performs inverse kinematics to return the wheel speeds from a desired chassis velocity. This method is often used to convert joystick values into wheel speeds.

This function also supports variable centers of rotation. During normal operations, the center of rotation is usually the same as the physical center of the robot; therefore, the argument is defaulted to that use case. However, if you wish to change the center of rotation for evasive maneuvers, vision alignment, or for any other use case, you can do so.

Parameters:

- MecaDriveKinematics - Data structure
- chassisSpeeds - The desired chassis speed.

- centerOfRotation - The center of rotation. For example, if you set the center of rotation at one corner of the robot and provide a chassis speed that only has a dtheta component, the robot will rotate around that corner. (Meters)

Returns:

- MecaDriveKinematics - Updated data structure

- WheelSpeeds - The wheel speeds. Use caution because they are not normalized. Sometimes, a user input may cause one of the wheel speeds to go above the attainable max velocity. Use the MecaWheel_normalize function to rectify this issue.

MecaKinematics_ToWheelSpeedsZeroCenter



Performs inverse kinematics to return the wheel speeds from a desired chassis velocity. This method is often used to convert joystick values into wheel speeds.

During normal operations, the center of rotation is usually the same as the physical center of the robot. This subVI, uses the physical center of the robot. Use MecaKinematics_ToWheelSpeeds to use a different center of rotation.

Parameters:

- MecaDriveKinematics - Data structure
- chassisSpeeds - The desired chassis speed.

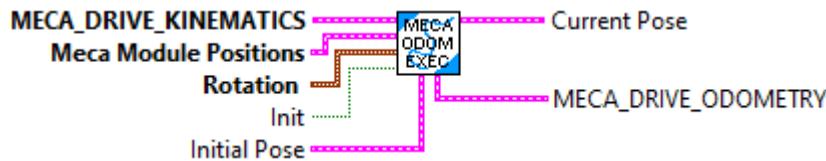
Returns:

- MecaDriveKinematics - Updated data structure

- WheelSpeeds - The wheel speeds. Use caution because they are not normalized. Sometimes, a user input may cause one of the wheel speeds to go above the attainable max velocity. Use the MecaWheel_normalize function to rectify this issue.

MecaOdometry

MecaOdometry_Execute



SIngle call LabVIEW function to execute Mecanum Drive Odometry to calculate a relative or absolute POSE.

Inputs:

- Mecanum_Drive_Kinematics -- cluster -- mecanum drive kinematics
- Mecanum Wheel Positions -- array of cluster -- current wheel positions (distances).
- Gyro Value -- rotation2d -- current gyro reading.
- Init -- boolean -- Reset position to the value of Initial Pose. Odometry is calculated from this position.
- Initial Pose -- Pose2d -- Pose to use when Init is TRUE. (Optional. Default: 0,0,0)

Returns

- Current Pose -- Pose2d -- Current position calculated by Mecanum Odometry
- Mecanum_Drive_Odometry -- cluster -- current value of the mecanum drive odometry cluster.

MecaOdometry_GetKinematics



Returns the kinematics data cluster for this odometry

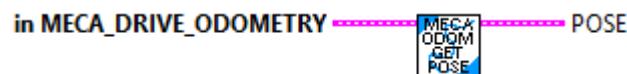
Parameters:

- MecaDriveOdometry - Data structure

Returns:

Mecanum Kinematics - The kinematics data cluster

MecaOdometry_GetPose



Returns the position of the robot on the field.

Parameters:

- MecaDriveOdometry - Data structure

Returns:

Pose - The pose of the robot (x and y are in meters).

MecaOdometry_New



Constructs a MecanumDriveOdometry cluster data structure

SubVIs for mecanum drive odometry. Odometry allows you to track the robot's position on the field over a course of a match using readings from your mecanum wheel encoders.

Teams can use odometry during the autonomous period for complex tasks like path following. Furthermore, odometry can be used for latency compensation when using computer-vision systems.

Parameters:

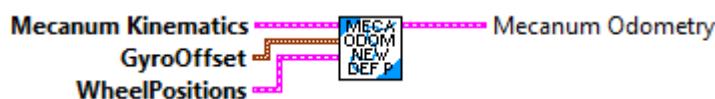
- Mecanum Kinematics -- The mechanum kinematics data cluster for this drive

- GyroOffset - The angle reported by the gyroscope. (Rotation)
- wheelPositions -- The distances driven by each wheel.
- initialPose - The starting position of the robot on the field. (Pose - Meters)

Returns:

- MecaDriveOdometry - Data structure

MecaOdometry_NewDefaultPose



Constructs a MecanumDriveOdometry cluster data structure with a default pose at the origin.

SubVIs for mecanum drive odometry. Odometry allows you to track the robot's position on the field over a course of a match using readings from your mecanum wheel encoders.

Teams can use odometry during the autonomous period for complex tasks like path following. Furthermore, odometry can be used for latency compensation when using computer-vision systems.

Parameters:

- MecanumKinematics -- Mecanum Kinematics data cluster
- GyroOffset - The angle reported by the gyroscope. (Rotation)
- wheelPositions -- The distances driven by each wheel.

Returns:

- MecaDriveOdometry - Data structure

MecaOdometry_Reset



Resets the robot's position on the field.

The gyroscope angle does not need to be reset here on the user's robot code. The library automatically takes care of offsetting the gyro angle.

Parameters

- MecaDriveOdometry - Data structure
- gyroOffset - The angle reported by the gyroscope.
- wheelPositions -- The distances driven by each wheel.
- pose - The position on the field that your robot is at. (Meters)

Returns:

- MecaDriveOdometry - Updated ata structure

MecaOdometry_Update



Updates the robot's position on the field using forward kinematics and integration of the pose over time. This method takes in an angle parameter which is used instead of the angular rate that is calculated from forward kinematics, in addition to the current distance measurement at each wheel.

Parameters:

- MecaDriveOdometry - Data structure
- gyroAngle - The angle reported by the gyroscope.
- wheelPositions - The distances driven by each wheel..

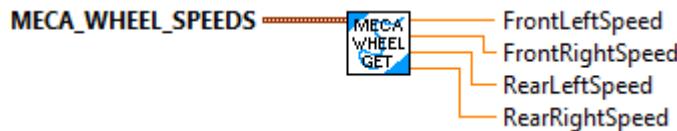
Returns:

- MecaDriveOdometry - Updated data structure

- Pose - The new pose of the robot.

MecaWheel

MecaWheel_GetAll



Returns the individual wheel speeds from a MecanumDriveWheelSpeeds data structure.

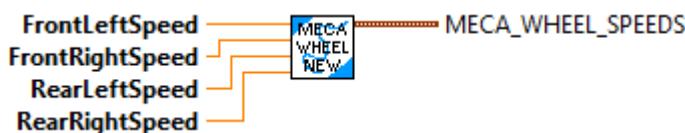
Parameters:

- Meca_wheel_speeds - Data structure

Returns:

- Left Front Speed - Speed of left front wheel (meters/sec)
- Right Front Speed - Speed of right front wheel (meters/sec)
- Left Rear Speed - Speed of left rear wheel (meters/sec)
- Right Rear Speed - Speed of right rear wheel (meters/sec)

MecaWheel_New



Constructs a MecanumDriveWheelSpeeds data structure.

Parameters:

- Left Front Speed - Speed of left front wheel (meters/sec)
- Right Front Speed - Speed of right front wheel (meters/sec)
- Left Rear Speed - Speed of left rear wheel (meters/sec)
- Right Rear Speed - Speed of right rear wheel (meters/sec)

Returns:

- Meca_wheel_speeds - Data structure
 - Normalized Right Wheel Demand
-

MecaWheel_normalize



Normalizes the wheel speeds using some max attainable speed. Sometimes, after inverse kinematics, the requested speed from a/several modules may be above the max attainable speed for the driving motor on that module. To fix this issue, one can "normalize" all the wheel speeds to make sure that all requested module speeds are below the absolute threshold, while maintaining the ratio of speeds between modules.

Parameters:

- in Meca_wheel_speeds - Data structure
- MaxAttainableSpeed - The absolute max speed that a wheel can reach.

Returns:

- out Meca_wheel_speeds - Data structure

MecaWheelPos

MecaWheelPos_Get



Extract individual positions (distances) from a MecanumDriveWheelPosition data structure.

Parameters:

- Meca_wheel_Position- Data structure

Returns:

- Left Front Position - Distance measured by left front wheel (meters)
- Right Front Position - Distance measured by right front wheel (meters)
- Left Rear Position - Distance measured by left rear wheel (meters)
- Right Rear Position - Distance measured by right rear wheel (meters)

MecaWheelPos_New



Constructs a MecanumDriveWheelPosition data structure.

Parameters:

- Left Front Position - Distance measured by left front wheel (meters)
- Right Front Position - Distance measured by right front wheel (meters)
- Left Rear Position - Distance measured by left rear wheel (meters)
- Right Rear Position - Distance measured by right rear wheel (meters)

Returns:

- Meca_wheel_Position- Data structure
-

MecaWheelPos_Sub



Subtract two MecanumDriveWheelPosition data structures to create a delta MecanumDriveWheelPosition

Returns MecaWheelPos1 - MecaWheelPos2

Parameters:

- in Meca_wheel_Position1- Data structure
- in Meca_wheel_Position2- Data structure

Returns:

- Meca_wheel_Position- Data structure

MedianFilter

MedianFilter_Calculate



Calculates the moving-window median for the next value of the input stream.

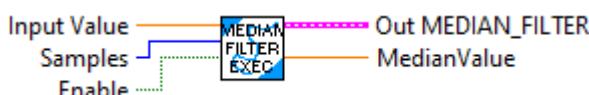
Inputs:

- In Median Filter -- Median filter data structure
- Input Value -- The next input value.

Outputs:

- Out Median Filter -- Updated median filter data structure
- Median Value -- The median of the moving window, updated to include the next value..

MedianFilter_Execute



Convenience, single call, LabVIEW function. Creates and calculates the moving-window median for the next value of the input stream.

The optional enable parameter bypasses the filter calculation when "enabled" is false. When enabled is "false" the internal buffer is continually filled with the current input value.

This implements a moving-window median filter. Useful for reducing measurement noise, especially with processes that generate occasional, extreme outliers (such as values from vision processing, LIDAR, or ultrasonic sensors).

Inputs:

- Input Value -- The next input value.
- samples -- The number of samples in the moving window.

- enable -- Perform the filter calcuation when True. When False, output the input value unchanged. (Default=True)

Outputs:

- Out Median Filter -- Updated median filter data structure
 - Median Value -- The median of the moving window, updated to include the next value..
-
-

MedianFilter_New



Creates a new MedianFilter data cluster.

This implements a moving-window median filter. Useful for reducing measurement noise, especially with processes that generate occasional, extreme outliers (such as values from vision processing, LIDAR, or ultrasonic sensors).

Inputs:

- samples -- The number of samples in the moving window.

Outputs:

- Median Filter -- Median filter data structure
-
-

MedianFilter_Reset



Resets the filter, clearing the window of all elements. All stored values are set to zero.

Inputs:

- In Median Filter -- Median filter data structure

Outputs:

- Out Median Filter -- Updated median filter data structure
-

MedianFilter_ResetToValue



Resets the filter, clearing the window of all elements. All stored values are set to "value:".

Inputs:

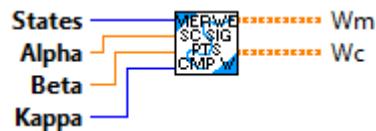
- Value -- Storage buffer elements are set to this value.
- In Median Filter -- Median filter data structure

Outputs:

- Out Median Filter -- Updated median filter data structure

MerweScSigPts

MerweScSigPts_ComputeWeights



Computes the weights for the scaled unscented Kalman filter.

Inputs:

- States -- Number of states
- Alpha --
- beta -- Incorporates prior knowledge of the distribution of the mean.
- Kapa --

Outputs:

- Wm --

MerweScSigPts_GetNumSigmas



Returns number of sigma points for each variable in the state x.

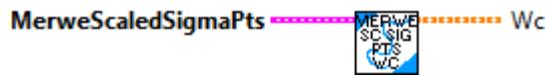
Inputs:

- MerweScaledSigmaPts -- data cluster

Outputs:

- NumSigmas -- The number of sigma points for each variable in the state x.

MerweScSigPts_GetWc



Returns the weight for each sigma point for the covariance.

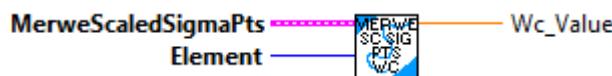
Inputs:

- MerweScaledSigmaPts -- data cluster

Outputs:

- Wc -- the weight for each sigma point for the covariance.

MerweScSigPts_GetWc_Single



Returns an element of the weight for each sigma point for the covariance.

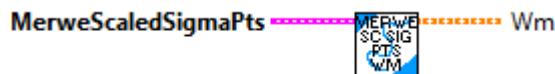
Inputs:

- MerweScaledSigmaPts -- data cluster
- element -- Element of vector to return.

Outputs:

- Wc_Value -- The element I's weight for the covariance.

MerweScSigPts_GetWm



Returns the weight for each sigma point for the mean.

Inputs:

- MerweScaledSigmaPts -- data cluster

Outputs:

- Wm -- the weight for each sigma point for the mean.

MerweScSigPts_GetWm_Single



Returns an element of the weight for each sigma point for the mean.

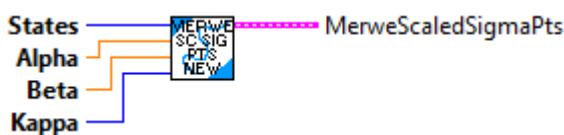
Inputs:

- MerweScaledSigmaPts -- data cluster
- element -- Element of vector to return.

Outputs:

- Wm_Value -- the element i's weight for the mean.

MerweScSigPts_New



Generates sigma points and weights according to Van der Merwe's 2004 dissertation[1] for the UnscentedKalmanFilter class.

It parametrizes the sigma points using alpha, beta, kappa terms, and is the version seen in most publications. Unless you know better, this should be your default choice.

States is the dimensionality of the state. $2 * \text{States} + 1$ weights will be generated.

[1] R. Van der Merwe "Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models" (Doctoral dissertation)

Constructs a generator for Van der Merwe scaled sigma points.

Inputs:

- states -- the number of states.
- alpha -- Determines the spread of the sigma points around the mean. Usually a small positive value (1e-3).
- beta -- Incorporates prior knowledge of the distribution of the mean. For Gaussian distributions, beta = 2 is optimal.
- kappa -- Secondary scaling parameter usually set to 0 or 3 - States.

Outputs:

- MerweScaledSigmaPts -- data cluster

MerweScSigPts_New_Default



Generates sigma points and weights according to Van der Merwe's 2004 dissertation[1] for the UnscentedKalmanFilter class.

It parametrizes the sigma points using alpha, beta, kappa terms, and is the version seen in most publications. Unless you know better, this should be your default choice.

States is the dimensionality of the state. $2*\text{States}+1$ weights will be generated.

[1] R. Van der Merwe "Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models" (Doctoral dissertation)

Constructs a generator for Van der Merwe scaled sigma points with default values for alpha, beta, and kappa.

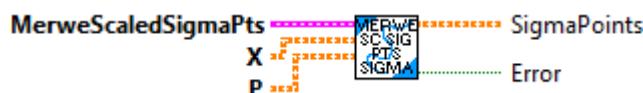
Inputs:

- states -- the number of states.

Outputs:

- MerweScaledSigmaPts -- data cluster

MerweScSigPts_SigmaPoints



Computes the sigma points for an unscented Kalman filter given the mean (x) and covariance(P) of the filter.

Inputs:

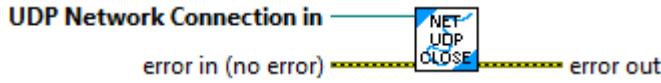
- MerweScaledSigmaPts -- data cluster
- x -- An array of the means.
- P -- Covariance of the filter.

Outputs:

- SigmaPoints -- Two dimensional array of sigma points. Each column contains all of the sigmas for one dimension in the problem space. Ordered by $X_i_0, X_i_{\{1..n\}}, X_i_{\{n+1..2n\}}$.
- Error -- If TRUE, an error occurred.

NetworkUDP

NetworkUDP_Close



Close an open UDP connection.

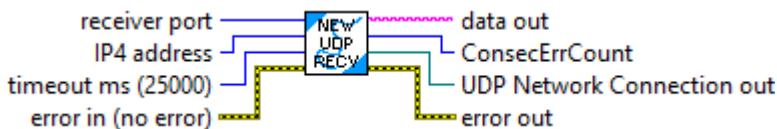
Inputs

- UDP Network Connection -- UDP Connection Reference -- The connection to close
- Error in -- Error cluster -- input propagated error. (Optional. Default: No error)

Outputs:

- Error out -- Error cluster -- output propagated error.

NetworkUDP_Receive



Listen for and read a packet of data from the network using the UDP protocol. This routine can listen for a single computer or listen for any senders on the subnet.

This subVI does not return to the caller until either data has been received or a timeout has occurred. Suggest putting this function in a separate loop to avoid issues slowing down other code.

Inputs:

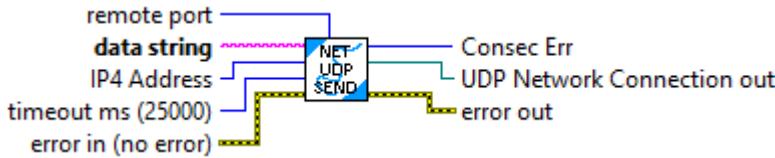
- Receiver Port -- UInt16 -- UDP port number used for listening. Only data sent to this port will be returned. (Optional. Default: 5801)
- IP4 Address -- UInt32 -- The IP address of the computer expected to send data. If this value is 0, then data will be returned for any sender. Use the standard TCP/IP LabVIEW functions to convert a standard IP4 address in the form of 123.123.123.123 to an unsigned integer for this input. (Optional. Default: 0)
- Timeout ms -- Int32 -- Timeout value in milliseconds. (Optional. Default: 25000)

-- Error in -- Error cluster -- Propagated error cluster. (Optional. Default: no error)

Outputs:

- data out -- string -- Last data message sent to the specified port number.
 - Consec Err -- Int32 -- Counter indicating the number of consecutive errors that have occurred.
 - UDP Network Connection -- UDP Connection Reference -- Reference to the UDP network connection. When finished with communication, the connection should be closed.
 - Error out -- Error cluster -- Current propagated error cluster.
-

NetworkUDP_Send



Send a packet of data over the network using the UDP protocol. Data can be directed at a single computer or broadcast to the entire subnet.

Inputs:

- Remote Port -- UInt16 -- UDP port number to use when sending the data. The receiver must be listening for data on this port number. (Optional. Default: 5801)
- Data String -- string -- string of data to be sent. (Generally the maximum reliable size of a UDP packet is approximately 500 bytes. On Ethernet, packets of approximately 1450 bytes may be possible.)
- IP4 Address -- UInt32 -- The IP address of the computer listening for this data. If this value is 0, then the data is broadcast to the entire subnet. Use the standard TCP/IP LabVIEW functions to convert a standard IP4 address in the form of 123.123.123.123 to an unsigned integer for this input. (Optional. Default: 0)
- Timeout ms -- Int32 -- Timeout value in milliseconds. (Optional. Default: 1000)
- Error in -- Error cluster -- Propagated error cluster. (Optional. Default: no error)

Outputs:

- Consec Err -- Int32 -- Counter indicating the number of consecutive errors that have occurred.

-- UDP Network Connection -- UDP Connection Reference -- Reference to the UDP network connection. When finished with communication, the connection should be closed.

-- Error out -- Error cluster -- Current propagated error cluster.

numCmd

numCmd_ObtainQueue_Array

Command Name  queue out
error out

numCmd_ObtainQueue_Generic

Command Name  queue out
error out

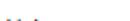
numCmd_ObtainQueue_OneDbl

Command Name  queue out
error out

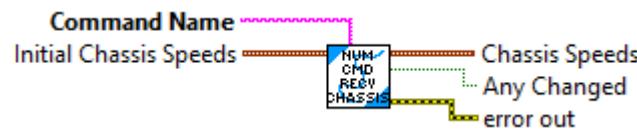
numCmd_ObtainQueue_TwoDbl

Command Name  queue out
error out

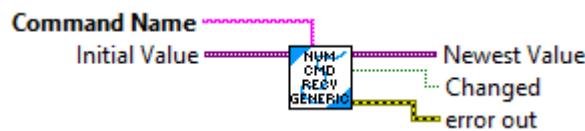
numCmd_Recv_Array

Command Name 
Initial Values  Values 
Any Changed  error out

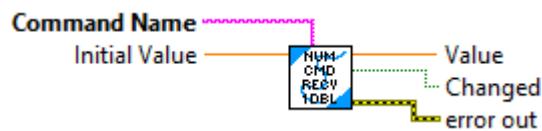
numCmd_Recv_Chassis



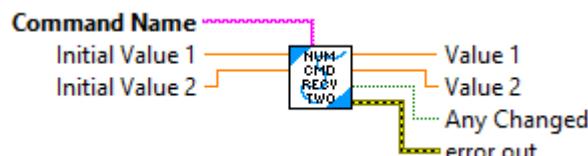
numCmd_Recv_Generic



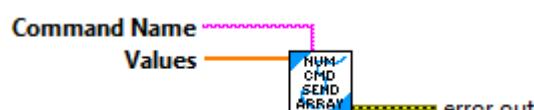
numCmd_Recv_OneDbl



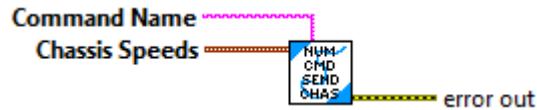
numCmd_Recv_TwoDbl



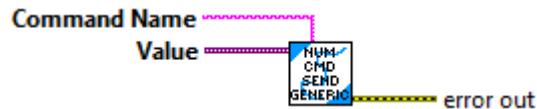
numCmd_Send_Array



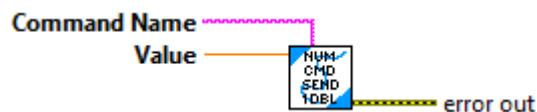
numCmd_Send_Chassis



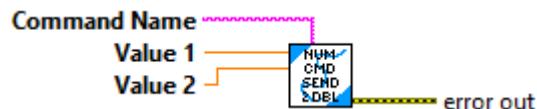
numCmd_Send_Generic



numCmd_Send_OneDbl

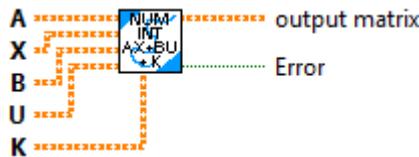


numCmd_Send_TwoDbl



NumIntegrate

NumIntegrate_Func_Ax_Bu_K



Performs the matrix equation

$$\text{result} = A \times X + B \times U + K$$

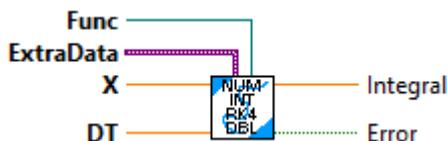
Inputs:

- A -- Input A matrix
- X -- input X matrix
- B -- input B matrix
- U -- input U matrix
- K - input K matrix

Outputs:

- Output Matrix -- Result of calculation
- Error -- If TRUE an error occurred.

NumIntegrate_Rk4_Dbl_X



Performs Runge Kutta integration (4th order).

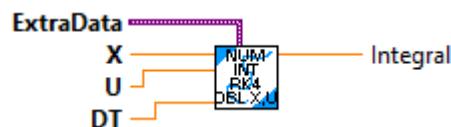
Inputs:

- Func -- The function to integrate, which takes one argument x.
- ExtraData -- Variant containing extra data, if any, needed by Func.
- x -- The initial value of x.
- dt - The time, in seconds, over which to integrate.

Outputs:

- Integral -- the integration of $dx/dt = f(x)$ for dt.
- Error -- If TRUE, an error occurred.

NumIntegrate_Rk4_Dbl_X_U



Performs Runge Kutta integration (4th order).

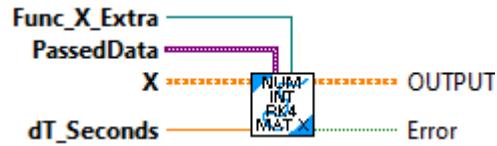
Inputs:

- Func -- The function to integrate. It must take two arguments x and u.
- ExtraData -- Variant containing extra data, if any, needed by Func.
- x -- The initial value of x.
- u -- The value u held constant over the integration period.
- dt - The time, in seconds, over which to integrate.

Outputs:

- Integral -- the integration of $dx/dt = f(x)$ for dt.
- Error -- If TRUE, an error occurred.

NumIntegrate_Rk4_Mat_X



Performs 4th order Runge-Kutta integration of $dx/dt = f(\text{matrix } x)$ for dt .

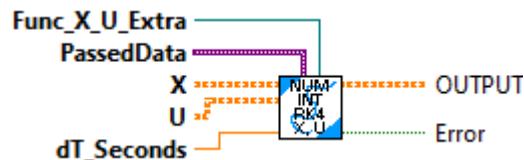
Inputs:

- Func_X_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix X, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- X -- X input matrix
- dT -- Time differential in seconds

Outputs:

- OUTPUT -- The integration of $dx/dt = f(x, u)$ for dt .
- Error -- If TRUE, an error occurred.

NumIntegrate_Rk4_Mat_X_U



Performs 4th order Runge-Kutta integration of $dx/dt = f(\text{matrix } x, \text{matrix } u)$ for dt .

Inputs:

- Func_X_U_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix X, U, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- X -- X input matrix

- U -- U input matrix.
- dT -- Time differential in seconds

Outputs:

- OUTPUT -- The integration of $dx/dt = f(x, u)$ for dt.
- Error -- If TRUE, an error occurred.

NumIntegrate_Rkdp_Func_A



Internal function used by Rkdp_Impl.vi

Inputs:

- Row -- Row to look up (0 - 6)
- Column -- Column value to look up (0 - 6)

Outputs:

- A -- Value looked up by index.

NumIntegrate_Rkdp_Func_B1



Internal function used by Rkdp_Impl.vi

Inputs:

- Index -- Index into B1 array (must be 0 - 6)

Outputs:

- B1 -- Value looked up by index.
-

NumIntegrate_Rkdp_Func_B1B2



Internal function used by Rkdp_Impl.vi

Inputs:

- Index -- Index into B1 and B2 array (must be 0 - 6)

Outputs:

- B1B2 -- Value of B1 - B2 looked up by index.
-

NumIntegrate_Rkdp_Func_B2



Internal function used by Rkdp_Impl.vi

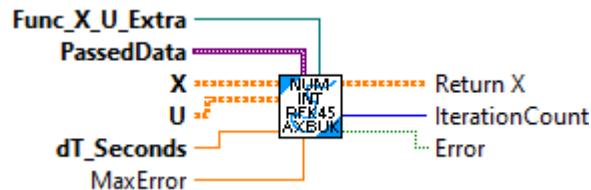
Inputs:

- Index -- Index into B2 array (must be 0 - 6)

Outputs:

- B2 -- Value looked up by index.
-

NumIntegrate_Rkdp_Mat_X_U



Performs adaptive Dormand-Prince integration of $dx/dt = f(x, u)$ for dt . By default, the max error is 1e-6.

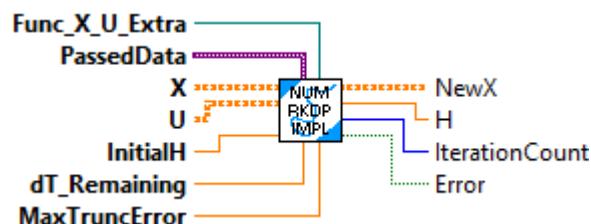
Inputs:

- Func_X_U_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix X, U, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- X -- X input matrix
- U -- U input matrix.
- dT -- The time over which to integrate
- maxError -- The maximum acceptable truncation error. Usually a small number like 1e-6.

Outputs:

- Return X -- the integration of $dx/dt = f(x, u)$ for dt .
- Error -- If TRUE, an error occurred.
- Iteration Count -- The number of times the integration iterated to reach the desired maxError.

NumIntegrate_Rkdp_Impl



Implements one loop of RKDP. This takes an initial state, dt guess, and max truncation error, and returns a new x and the dt over which that x was updated. This should be called until there is no dt remaining.

This is an INTERNAL FUNCTION AND SHOULD NOT BE CALLED DIRECTLY.

Inputs:

- Func_X_U_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix X, U, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- X -- The initial value of x.
- U -- The value u held constant over the integration period.
- InitialH -- The initial dt guess. This is refined to clamp truncation error to the specified max.
- dT_Remaining -- How much time is left to integrate over. Used to clamp h.
- Max TruncError -- The max truncation error acceptable. Usually a small number like 1e-6.

Outputs:

- New X -- the integration of $dx/dt = f(x, u)$ for dt
- H --
- Iteration count -- How many times this routine looped to achieve the desired error tolerance
- Error -- If TRUE, an error occurred.

NumIntegrate_Rkf45_Func_A



Internal function used by Rkdp_Impl.vi

Inputs:

- Row -- Row to look up (0 - 4)

- Column -- Column value to look up (0 - 4)

Outputs:

- A -- Value looked up by index.

NumIntegrate_Rkf45_Func_B1



Internal function used by Rkdp_Impl.vi

Inputs:

- Index -- Index into B1 array (must be 0 - 6)

Outputs:

- B1 -- Value looked up by index.

NumIntegrate_Rkf45_Func_B1B2



Internal function used by Rkdp_Impl.vi

Inputs:

- Index -- Index into B1 and B2 array (must be 0 - 6)

Outputs:

- B1B2 -- Value of B1 - B2 looked up by index.

NumIntegrate_Rkf45_Func_B2



Internal function used by RkdPImpl.vi

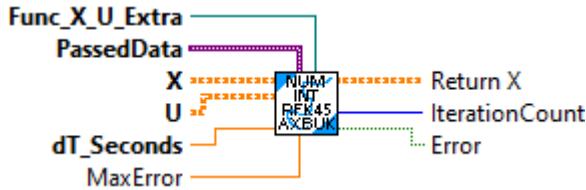
Inputs:

- Index -- Index into B2 array (must be 0 - 6)

Outputs:

- B2 -- Value looked up by index.

NumIntegrate_Rkf45_Mat_X_U



Performs adaptive RKF45 integration of $dx/dt = f(x, u)$ for dt , as described in
https://en.wikipedia.org/wiki/Runge%20%93Kutta%20%93Fehlberg_method

Inputs:

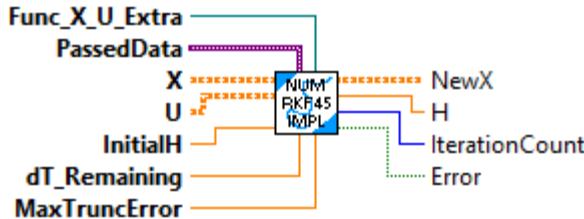
- Func_X_U_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix X, U, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- X -- X input matrix
- U -- U input matrix.
- dT -- The time over which to integrate
- maxError -- The maximum acceptable truncation error. Usually a small number like 1e-6.

Outputs:

- Return X -- the integration of $dx/dt = f(x, u)$ for dt .

- Error -- If TRUE, an error occurred.
 - Iteration Count -- The number of times the integration iterated to reach the desired maxError.
-

NumIntegrate_Rkf45_Impl



Implements one loop of RKDP. This takes an initial state, dt guess, and max truncation error, and returns a new x and the dt over which that x was updated. This should be called until there is no dt remaining.

This is an INTERNAL FUNCTION AND SHOULD NOT BE CALLED DIRECTLY.

Inputs:

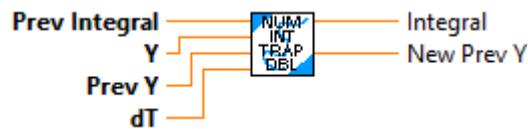
- Func_X_U_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix X, U, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- X -- The initial value of x.
- U -- The value u held constant over the integration period.
- InitialH -- The initial dt guess. This is refined to clamp truncation error to the specified max.
- dT_Remaining -- How much time is left to integrate over. Used to clamp h.
- Max TruncError -- The max truncation error acceptable. Usually a small number like 1e-6.

Outputs:

- New X -- the integration of $dx/dt = f(x, u)$ for dt
- H --
- Iteration count -- How many times this routine looped to achieve the desired error tolerance

- Error -- If TRUE, an error occurred.

NumIntegrate_Trap_Dbl



Performs the calculation for trapezoidal integration. The previous value of X is not stored inside this function. The user is responsible for this.

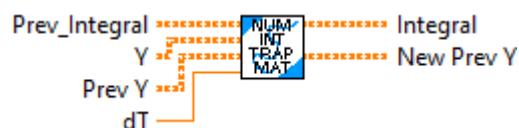
Inputs:

- Prev_Integral -- The previous value of the integrated value
- Y -- Current value of Y
- Prev Y -- The previous value of Y.
- dT -- Time interval over which to integrate

Outputs:

- Integral -- Updated integrated output
- New Prev Y -- The previous value of Y to use the next iteration.

NumIntegrate_Trap_Mat



Performs the calculation for trapezoidal integration of matrices. The previous value of X is not stored inside this function. The user is responsible for this.

Inputs:

- Prev_Integral -- The current value of the integrated matrix

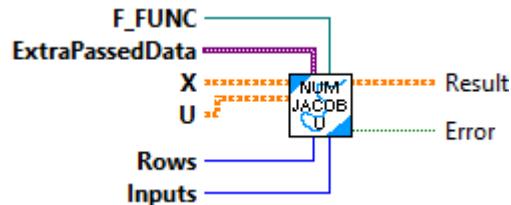
- Y -- Current value of Y matrix
- Prev Y -- The previous value of Y matrix.
- dT -- Time interval over which to integrate

Outputs:

- Integral -- Updated integrated output matrix
- New Prev Y -- The previous value of Y to use the next iteration matrix.

NumJacobian

NumJacobian_U



Returns the numerical Jacobian with respect to u for $f(x, u)$.

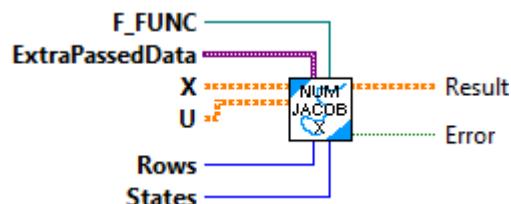
Inputs:

- rows -- Number of rows in the result of $f(x, u)$.
- inputs -- The inputs to the system.
- f -- Vector-valued function reference from which to compute Jacobian.
- x -- State vector.
- u -- Input vector.

Outputs:

- result -- The numerical Jacobian with respect to U for $f(x, u, \dots)$.
- error -- If TRUE, an error occurred.

NumJacobian_X



Returns numerical Jacobian with respect to x for $f(x, u, \dots)$.

Inputs:

- rows -- Number of rows in the result of $f(x, u)$.
- states -- Number of rows in x .
- f -- Vector-valued function reference from which to compute Jacobian.
- x -- State vector.
- u -- input vector.

Outputs:

- result -- The numerical Jacobian with respect to x for $f(x, u, \dots)$.
- error -- If TRUE, an error occurred.

PathfinderUtil

PathfinderUtil_Continuous_Heading_Difference



Normalize a heading difference so that it falls within ($+$ / $- \pi$).

PathfinderUtil_OptimizeTrajectoryStates



Optimize trajectory states by removing states that are too similar. The first and last states are always kept. This reduces the overall number of states. This also prevents potential divide by zero errors during sample interpolation. This is used when converting a Pathfinder Path to a Trajectory.

PathfinderUtil_ToTrajectory



This routine provides a complete conversion of a Pathfinder Path to a Trajectory. Each Pathfinder path sample is converted to a Trajectory State. The Trajectory states are optimized. (This is required to ensure that divide by zero doesn't occur when interpolating between states.) Then a Trajectory is created from the array of Trajectory states.

Parameters:

- Pathfinder Path - Array of pathfinder created states

Returns:

- Trajectory - Trajectory data structure.

PathfinderUtil_ToTrajectoryStates



Convert an array of Pathfinder created Path states to an array of Trajectory States. Inches are converted to Meters. Angles are converted to headings (+/- PI). This routine only does a portion of the work needed to convert a Pathfinder Path to a complete Trajectory. (See PathfinderUtil_ToTrajectory for a complete routine.)

Parameters:

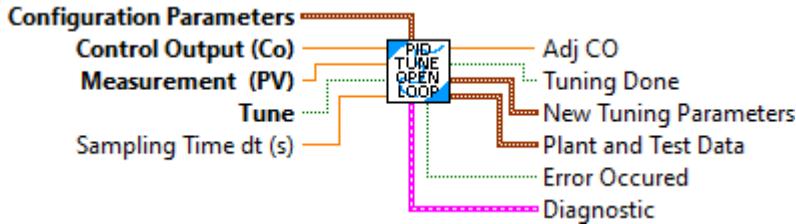
- Pathfinder Path - Array of pathfinder created states

Returns:

- Trajectory States - Array of Trajectory states.

PIDAutoTune

PIDAutoTune_ClosedLoopStep



This routine can be used to make a step change to a PID control loop, track the results, and calculate PID tuning parameters. It wraps the NI LabVIEW PID auto-tuning functions for use with the WPILIB LabVIEW Math PID. This uses the CLOSED LOOP STEP type of test.

Warning -- This routine will modify the control output. Use it in such a manner to be able to stop the tuning if problems occur -- like running out of field space while tuning a drive motor.

Inputs:

- Configuration Parameters -- cluster -- This contains the parameters defining the conditions of the test. Use the Pack routine to create this data cluster.
- Control Output -- double -- The Control Output (CO) value from the output of the PID to be tuned.
- Measurement (PV) -- double -- The Process Variable (PV) value used by the PID to be tuned.
- Tune -- boolean -- When TRUE, tuning is performed..
- Sampling time -- double -- Loop time in seconds. How fast the PID and this autotune function are executed. (Optional: Default 0.02 seconds)

Outputs:

- Adj CO -- double -- The adjusted CO. This is sent to the actuator. (This vi is put between the PID and the actuator device)
- Tuning Done -- boolean -- This is set to TRUE when tuning has completed. Depending on the PV noise, tuning may never be "DONE". However if the new tuning parameters are relatively constant, they can still be used.
- New Tuning Parameters -- cluster --- Data cluster containing the calculated PID tuning parameters.
- Plant and Test Data -- cluster -- Data cluster containing information describing the transfer function of the system being controlled (Plant).

-- Error Occured -- Boolean -- If TRUE, an error occurred.

-- Diagnostic Info -- cluster -- This contains additional information about the test being performed.

PIDAutoTune_Convert_Academic_To_NonInteracting



Convert PID tuning parameters calculated by the NI LabVIEW PID autotuning from the NI Academic (classical) PID form to the non-interacting PID form used by the WPILIB LabVIEW Math library.

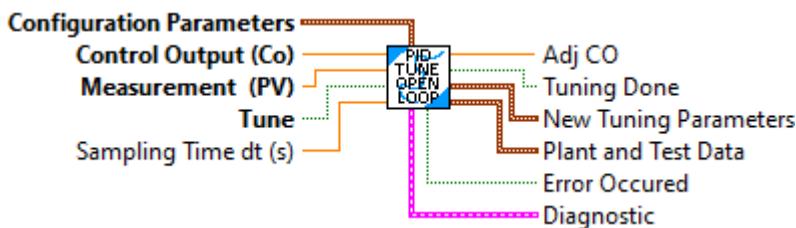
Inputs:

- PID Gains NI Academic -- cluster -- PID tuning parameters in NI Academic form.
- Kn -- double -- PID normalization constant. This is normally Max CO / Max PV (If normalization constant is not used, this value should be set to 1.0)

Outputs:

- PID Gains -- cluster -- Converted PID tuning parameter gains.

PIDAutoTune_OpenLoopStep



This routine can be used to make a step change to a PID control loop, track the results, and calculate PID tuning parameters. It wraps the NI LabVIEW PID auto-tuning functions for use with the WPILIB LabVIEW Math PID. This uses the OPEN LOOP STEP type of test.

Warning -- This routine will modify the control output. Use it in such a manner to be able to stop the tuning if problems occur -- like running out of field space while tuning a drive motor.

Inputs:

-- Configuration Parameters -- cluster -- This contains the parameters defining the conditions of the test. Use the Pack routine to create this data cluster.

-- Control Output -- double -- The Control Output (CO) value from the output of the PID to be tuned.

-- Measurement (PV) -- double -- The Process Variable (PV) value used by the PID to be tuned.

-- Tune -- boolean -- When TRUE, tuning is performed..

-- Sampling time -- double -- Loop time in seconds. How fast the PID and this autotune function are executed. (Optional: Default 0.02 seconds)

Outputs:

-- Adj CO -- double -- The adjusted CO. This is sent to the actuator. (This vi is put between the PID and the actuator device)

-- Tuning Done -- boolean -- This is set to TRUE when tuning has completed. Depending on the PV noise, tuning may never be "DONE". However if the new tuning parameters are relatively constant, they can still be used.

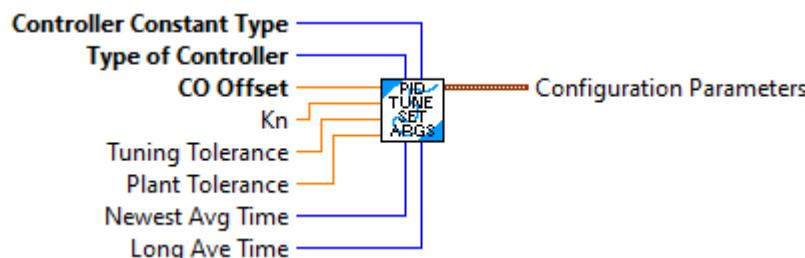
-- New Tuning Parameters -- cluster --- Data cluster containing the calculated PID tuning parameters.

-- Plant and Test Data -- cluster -- Data cluster containing information describing the transfer function of the system being controlled (Plant).

-- Error Occured -- Boolean -- If TRUE, an error occurred.

-- Diagnostic Info -- cluster -- This contains additional information about the test being perfromed.

PIDAUTOTUNE_SetTuningArguments



Pack the individual PID AutoTune testing parameters into a data cluster for use with the PID AutoTune functions.

Inputs:

-- Control Constant Type -- enum -- Desired PID performance. This guides the calculation of the tuning parameters after the system transfer function (plant) has been determined. This choice determines how aggressive the PID is in controlling the system error. Choices are:

- Ziegler Nicols - Fast
- Ziegler Nicols - Normal
- Ziegler Nicols - Slow
- CHR Regulator - 0% Overshoot
- CHR Regulator - 20% Overshoot
- CHR Servo - 0% Overshoot
- CHR Servo - 20% Overshoot

-- Type of Controller -- enum -- Type of PID. Choices are:

- P -- Proportional only
- PI -- Proportional and Integral only
- PID -- Full Proportional, Integral, Derivative.

-- CO Offset -- double -- The amount to step the control output to perform the test

-- Kn -- double -- Normalization constant. This is the same value that is used with the PID. This is normally Max CO / Max PV. If this isn't used, set the value to 1.0. (Optional. Default: 1.0)

-- Tuning Tolerance -- double -- Value used to determine if the tuning is complete. (Optional. Default 0.001)

-- Plant Tolerance -- double -- Value used to determine if the tuning process is complete. (Optional. Default: 0.001)

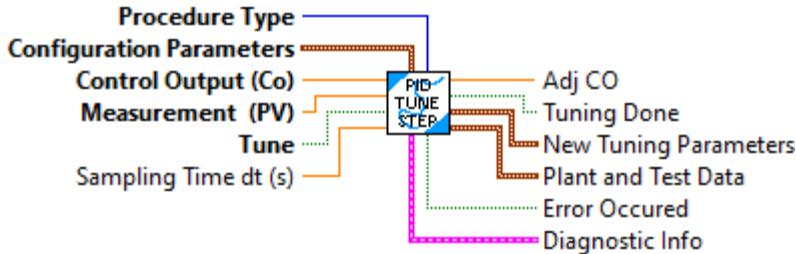
-- Newest Avg Time -- double -- Time in seconds to average most recent calculated tuning constants. (Optional. Default: 1.0)

-- Long Avg Time -- double -- Time in seconds to average tuning constants to determine if the values are still changing. (Optional. Default 6.0)

Outputs:

-- Configuration Parameters -- cluster -- This contains the parameters defining the conditions of the test. Use the Pack routine to create this data cluster.

PIDAutoTune_Step_Execute



This routine can be used to make a step change to a PID control loop, track the results, and calculate PID tuning parameters. It wraps the NI LabVIEW PID auto-tuning functions for use with the WPILIB LabVIEW Math PID.

Warning -- This routine will modify the control output. Use it in such a manner to be able to stop the tuning if problems occur -- like running out of field space while tuning a drive motor.

Inputs:

- Procedure Type -- enum -- The type of tuning procedure to be used: Values are:
 - Open Loop Step -- This uses the frozen CO input when the test starts. A step change is made to the output. This is the recommended method
 - Closed Loop Step -- This uses the continually changing CO input and adds an offset value to this during the test.
- Configuration Parameters -- cluster -- This contains the parameters defining the conditions of the test. Use the Pack routine to create this data cluster.
- Control Output -- double -- The Control Output (CO) value from the output of the PID to be tuned.
- Measurement (PV) -- double -- The Process Variable (PV) value used by the PID to be tuned.
- Tune -- boolean -- When TRUE, tuning is performed..
- Sampling time -- double -- Loop time in seconds. How fast the PID and this autotune function are executed. (Optional: Default 0.02 seconds)

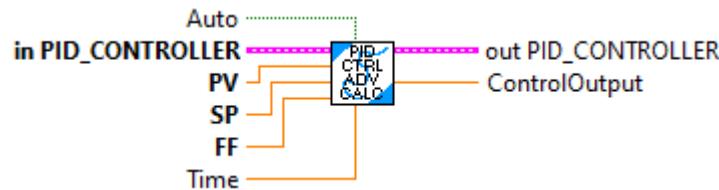
Outputs:

- Adj CO -- double -- The adjusted CO. This is sent to the actuator. (This vi is put between the PID and the actuator device)

- Tuning Done -- boolean -- This is set to TRUE when tuning has completed. Depending on the PV noise, tuning may never be "DONE". However if the new tuning parameters are relatively constant, they can still be used.
- New Tuning Parameters -- cluster --- Data cluster containing the calculated PID tuning parameters.
- Plant and Test Data -- cluster -- Data cluster containing information describing the transfer function of the system being controlled (Plant).
- Error Occured -- Boolean -- If TRUE, an error occurred.
- Diagnostic Info -- cluster -- This contains additional information about the test being performed.

PIDController

PIDController_AdvCalculate_FF_Sp_Pv



Returns the next output of the "advanced" PID controller. This controller implements the same PID as the "normal" controller with the following enhancements:

- Integration by trapezoids is used.
- Optionally filters derivative terms. (See SetDerivativeFilter subVI documentation for information.)
- Incorporates optional feedforward.
- Includes integral windup protection. This is only useful when output limits are specified.
- When not called with a specific period, uses the actual elapsed time. (This subVI calculates the actual elapsed time.)
- Limits the outputs. When output limits are specified the output is coerced be within these limits.
- Includes an Auto/Manual mode. In manual mode the scaled feedforward is sent directly to the output and the PID controller tracks this output value to allow for smooth transition back to auto mode.

The generalized equation for this PID when in auto is:

$$CO = Kf \times \text{feedforward} + Kp \times \text{error} + Ki \times \text{Integral(error)} + Kd \times \text{Derivative(error)}$$

When auto mode is disabled, the generalized equation is:

$$CO = Kf \times \text{feedforward}$$

Inputs:

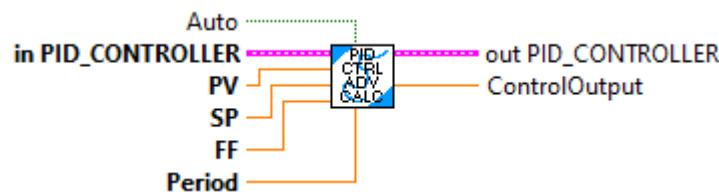
- PID_Controller -- PID_Controller data cluster
- PV-measurement -- The current measurement of the process variable. (PV)
- SP-setpoint -- The new setpoint of the controller. (SP)

- FF-feedforward -- Feedforward in the same units as the PV and SP
- Auto -- Use the PID calculate when this is True

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
- CO-ControlOutput -- Controller output (CO)

PIDController_AdvCalculate_FF_Sp_Pv_Per



Returns the next output of the "advanced" PID controller. This controller implements the same PID as the "normal" controller with the following enhancements:

- Integration by trapezoids is used.
- Optionally filters derivative terms. (See SetDerivativeFilter subVI documentation for information.)
- Incorporates optional feedforward.
- Includes integral windup protection. This is only useful when output limits are specified.
- When not called with a specific period, uses the actual elapsed time. (This subVI requires the call to pass the time period.)
- Limits the outputs. When output limits are specified the output is coerced be within these limits.
- Includes an Auto/Manual mode. In manual mode the scaled feedforward is sent directly to the output and the PID controller tracks this output value to allow for smooth transition back to auto mode.

The generalized equation for this PID when in auto is:

$$CO = K_f \times \text{feedforward} + K_p \times \text{error} + K_i \times \text{Integral(error)} + K_d \times \text{Derivative(error)}$$

When auto mode is disabled, the generalized equation is:

$$CO = K_f \times \text{feedforward}$$

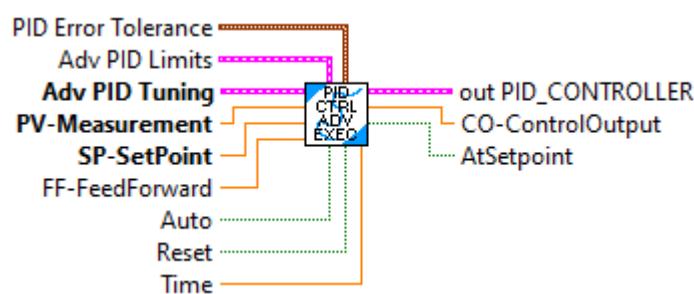
Inputs:

- PID_Controller -- PID_Controller data cluster
- PV-measurement -- The current measurement of the process variable. (PV)
- SP-setpoint -- The new setpoint of the controller. (SP)
- FF-feedforward -- Feedforward in the same units as the PV and SP
- Period -- Period of this controller in seconds.
- Auto -- Use the PID calculate when this is True

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
- CO-ControlOutput -- Controller output (CO)

PIDController_AdvExecute



Convenience, single call, LabVIEW function. Creates and calculates the "advanced" PID controller. Call this routine periodically to calculate the newest output for the provided inputs.

See the "PID Adv Calculate" VI documentation for additional details on the "advanced PID"

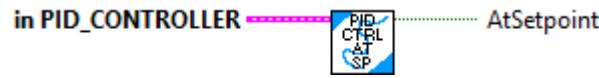
Inputs:

- PID Error Tolerance -- Tolerances for determining "At Setpoint", contains:
 - Position Tolerance -- (Default: 0.001)
 - Velocity Tolerance -- (Default: 9.9E+30)
- Adv PID Limits -- Cluster containing:
 - MaxInput -- (Default: 0)
 - MinInput -- (Default: 0)
 - Continous -- When True indicates that the input is continuous. (Default: False)
 - MaxOutput -- The largest allowed value for the control output (CO)
 - MinOutput -- The smallest allowed value for the control output (CO)
- PID Tuning -- PID Tuning parameters containing:
 - Kf -- The feedforward gain coefficient.
 - Kp -- The proportional coefficient.
 - Ki -- The integral coefficient.
 - Kd -- The derivative coefficient.
 - Maximum Integral -- The maximum value of the integrator.
 - Minimum Integral -- The minimum value of the integrator.
 - Filter Derivative -- Use derivative filtering when True.
- PV-measurement -- The current measurement of the process variable. (PV)
- SP-setpoint -- The new setpoint of the controller. (SP)
- FF-Feedforward -- The feedforward value.
- Auto -- Use the PID calculate when this is True
- Reset -- When True, causes the PID to be reset. (Default: False)

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
- CO-ControlOutput -- Controller output (CO)
- AtSetpoint -- True when PV is within the specified tolerances of the SP.

PIDController_AtSetpoint



Returns true if the error is within the percentage of the total input range, determined by SetTolerance. This assumes that the maximum and minimum input were set using SetInput.

This will return false until at least one input value has been computed.

This uses the error values from the most recently called calculate routine. It should be called after calling the calculate routine.

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- AtSetpoint -- Whether the error is within the acceptable bounds.

PIDController_CalculateError_Internal



Internal routine to calculate the position and velocity errors.

PIDController_Calculate_PV



Returns the next output of the PID controller. The saved setpoint (SP) in the PID_Controller data structure is used.

Inputs:

- PID_Controller -- PID_Controller data cluster
- PV-measurement -- The current measurement of the process variable. (PV)

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
- CO-ControlOutput -- Controller output (CO)

PIDController_Calculate_SP_PV



Returns the next output of the PID controller.

Inputs:

- PID_Controller -- PID_Controller data cluster
- PV-measurement The current measurement of the process variable. (PV)
- SP-setpoint The new setpoint of the controller. (SP)

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
- CO-ControlOutput -- Controller output (CO)

PIDController_DisableContinuousInput



Disables continuous input.

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_EnableContinuousInput



Enables continuous input.

Rather than using the max and min input range as constraints, it considers them to be the same point and automatically calculates the shortest route to the setpoint.

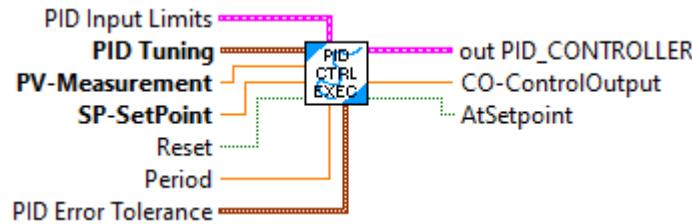
Inputs:

- PID_Controller -- PID_Controller data cluster
- minimumInput -- The minimum value expected from the input.
- maximumInput -- The maximum value expected from the input.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_Execute



Convenience, single call, LabVIEW function. Creates and calculates the PID controller. Call this routine periodically to calculate the newest output for the provided inputs.

Inputs:

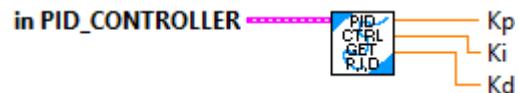
- PID Input Limits -- Cluster containing:
 - MaxInput -- (Default: 0)
 - MinInput -- (Default: 0)
 - Continous -- When True indicates that the input is continuous. (Default: False)
- PID Tuning -- PID Tuning parameters containing:
 - Kp -- The proportional coefficient.
 - Ki -- The integral coefficient.
 - Kd -- The derivative coefficient.
 - Maximum Integral -- The maximum value of the integrator.
 - Minimum Integral -- The minimum value of the integrator.
- PV-measurement The current measurement of the process variable. (PV)
- SP-setpoint The new setpoint of the controller. (SP)
- Reset -- When True, causes the PID to be reset. (Default: False)

- Period -- Period of repeated calls in seconds. (Default: 0.020)
- PID Error Tolerance -- Tolerances for determining "At Setpoint", contains:
 - Position Tolerance -- (Default: 0.001)
 - Velocity Tolerance -- (Default: 9.9E+30)

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
- CO-ControlOutput -- Controller output (CO)
- AtSetpoint -- True when PV is within the specified tolerances of the SP.

PIDController_GetPID



Get the Proportional, Intergral, and Differential coefficients.

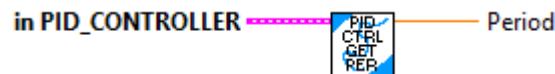
Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- Kp -- The proportional coefficient.
- Ki -- The integral coefficient.
- Kd -- The derivative coefficient.

PIDController_GetPeriod



Returns the period of this controller.

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- period -- the period of the controller.

PIDControllerGetPositionError



Returns the difference between the setpoint and the measurement.

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- positionError -- The error.

PIDController_GetSetpoint



Returns the current setpoint of the PIDController.

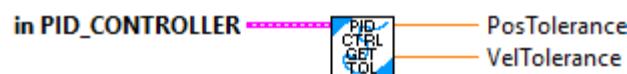
Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- setpoint -- The current setpoint.
-
-

PIDController_GetTolerance



Returns the position and velocity tolerance values

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- PosTolerance -- The position tolerance
 - VelTolerance -- The velocity tolerance
-
-

PIDController_GetVelocityError



Returns the velocity error.

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- velocityError -- The velocity error

PIDController_IsContinuousInputEnabled



Returns true if continuous input is enabled

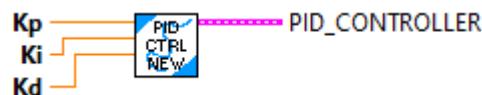
Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- ContinuousInputEnabled -- TRUE if continuous input is enabled.

PIDController_New



Allocates a PIDController data cluster with the given constants for Kp, Ki, and Kd and a default period of 0.02 seconds.

Implements a PID control loop. This data structure also can be used with the "Advanced" PID calculation subVIs.

Inputs:

- Kp -- The proportional coefficient.
- Ki -- The integral coefficient.
- Kd -- The derivative coefficient.

Outputs:

- PID_Controller -- Initialized PID_Controller data cluster
-
-

PIDController_NewPeriod



Allocates a PIDController data cluster with the given constants for Kp, Ki, and Kd.

Implements a PID control loop. This data structure also can be used with the "Advanced" PID calculation subVIs.

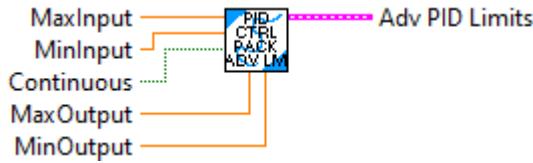
Inputs:

- Kp -- The proportional coefficient.
- Ki -- The integral coefficient.
- Kd -- The derivative coefficient.
- period -- The period between controller updates in seconds.

Outputs:

- PID_Controller -- Initialized PID_Controller data cluster
-
-

PIDController_Pack_AdvLimits



Convenience function to pack individual input and output limit values into a cluster to pass to the Advanced PID controller

Note -- These values are considered to be static after they are initially set. Changes during run-time may be ignored.

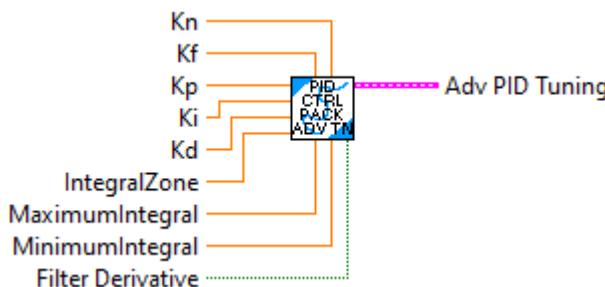
Inputs:

- Max Input -- The maximum allowed input The units are the same as the Setpoint (SP) and Process variable (PV) (When both Max and Min values are 0.0, input limit checking is disabled.)
- Min Input -- The minimum allowed input value The units are the same as the Setpoint (SP) and Process variable (PV) (When both Max and Min values are 0.0, input limit checking is disabled.)
- Continuous -- Boolean indicating that the inputs wrap from the maximum value to the minimum value.. (Example - gyro and heading control)
- MaxOutput -- The maximum allowed output The units are the same as the output. This is used to clamp the output and is used for anti-windup protection (Default 9.9E+32)
- MinOutput -- The maximum allowed output The units are the same as the output. This is used to clamp the output and is used for anti-windup protection (Default -9.9E+32)

Output:

- Adv PID Input Limits -- Cluster containing the input values

PIDController_Pack_AdvTuning



Convenience function to pack individual tuning parameter values into a cluster to pass to the Advanced PID controller

Inputs:

- Kn -- Optional normalizing constants. When present, the other constants are multiplied by this constant. Set this equal to the maximum controller output divided by the corresponding maximum process value that corresponds to the maximum controller output. Using this can help simplify tuning.

- Kf -- Feedforward gain constant. The feedforward value is multiplied by this value. Generally this translates the units of the input to the units of the output. (For example if a drive maximum speed is 10 FT/SEC, when the motor output is 1.0, this gain can be set to 0.1.)

- Kp -- Proportional gain constant. The error is multiplied by this value. (Default = 0.0)

- Ki -- Integral gain constant. The integral of the error is multiplied by this value. (Default = 0.0)

- Kd -- Derivative gain constant. The derivative of the error is multiplied by this value. (Default = 0.0)

- IntegralZone - double - Absolute value of error (in process variable units) to allow integration. Specifying this value helps to prevent integral windup when moving from one setpoint to another. (Default = 9.9E+30)

- Maximum Integral -- The integral component of the output is not allowed to exceed this value. This limits the travel of the integral. This is in units of the output value. (Default = 9.9E+30)

- Minimal Integral -- The integral component of the output is not allowed to be less than this value. This limits the travel of the integral. This is in units of the output value. (Default = -9.9E+30)

- Filter Derivative -- Taking a derivative of the error amplifies the noise in this signal. When this boolean is TRUE, the error value used to calculate the derivative portion of the output is passed through a 3 sample average filter.

Output:

- Adv PID Tuning -- Cluster containing the tuning values.

PIDController_Pack_ErrorTolerance



Convenience function to pack the input tolerance values into a data cluster that is used as input to the PID controller.

Inputs:

- Position Tolerance -- Position tolerance in the same units as the PV and SP. (Default 0.001)
 - Velocity Tolerance -- Velocity tolerance in the same units as the PV and SP.(Default 9.9E30)
-

PIDController_Pack_InputLimits



Convenience function to pack individual input limit values into a cluster to pass to the PID controller.

Note -- These values are considered to be static after they are initially set. Changes during run-time may be ignored.

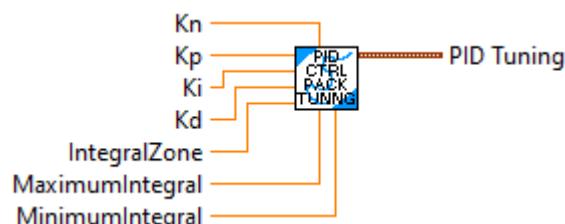
Inputs:

- Max Input -- The maximum allowed input The units are the same as the Setpoint (SP) and Process variable (PV)
- Min Input -- The minimum allowed input value The units are the same as the Setpoint (SP) and Process variable (PV)
- Continuous -- Boolean indicating that the inputs wrap from the maximum value to the miimum value.. (Example - gyro and heading control)

Output:

- PID Input Limits -- Cluster containing the input values
-

PIDController_Pack_Tuning



Convenience function to pack individual tuning parameter values into a cluster to pass to the PID controller

Inputs:

- Kn -- Optional normalizing constants. When present, the other constants are multiplied by this constant. Set this equal to the maximum controller output divided by the corresponding maximum process value that corresponds to the maximum controller output. Using this can help simplify tuning.
- Kp -- Proportional gain constant. The error is multiplied by this value. (Default = 0.0)
- Ki -- Integral gain constant. The integral of the error is multiplied by this value. (Default = 0.0)
- Kd -- Derivative gain constant. The derivative of the error is multiplied by this value. (Default = 0.0)
- Maximum Integral -- The integral component of the output is not allowed to exceed this value. This limits the travel of the integral. This is in units of the output value. (Default = 9.9E+30)
- Minimal Integral -- The integral component of the output is not allowed to be less than this value. This limits the travel of the integral. This is in units of the output value. (Default = -9.9E+30)
- IntegralZone - double - Absolute value of error (in process variable units) to allow integration. Specifying this value helps to prevent integral windup when moving from one setpoint to another. (Default = 9.9E+30)

Output:

- PID Tuning -- Cluster containing the tuning values.

PIDController_Reset



Resets the previous error and the integral term. This also resets stored data for the advanced PID.

Inputs:

- PID_Controller -- PID_Controller data cluster

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetD



Sets the Differential coefficient of the PID controller gain.

Inputs:

- PID_Controller -- PID_Controller data cluster
- Kd -- differential coefficient

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
-

PIDController_SetDerivativeFilter



Enable or disable filtering on the calculated derivative term. When enabled the filter derivative is calculated as the average of the last three raw derivative terms. This is useful when the PV (process value) is noisy. Derivative filtering only functions when using the "advanced" PID.

Inputs:

- PID_Controller -- PID_Controller data cluster
- DerivativeFilter -- Enable the derivative filter function when input is True.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetI



Sets the Integral coefficient of the PID controller gain.

Inputs:

- PID_Controller -- PID_Controller data cluster
- Ki -- integral coefficient

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetIntegratorRange



Sets the minimum and maximum values for the integrator.

When the cap is reached, the integrator value is added to the controller output rather than the integrator value times the integral gain.

Inputs:

- PID_Controller -- PID_Controller data cluster
- minimumIntegral -- The minimum value of the integrator.
- maximumIntegral -- The maximum value of the integrator.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetIntegratorZone



Sets the integrator zone.

Integration is only allowed when the absolute value of the error is less than the value of Integrator Zone. This helps to prevent PID integral windup when moving from one setpoint to another.

Inputs:

- PID_Controller -- PID_Controller data cluster
- IntegralZone - double - The new integral zone. Absolute value of the error to allow integration.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetOutputLimits



Sets the minimum and maximum output value. When these values are set the "advanced" PID can provide integral windup protection. If this routine is not used, the default output values are essentially +/- infinity. PID windup protection is only available when using the "advanced" PID.

Inputs:

- PID_Controller -- PID_Controller data cluster
- MinOutput -- The smallest allowed value for the control output (CO)
- MaxOutput -- The largest allowed value for the control output (CO)

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetP



Sets the Proportional coefficient of the PID controller gain.

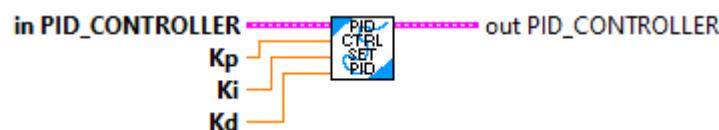
Inputs:

- PID_Controller -- PID_Controller data cluster
- Kp proportional -- coefficient

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetPID



Sets the PID Controller gain parameters.

Set the proportional, integral, and differential coefficients.

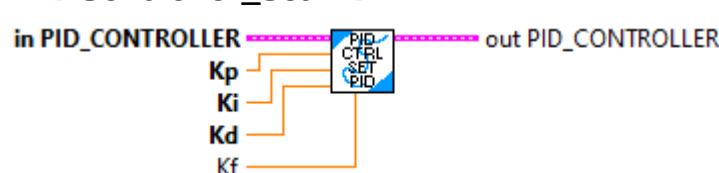
Inputs:

- PID_Controller -- PID_Controller data cluster
- Kp -- The proportional coefficient.
- Ki -- The integral coefficient.
- Kd -- The derivative coefficient.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PIDController_SetPIDF



Sets the PID Controller and feedforward gain parameters.

Set the proportional, integral, differential, and feedforward coefficients. The feedforward is only used by the advanced PID.

Inputs:

- PID_Controller -- PID_Controller data cluster
- Kf -- The feedforward gain coefficient
- Kp -- The proportional coefficient.
- Ki -- The integral coefficient.
- Kd -- The derivative coefficient.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
-

PIDController_SetPeriod



Sets the period, in seconds, of this controller.

Inputs:

- PID_Controller -- PID_Controller data cluster
- period -- the period of the controller.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
-

PIDController_SetSetpoint



Sets the setpoint for the PIDController.

Inputs:

- PID_Controller -- PID_Controller data cluster
- setpoint -- The desired setpoint.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
-
-

PIDController_SetTolerance



Sets the error which is considered tolerable for use with atSetpoint().

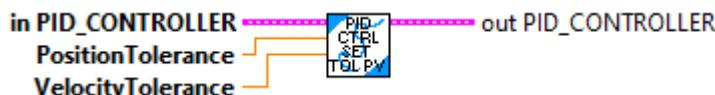
Inputs:

- PID_Controller -- PID_Controller data cluster
- positionTolerance -- Position error which is tolerable.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster
-
-

PIDController_SetTolerancePandV



Sets the error which is considered tolerable for use with atSetpoint().

Inputs:

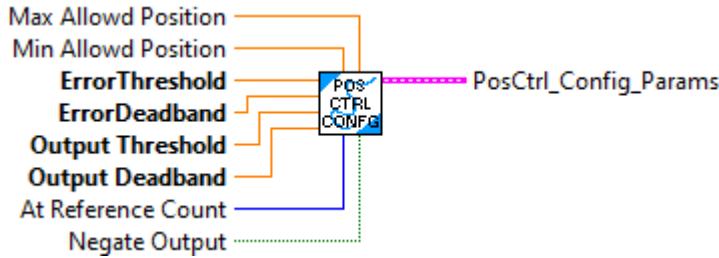
- PID_Controller -- PID_Controller data cluster
- positionTolerance -- Position error which is tolerable.
- velocityTolerance -- Velocity error which is tolerable.

Outputs:

- PID_Controller -- Updated PID_Controller data cluster

PosCtrl

PosCtrl_Config_Threshold



This VI creates the configuration parameters for the position control controller. The straight line slope and intercept are calculated from the error threshold, error deadband, output threshold, and output deadband. A separate maximum output input to the controller executable allows the rate of output reduction to be constant for different maximum output values.

The optional inputs Max Allowed Position and Min Allowed Position provide a virtual limit switch capability that forces the output to zero when it would cause movement in the direction of the exceeded limit.

The optional "at reference count" parameter specifies the number of consecutive execution cycles the absolute value of the error has to be within the deadband to set "At Reference" to true. For a 20 millisecond loop time, the default value of 13 forces the error to be within the deadband for 260 milliseconds before At Reference is set to true.

The optional Negate Output parameter causes the output value to be multiplied by negative one.

Inputs:

- Max Allowed Position - double - (Optional,, default: 9E+30)
- Max Allowed Position - double - (Optional,, default: 9E+30)
- Error threshold- double - When the error reaches this point, the output will start reducing from Output Threshold to Output Deadband as the error decreases. Thie value must be positive.
- Error Deadband - double - When the absolute value of the error is less than this, the output will be set to zero. Thie value must be positive.
- Output Threshold - double - This is the output value used to calculate the slope of the slow down. The actual Maximum Output is specified in the Execute VI. The actual Maximum output can be greater or smaller than this value. Thie value must be positive.
- Output Deadband - double - This is the output value when error reaches the Error Deadband. Thie value must be positive.

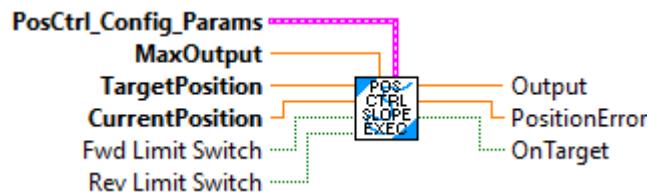
- At Reference count - integer - The number of consecutive cycles that the error has to be within the Error Deadband to set the At Reference output to TRUE. (Optional, default: 13)

- Negate Output - boolean - When TRUE causes the output to be negated. (Optional, default: FALSE)

Outputs:

- PosCtrl_Config_Params - cluster - Data cluster containing the configuration parameters for the Position Control Execute VI.

PosCtrl_Execute



This VI calculates closed loop position control output (CO), provided the Setpoint (SP), and Process Variable (PV). It incorporates a position deadband. It also incorporates output slowdown as the desired position is approached.

Pose2d

Pose2d_Div



Divide the pose by the given scalar"

Parameters:

- IN POSE - The input POSE data structure
- Scalar - The value to divide the pose by.

Return:

- OUT POSE - The output pose.

Pose2d_Equals



Checks equality between this Pose2d and another Pose2d.

Parameters:

- this Pose - This POSE2D data structure
- other POSE - The other POSE2d data structure

Returns:

- Equals - Value will be TRUE if both POSES are the same

Pose2d_Exp



Obtain a new Pose2d from a (constant curvature) velocity.

See [Controls Engineering in the FIRST Robotics Competition](#) section on nonlinear pose estimation for derivation.

The twist is a change in pose in the robot's coordinate frame since the previous pose update. When the user runs `exp()` on the previous known field-relative pose with the argument being the twist, the user will receive the new field-relative pose.

"Exp" represents the pose exponential, which is solving a differential equation moving the pose forward in time.

PARAMETERS:

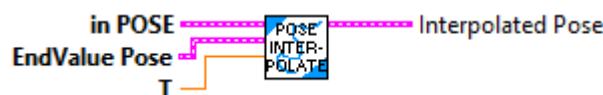
- POSE - current POSE data structure

- TWIST - The change in pose in the robot's coordinate frame since the previous pose update. For example, if a non-holonomic robot moves forward 0.01 meters and changes angle by 0.5 degrees since the previous pose update, the twist would be `Twist2d{0.01, 0.0, toRadians(0.5)}`

RETURNS:

- OUT POSE - The new pose of the robot.

Pose2d_Interpolate



Interpolate between 2 poses

Parameter:

- IN POSE - Current POSE data structure
- EndValue POSE -- The end value POSE..
- T -- Value between 0 and 1.

Returns:

- Interpolated POSE - The interpolated POSE.

Pose2d_Log



Returns a Twist2d that maps this pose to the end pose. If c is the output of $a.\text{Log}(b)$, then $a.\text{Exp}(c)$ would yield b .

Parameters:

- IN POSE - The current POSE data structure
- OTHER POSE - The end pose for the transformation.

Returns:

- OUT TWIST - The twist that maps this to end.

Pose2d_Minus



Returns the Transform2d that maps the one pose to another.

Parameter:

- IN POSE - Current POSE data structure
- OTHER POSE - The initial pose of the transformation.

Returns:

- OUT TRANSFORMATION - The transform that maps the other pose to the current pose.
-

Pose2d_NearestTo



Returns the nearest Pose2d from a list of poses. If two or more poses in the list have the same distance from this pose, return the one with the closest rotation component.

Parameter:

- IN POSE -- Pose2d -- The pose to compare against the OTHER POSES to find the closest pose.
- OTHER POSE S -- Pose2d array -- The list of poses to search for the closest pose.

Returns:

- Nearest Pose -- Pose2d -- The pose closest to IN POSE.
 - Found -- boolean -- True if a POSE was found. This should only be false if the Other Pose array is empty.
-

Pose2d_New



Convenience constructors that takes in x and y values directly instead of having to construct a Translation2d.

Parameters:

- X - The x component of the translational component of the pose.
- Y - The y component of the translational component of the pose.

- Rotation (Heading) - The rotational component of the pose.

Returns:

- OUT POSE - Output POSE

Pose2d_New_TRRO



Constructs a pose with the specified translation and rotation.

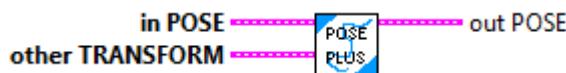
Parameters:

- translation - The translational component of the pose.
- rotation - The rotational component of the pose.

Return:

- POSE - Pose data structure

Pose2d_Plus



Transforms the pose by the given transformation and returns the new transformed pose. "plus" is the same as "transformBy"

The matrix multiplication is as follows

$$[x_{\text{new}}] = [\cos, -\sin, 0][\text{transform.x}]$$

$$[y_{\text{new}}] += [\sin, \cos, 0][\text{transform.y}]$$

$$[t_{\text{new}}] = [0, 0, 1][\text{transform.t}]$$

Parameters:

- IN POSE - The input POSE data structure
- OTHER TRANSFORM - The transform to transform the pose by.

Return:

- OUT POSE - The transformed pose.

Pose2d_RelativeTo



Returns the other pose relative to the current pose.

This function can often be used for trajectory tracking or pose stabilization algorithms to get the error between the reference and the current pose.

Parameters:

- IN POSE - The POSE data structure
- OTHER POSE - The pose that is the origin of the new coordinate frame that the current pose will be converted into.

Return:

- OUT POSE - The current pose relative to the new origin pose.

Pose2d_Times



Multiply the pose by the given scalar"

Parameters:

- IN POSE - The input POSE data structure
- Scalar - The value to multiply the pose by.

Return:

- OUT POSE - The output pose.

Pose2d_TransformBy



Transforms the pose by the given transformation and returns the new transformed pose. "plus" is the same as "transformBy"

The matrix multiplication is as follows

$$\begin{aligned}
 [x_{\text{new}}] &= [\cos, -\sin, 0][\text{transform.x}] \\
 [y_{\text{new}}] &+= [\sin, \cos, 0][\text{transform.y}] \\
 [t_{\text{new}}] &= [0, 0, 1][\text{transform.t}]
 \end{aligned}$$

Parameters:

- IN POSE - The input POSE data structure
- OTHER TRANSFORM - The transform to transform the pose by.

Return:

- OUT POSE - The transformed pose.

Pose2d_getRotation



Returns the rotational component of the transformation.

Parameters:

- IN POSE -- Input POSE data structure

RETURNS:

- out ROTATION - The rotational component of the pose.

Pose2d_getTranslation



Returns the translation component of the transformation.

Parameters:

- IN POSE - The POSE data structure

RETURNS:

- OUT TRANSLATION - The translational component of the pose.

Pose2d_getXY



Returns the X, Y elements of the translation component of the transformation.

Parameters:

- IN POSE - The POSE data structure

RETURNS:

- X - The X element of the translational component of the pose.

- Y - The Y element of the translational component of the pose.
-

Pose2d_getXYAngle



Returns the X, Y, Angle components of the pose.

Parameters:

- IN POSE - The POSE data structure

RETURNS:

- X - X translation value
- Y - Y translation value
- Rotataion angle value

Pose3d

Pose3d_Div



Divide the pose by the given scalar and returns the new pose3d.

Parameters:

- IN POSE3D - The input POSE data structure
- Scalar - The value to divide the pose by.

Return:

- OUT POSE3D - The output pose.

Pose3d_Equals



Checks equality between this Pose3d and another Pose3d.

Parameters:

- this Pose3d - This POSE3D data structure
- other POSE3d - The other POSE23 data structure

Returns:

- Equals - Value will be TRUE if both POSEs are the same

Pose3d_Exp



Obtain a new Pose3d from a (constant curvature) velocity.

The twist is a change in pose in the robot's coordinate frame since the previous pose update. When the user runs exp() on the previous known field-relative pose with the argument being the twist, the user will receive the new field-relative pose.

"Exp" represents the pose exponential, which is solving a differential equation moving the pose forward in time. Obtain a new Pose2d from a (constant curvature) velocity.

PARAMETERS:

- POSE3D - current POSE data structure

- TWIST 3D- TThe change in pose in the robot's coordinate frame since the previous pose update. For example, if a non-holonomic robot moves forward 0.01 meters and changes angle by 0.5 degrees since the previous pose update, the twist would be Twist3d(0.01, 0.0, 0.0, new new Rotation3d(0.0, 0.0, Units.degreesToRadians(0.5))).

RETURNS:

- OUT POSE3D - The new pose of the robot.

- Error -- boolean -- If TRUE, an error occurred.

Pose3d_Interpolate



Interpolate between 2 poses

Parameter:

- IN POSE3D - Current POSE data structure

- EndValue POSE3D -- The end value POSE..
- T -- Value between 0 and 1.

Returns:

- Interpolated POSE3D - The interpolated POSE.

Pose3d_Log



Returns a Twist3d that maps this pose to the end pose. If c is the output of a.Log(b), then a.Exp(c) would yield b.

Parameters:

- IN POSE3D - The current POSE data structure
- OTHER POSE3D - The end pose for the transformation.

Returns:

- OUT TWIST3D - The twist that maps this to end.
- Error -- boolean -- If value is TRUE, an error occurred.

Pose3d_Minus



Returns the Transform3d that maps the one pose to another.

Parameter:

- IN POSE3D - Current POSE data structure

- OTHER POSE3D - The initial pose of the transformation.

Returns:

- OUT TRANSFORM3D - The transform that maps the other pose to the current pose.

Pose3d_NearestTo



Returns the nearest Pose3d from a list of poses. If two or more poses in the list have the same distance from this pose, return the one with the closest rotation component.

Parameter:

- IN POSE -- Pose3d -- The pose to compare against the OTHER POSES to find the closest pose.
- OTHER POSE S -- Pose3d array -- The list of poses to search for the closest pose.

Returns:

- Nearest Pose -- Pose3d -- The pose closest to IN POSE.
- Found -- boolean -- True if a POSE was found. This should only be false if the Other Pose array is empty.

Pose3d_New



Constructs a pose with x, y, and z translations instead of a separate Translation3d.

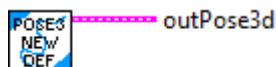
Parameters:

- X - The x component of the translational component of the pose.
- Y - The y component of the translational component of the pose.
- Z - The z component of the translational component of the pose.
- Rotation3d - The rotational component of the pose.

Returns:

- OUT POSE3D - Output POSE3D
-

Pose3d_New_Default



Constructs a pose at the origin facing toward the positive X axis.

Parameters:

-- none --

Return:

- POSE3d - Pose3d data structure
-

Pose3d_New_Pose2d



Constructs a 3D pose from a 2D pose in the X-Y plane

Parameters:

- in Pose2 -- Pose 2d in the X-Y plane to convert to Pose 3D.

Returns:

- OUT POSE3D - Output POSE3D

Pose3d_New_Trans3dRot3d



Constructs a pose with the specified translation and rotation.

Parameters:

- translation - The translational component of the pose.
- rotation - The rotational component of the pose.

Return:

- POSE3D - Pose data structure
-

Pose3d_Plus



Transforms the pose by the given transformation and returns the new transformed pose. "plus" is the same as "transformBy"

Parameters:

- IN POSE3D - The input POSE data structure
- OTHER TRANSFORM3D - The transform to transform the pose by.

Return:

- OUT POSE3D - The transformed pose.
-

Pose3d_RelativeTo



Returns the other pose relative to the current pose.

This function can often be used for trajectory tracking or pose stabilization algorithms to get the error between the reference and the current pose.

Parameters:

- IN POSE3D - The POSE data structure
- OTHER POSE3D - The pose that is the origin of the new coordinate frame that the current pose will be converted into.

Return:

- OUT POSE3D - The current pose relative to the new origin pose.

Pose3d_RotationVectorToMatrix



Applies the hat operator to a rotation vector.

It takes a rotation vector and returns the corresponding matrix representation of the Lie algebra element (a 3x3 rotation matrix).

Parameters:

- RotationVector -- rotation The rotation vector.

Return:

- RotationMatrix -- The rotation vector as a 3x3 rotation matrix.

Pose3d_Times



Multiply the pose by the given scalar and returns the new pose3d.

Parameters:

- IN POSE3D - The input POSE data structure
- Scalar - The value to multiply the pose by.

Return:

- OUT POSE3D - The output pose.

Pose3d_ToPose2d



Returns a Pose2d representing this Pose3d projected into the X-Y plane.

Parameter:

- IN POSE3D - Current POSE data structure

Returns:

- out POSE2D - A Pose2d representing this Pose3d projected into the X-Y plane.

Pose3d_TransformBy



Transforms the pose by the given transformation and returns the new transformed pose. "plus" is the same as "transformBy"

Parameters:

- IN POSE3D - The input POSE data structure
- OTHER TRANSFORM3D - The transform to transform the pose by.

Return:

- OUT POSE3D - The transformed pose.
-

Pose3d_getRotation3d



Returns the rotational component of the transformation.

Parameters:

- IN POSE3D -- Input POSE data structure

RETURNS:

- out ROTATION3D - The rotational component of the pose.
-

Pose3d_getTranslation3d



Returns the translation component of the transformation.

Parameters:

- IN POSE3D - The POSE data structure

RETURNS:

- OUT TRANSLATION3D - The translational component of the pose.
-

Pose3d_getXYZ



Returns the X, Y, Z elements of the translation component of the transformation.

Parameters:

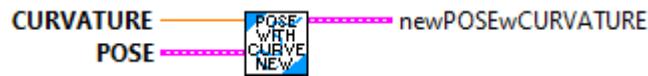
- IN POSE3D - The POSE data structure

RETURNS:

- X - The X element of the translational component of the pose.
- Y - The Y element of the translational component of the pose.
- Z - The Z element of the translational component of the pose.

PoseWithCurve

PoseWithCurve_New



Constructs a PoseWithCurvature.

Parameters:

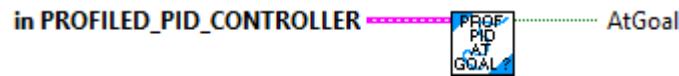
- pose - The pose. (Meters)
- curvature - The curvature. (Radians/Meter)

Returns:

- NewPoseWCurvature - The PoseWCurvature data structure

ProfiledPIDController

ProfiledPIDController_AtGoal



Returns true if the error is within the tolerance of the error.

This will return false until at least one input value has been computed.

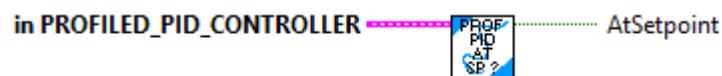
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- AtGoal -- Returns True if the error is within the defined error tolerance

ProfiledPIDController_AtSetpoint



Returns true if the error is within the tolerance of the error.

This will return false until at least one input value has been computed.

This uses the most recent error values calculated during the calculate call. This should be called after calculate.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- AtSetpoint -- Returns True if the error is within the defined error tolerance

ProfiledPIDController_Calculate_Meas



Returns the next output of the PID controller.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- measurement -- The current measurement of the process variable.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster
- CO-ControlOutput -- Calculated PID output

ProfiledPIDController_Calculate_Meas_Goal



Returns the next output of the PIDController.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

- measurement -- The current measurement of the process variable.
- goal -- The new goal of the controller.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster
- CO-ControlOutput -- Calculated PID output

ProfiledPIDController_Calculate_Meas_StateGoal



Returns the next output of the PID controller.

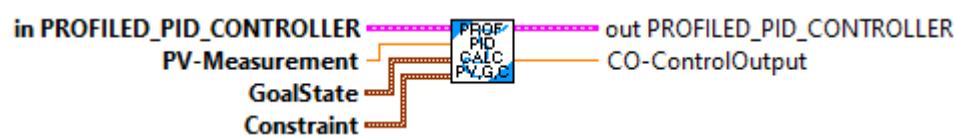
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- measurement -- The current measurement of the process variable.
- goal -- The new goal of the controller.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster
- CO-ControlOutput -- Calculated PID output

ProfiledPIDController_Calculate_Meas_StateGoal_TrapCnsrt



Returns the next output of the PID controller.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- measurement -- The current measurement of the process variable.
- goal -- The new goal of the controller.
- constraints -- Velocity and acceleration constraints for goal.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster
- CO-ControlOutput -- Calculated PID output

ProfiledPIDController_DisableContInput



Disables continuous input.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_EnableContInput



Enables continuous input.

Rather than using the max and min input range as constraints, it considers them to be the same point and automatically calculates the shortest route to the setpoint.

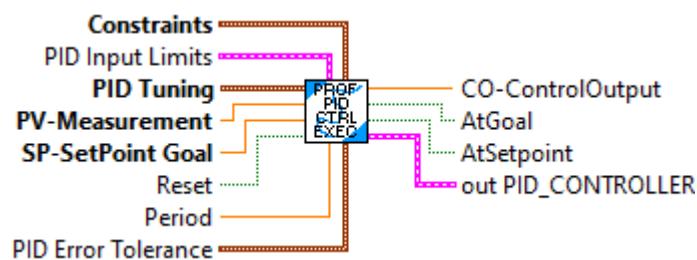
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- minimumInput -- The minimum value expected from the input.
- maximumInput -- The maximum value expected from the input.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_Execute



Convenience, single call, LabVIEW function. Creates and calculates the Profiled PID controller. Call this routine periodically to calculate the newest output for the provided inputs.

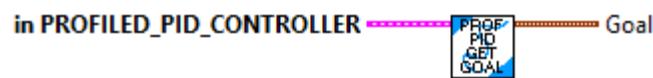
Inputs:

- Constraints -- Trapezoid Constraint cluster contains
 - Maximum Velocity
 - Maximum Acceleration
- PID Input Limits -- (Optional) Cluster containing:
 - MaxInput -- (Default: 0)
 - MinInput -- (Default: 0)
 - Continous -- When True indicates that the input is continuous. (Default: False)
- PID Tuning -- PID Tuning parameters containing:
 - Kp -- The proportional coefficient.
 - Ki -- The integral coefficient.
 - Kd -- The derivative coefficient.
 - Maximum Integral -- The maximum value of the integrator.
 - Minimum Integral -- The minimum value of the integrator.
- PV-measurement -- The current measurement of the process variable. (PV)
- SP-setpoint goal -- The new end goal setpoint of the controller. (SP)
- Reset -- When True, causes the PID to be reset. (Default: False)
- Period -- Period of repeated calls in seconds. (Default: 0.020)
- PID Error Tolerance -- Tolerances for determining "At Setpoint", contains:
 - Position Tolerance -- (Default: 0.001)
 - Velocity Tolerance -- (Default: 9.9E+30)

Outputs:

- CO-ControlOutput -- Controller output (CO)
- AtGoal -- True when PV is within the specified tolerances of the final goal SP.
- AtSetpoint -- True when PV is within the specified tolerances of the intermediate position SP.
- Profiler PID_Controller -- Updated PID_Controller data cluster

ProfiledPIDController_GetGoal



Gets the goal for the ProfiledPIDController.

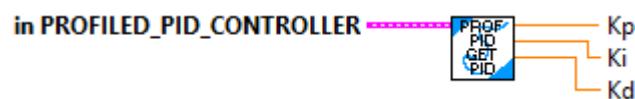
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- goal -- current goal as a Trapezoid Profile State data cluster

ProfiledPIDController_GetPID



Gets the proportional, integral, and derivative PID coefficients.

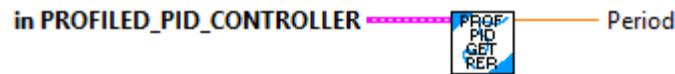
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- Kp -- proportional coefficient
- Ki -- integral coefficient
- Kd -- derivative coefficient

ProfiledPIDController_GetPeriod



Gets the period of this controller.

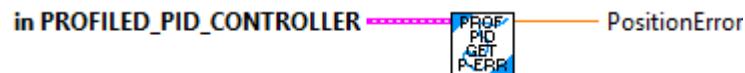
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- period -- The period of the controller. (seconds)

ProfiledPIDController_GetPositionError



Returns the difference between the setpoint and the measurement.

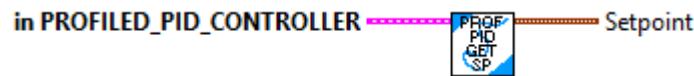
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- PositionError -- The error.

ProfiledPIDController_GetSetpoint



Returns the current setpoint of the ProfiledPIDController.

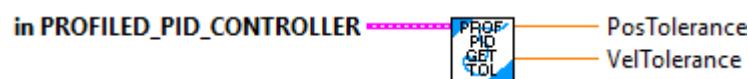
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- Setpoint -- The current setpoint, returned as a Trapezoid Profile State data cluster

ProfiledPIDController_GetTolerance



Returns the position and velocity tolerance values

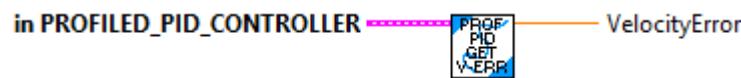
Inputs:

- Profiled PID_Controller -- Profiled PID_Controller data cluster

Outputs:

- PosTolerance -- The position tolerance
- VelTolerance -- The velocity tolerance

ProfiledPIDController_GetVelocityError



Returns the change in error per second.

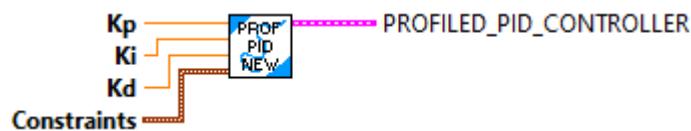
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster

Outputs:

- VelocityError -- The current velocity error

ProfiledPIDController_New



Allocates a ProfiledPIDController with the given constants for Kp, Ki, and Kd.

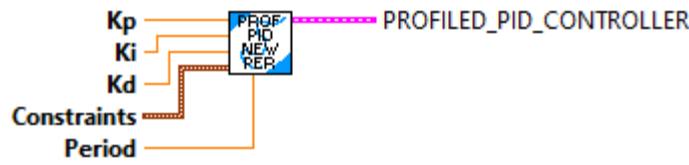
Inputs:

- Kp -- The proportional coefficient.
- Ki -- The integral coefficient.
- Kd -- The derivative coefficient.
- constraints -- Velocity and acceleration constraints for goal.

Outputs:

- Profiled_PID_Controller -- Initialized Profiled_PID_Controller data cluster

ProfiledPIDController_NewPeriod



Allocates a ProfiledPIDController with the given constants for Kp, Ki, and Kd.

Inputs:

- Kp The proportional coefficient.
- Ki The integral coefficient.
- Kd The derivative coefficient.
- constraints Velocity and acceleration constraints for goal.
- period The period between controller updates in seconds. The default is 0.02 seconds.

Outputs:

- Profiled_PID_Controller -- Initialized Profiled_PID_Controller data cluster

ProfiledPIDController_Reset



Reset the previous error and the integral term.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- measurement -- The current measured State of the system.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_Reset_PosOnly



Reset the previous error and the integral term.

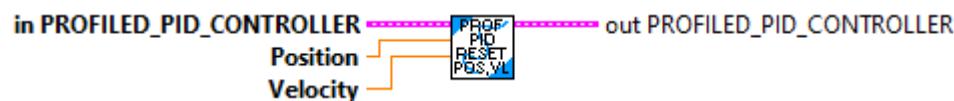
Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- Position -- The current measured position of the system. The velocity is assumed to be zero.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_Reset_PosVel



Reset the previous error and the integral term.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- Position -- The current measured position of the system.
- Velocity -- The current measured velocity of the system.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster
-
-

ProfiledPIDController_SetConstraints



Set velocity and acceleration constraints for goal.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- constraints -- Velocity and acceleration constraints for goal.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster
-
-

ProfiledPIDController_SetGoal



Sets the goal for the ProfiledPIDController.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- goal -- The desired goal state.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_SetGoal_PosOnly



Sets the goal for the ProfiledPIDController.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- goalPosition -- The desired goal position.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_SetIntegratorRange



Sets the minimum and maximum values for the integrator.

When the cap is reached, the integrator value is added to the controller

output rather than the integrator value times the integral gain.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- minimumIntegral -- The minimum value of the integrator.
- maximumIntegral -- The maximum value of the integrator.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_SetPID



Sets the PID Controller gain parameters.

Sets the proportional, integral, and differential coefficients.

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- Kp -- Proportional coefficient
- Ki -- Integral coefficient
- Kd -- Differential coefficient

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_SetTolerance_PosOnly



Sets the error which is considered tolerable for use with atSetpoint().

Inputs:

- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- positionTolerance -- Position error which is tolerable.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

ProfiledPIDController_SetTolerance_PosVel



Sets the error which is considered tolerable for use with atSetpoint().

Inputs:

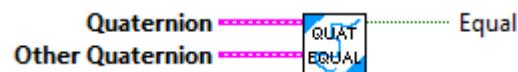
- Profiled_PID_Controller -- Profiled_PID_Controller data cluster
- positionTolerance -- Position error which is tolerable.
- velocityTolerance -- Velocity error which is tolerable.

Outputs:

- Profiled_PID_Controller -- Updated Profiled_PID_Controller data cluster

Quaternion

Quaternion_Equals



Checks equality between this Quaternion and another object.

Parameters:

- Quaternion - The quaternion data cluster.
- Other Quaternion - The other quaternion data cluster.

Returns:

- Equal - Returns TRUE if both quaternions are equal.
-
-

Quaternion_Get_All



Return the W, X, Y, and Z components of the quaternion

Parameters:

- Quaternion -- The quaternion data cluster.

Returns:

- W - Returns W component of the quaternion.
- X - Returns X component of the quaternion.
- Y - Returns Y component of the quaternion.
- Z - Returns Z component of the quaternion.

Quaternion_Get_LVQuat



Convert a Quaternion to a LabVIEW formatted quaternion data structure (array).

Parameters:

- Quaternion - The quaternion data cluster.

Returns:

- LV Quaternion - The LabVIEW quaternion

Quaternion_Get_Vect



Get the Vector component of the Quaternion data cluster

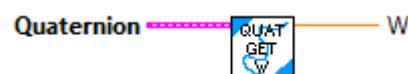
Input:

- Quaternion -- Quaternion data cluster

Output:

- Vector -- Vector component of the quaternion as an array.

Quaternion_Get_W



Return the W component of the quaternion

Parameters:

- Quaternion -- The quaternion data cluster.

Returns:

- W - Returns W component of the quaternion.

Quaternion_Inverse



Returns the inverse of the quaternion.

Parameters:

- Quaternion - Quaternion data cluster

Returns:

- Inverse - The inverse quaternion data cluster.

Quaternion_New



Constructs a quaternion with using the provided inputs.

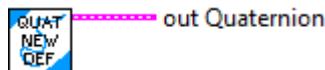
Parameters:

- w -- W component of the quaternion.
- x -- X component of the quaternion.
- y -- Y component of the quaternion.
- z -- Z component of the quaternion.

Returns:

- out Quaternion -- Created data cluster
-

Quaternion_New_Default



Constructs a quaternion with a default angle of 0 degrees

Parameters:

- None --

Returns:

- out Quaternion -- Created data cluster
-

Quaternion_New_LVQuat



Constructs a quaternion from a LabVIEW quaternion

Parameters:

- LV Quaternion -- LabVIEW quaternion data cluster

Returns:

- out Quaternion -- Created data cluster
-

Quaternion_Normalize



Normalizes the quaternion.

Parameters:

- Quaternion - The quaternion data cluster.

Returns:

- Normalize - The normalized quaternion data cluster.

Quaternion_Plus



Sum with another quaternion.

Parameters:

- Quaternion - The quaternion data cluster
- Other Quaternion - The other quaternion data cluster

Returns:

- Sum - The quaternion summation.

Quaternion_Times



Multiply with another quaternion.

Parameters:

- Quaternion - The quaternion data cluster

- Other Quaternion - The other quaternion data cluster

Returns:

- Times - The quaternion product.
-

Quaternion_ToRotationVector



Returns the rotation vector representation of this quaternion.

This is also the log operator of SO(3).

Parameters:

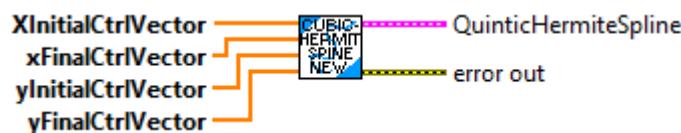
- Quaternion -- The quaternion data cluster.

Returns:

- Rotation Vector - The rotation vector representation of this quaternion.

QuinticHermiteSpline

QuinticHermiteSpline_New



Constructs a quintic hermite spline with the specified control vectors. Each control vector contains into about the location of the point, its first derivative, and its second derivative.

Parameters:

- xInitialControlVector The control vector for the initial point in the x dimension.
- xFinalControlVector The control vector for the final point in the x dimension.
- yInitialControlVector The control vector for the initial point in the y dimension.
- yFinalControlVector The control vector for the final point in the y dimension.

Returns:

- QuinticHermiteSpline - The resulting spline
- Error Out - The resulting error cluster

QuinticHermiteSpline_getControlVectorFromArrays



Returns the control vector for each dimension as a matrix from the user-provided arrays in the constructor.

Parameters:

- initialVector - The control vector for the initial point.
- finalVector - The control vector for the final point.

Returns:

- ControlVector - The control vector matrix for a dimension.
 - Error Out - The returned error cluster
-
-

QuinticHermiteSpline_makeHermiteBasis



Returns the hermite basis matrix for quintic hermite spline interpolation.

Parameters:

- none -

Returns:

- HermiteBasis - The hermite basis matrix for quintic hermite spline interpolation.

Ramsete

Ramsete_AtReference



Returns true if the pose error is within tolerance of the reference.

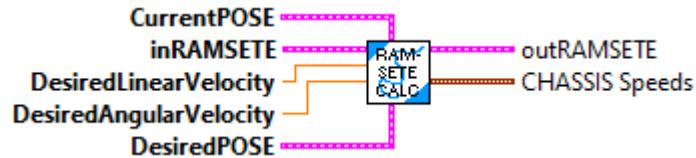
Parameters:

- Ramsete - Ramsete data structure

Returns:

- At Reference - Return value

Ramsete_Calculate



Returns the next output of the Ramsete controller.

The reference pose, linear velocity, and angular velocity should come from a drivetrain trajectory.

Parameters:

- InRamsete - Ramsete data structure
- currentPose - The current pose.
- poseRef - The desired pose.
- linearVelocity - The desired linear velocity in meters/sec.
- angularVelocity - The desired angular velocity in radians/second.

Returns:

- Out Ramsete -- Updated Ramsete data structure
 - Chassis Speeds
-

Ramsete_Calculate_Trajectory



Returns the next output of the Ramsete controller.

The reference pose, linear velocity, and angular velocity should come from a drivetrain trajectory.

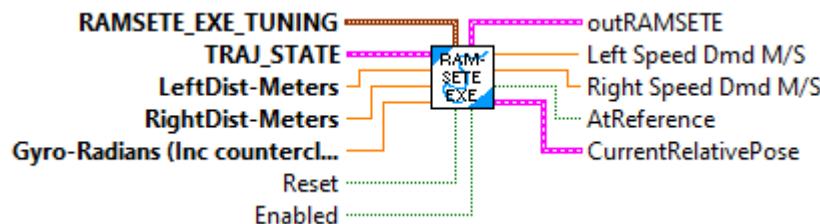
Parameters:

- InRamsete - Ramsete data structure
- TrajState - Desired trajectory state
- currentPose - The current pose.

Returns:

- Out Ramsete -- Updated Ramsete data structure
 - Chassis Speeds
-

Ramsete_Execute



Convenience function to create and calculate the ramsete control, returning normalized desired left and right speed setpoints

Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired pose consisting of x_d , y_d , and θ_d . We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVizi e le TEcnologie").

Parameters:

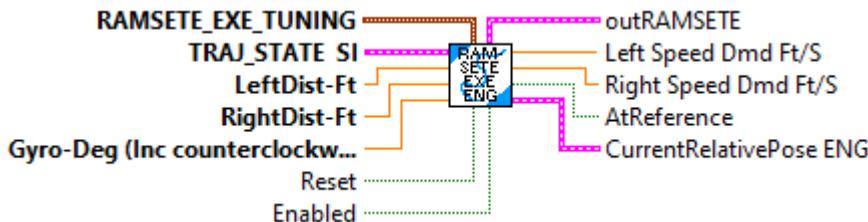
- Ramsete EXE Tuning -- Packed data cluster containing the ramsete tuning parameters and Differential Drive kinematics information.
- TrajState - Desired trajectory state
- Left Dist -- The left distance read from encoders (Meters). The encoders do NOT have to be reset at the beginning of trajectory execution.
- Right Dist -- The right distance read from encoders (Meters). The encoders do NOT have to be reset at the beginning of trajectory execution.
- Gyro - The current reading of a gyro whose value increases counter clockwise. (You may need to negate your gyro reading.) (Radians)
- Reset -- Resets the current robot pose and the beginning left and right distances and the gyro reading
- Enabled -- When enabled uses the ramsete closed loop control to drive the left and right speed demand outputs. When disabled the trajectory data is directly converted to left and right speed demands.

Returns:

- Left Speed Dmd -- Left Speed demand (Meters/Second)
- Right Speed Dmd -- Right Speed demand (Meters/Second)
- Out Ramsete -- Updated Ramsete data structure

- Current Relative Pose -- Current robot relative pose. This pose is reset to zero when the RESET input is set to TRUE.
 - At Reference -- Set to TRUE when the current robot pose is within the specified tolerance of the desired POSE from the trajectory state input.
-

Ramsete_Execute_ENG



Convenience function to create and calculate the ramsete control, returning normalized desired left and right speed setpoints

This is designed for ENG units -- Feet, degrees.

Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired pose consisting of x_d, y_d, and theta_d. We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVizi e le TECnologie").

Parameters:

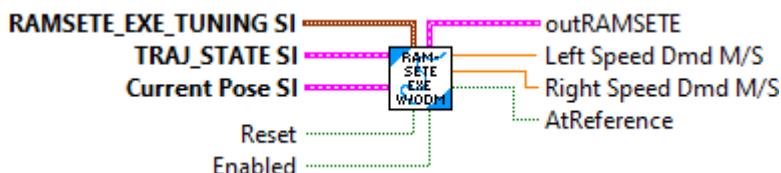
- Ramsete EXE Tuning -- Packed data cluster containing the ramsete tuning parameters and Differential Drive kinematics information.

- TrajState - Desired trajectory state
- Left Dist -- The left distance read from encoders (Feet). The encoders do NOT have to be reset at the beginning of trajectory execution.
- Right Dist -- The right distance read from encoders (Feet). The encoders do NOT have to be reset at the beginning of trajectory execution.
- Gyro - The current reading of a gyro whose value increases counter clockwise. (You may need to negate your gyro reading.) (Degrees)
- Reset -- Resets the current robot pose and the beginning left and right distances and the gyro reading
- Enabled -- When enabled uses the ramsete closed loop control to drive the left and right speed demand outputs. When disabled the trajectory data is directly converted to left and right speed demands.

Returns:

- Left Speed Dmd -- Left Speed demand (Feet/Second)
 - Right Speed Dmd -- Right Speed demand (Feet/Second)
 - Out Ramsete -- Updated Ramsete data structure
 - Current Relative Pose -- Current robot relative pose. This pose is reset to zero when the RESET input is set to TRUE.
 - At Reference -- Set to TRUE when the current robot pose is within the specified tolerance of the desired POSE from the trajectory state input.
-

Ramsete_Execute_Ext_Odom



Convenience function to create and calculate the ramsete control from current pose and trajectory state inputs, returning normalized desired left and right speed setpoints

This function is designed for SI units.

Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired pose consisting of x_d , y_d , and θ_d . We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVizi e le TEcnologie").

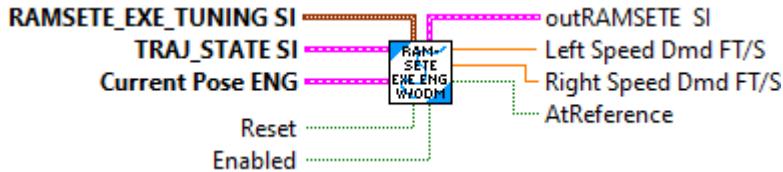
Parameters:

- Ramsete EXE Tuning -- Packed data cluster containing the ramsete tuning parameters and Differential Drive kinematics information. (SI Units)
- TrajState - Desired trajectory state (SI Units)
- Current Pose -- Current Pose2d robot location. (SI Units)
- Reset -- Resets the Ramsete data structure.
- Enabled -- When enabled uses the ramsete closed loop control to drive the left and right speed demand outputs. When disabled the trajectory data is directly converted to left and right speed demands.

Returns:

- Left Speed Dmd -- Left Speed demand (Meters/Second)
 - Right Speed Dmd -- Right Speed demand (Meters/Second)
 - Out Ramsete -- Updated Ramsete data structure (SI Units)
 - At Reference -- Set to TRUE when the current robot pose is within the specified tolerance of the desired POSE from the trajectory state input.
-
-

Ramsete_Execute_Ext_Odom_ENG



Convinience function to create and calculate the ramsete control from current pose and trajectory state inputs, returning normalized desired left and right speed setpoints

This function is designed for ENG units, with exception of the trajectory state which is SI units.

Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired pose consisting of x_d, y_d, and theta_d. We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVIZI e le TECnologie").

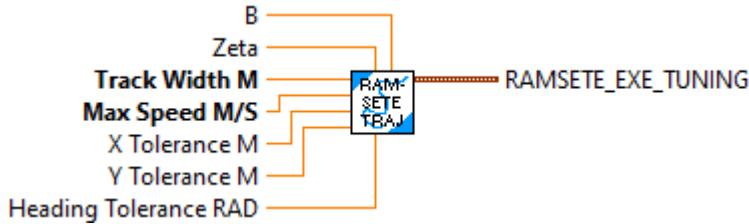
Parameters:

- Ramsete EXE Tuning -- Packed data cluster containing the ramsete tuning parameters and Differential Drive kinematics information. (SI Units)
- TrajState - Desired trajectory state (SI Units)
- Current Pose -- Current Pose2d robot location. (ENG Units)
- Reset -- Resets the Ramsete data structure.
- Enabled -- When enabled uses the ramsete closed loop control to drive the left and right speed demand outputs. When disabled the trajectory data is directly converted to left and right speed demands.

Returns:

- Left Speed Dmd -- Left Speed demand (Ft/Second)
 - Right Speed Dmd -- Right Speed demand (Ft/Second)
 - Out Ramsete -- Updated Ramsete data structure (SI Units)
 - At Reference -- Set to TRUE when the current robot pose is withing the specified tolerance of the desired POSE from the trajectory state input.
-
-

Ramsete_Execute_PackTuning



Packs tuning parameters for the Ramsete EXE VI (Which is designed for a differential drive robot)

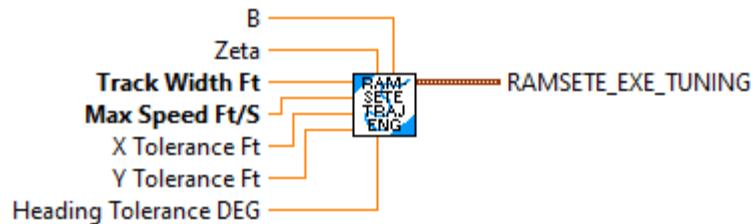
Inputs:

- b - Tuning parameter ($b > 0$), radians 2 / meters 2 , for which larger values make convergence more aggressive like a proportional term. (Default 2.0)
- zeta - Tuning parameter ($0 < zeta < 1$), / radians, for which larger values provide more damping in response. (Default 0.7)
- track width -- distance between right and left wheels on a differential drive robot (Meters)
- Max Speed -- The maximum wheel speed (Meters/Second)
- X Tolerance -- The desired allowable X position deviation (Meters). This is only used to calculate At Reference. (Default 0.051)
- Y Tolerance -- The desired allowable Y position deviation (Meters). This is only used to calculate At Reference. (Default 0.051)
- Heading Tolerance -- The desired allowable rotation deviation (Radians). This is only used to calculate At Reference. (Default 0.035)

Outputs:

- Ramsete EXE tuning -- Data cluster to be used as input to the Ramsete EXE VI.

Ramsete_Execute_PackTuning_ENG



Packs tuning parameters for the Ramsete EXE ENG VI (Which is designed for a differential drive robot)
This version is intended for ENG units (Feet, Degrees)

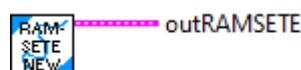
Inputs:

- b - Tuning parameter ($b > 0$), radians 2 / meters 2 , for which larger values make convergence more aggressive like a proportional term. (Default 2.0)
- zeta - Tuning parameter ($0 < zeta < 1$), / radians, for which larger values provide more damping in response. (Default 0.7)
- track width -- distance between right and left wheels on a differential drive robot (Feet)
- Max Speed -- The maximum wheel speed (Feet/Second)
- X Tolerance -- The desired allowable X position deviation (Feet). This is only used to calculate At Reference. (Default 0.083)
- Y Tolerance -- The desired allowable Y position deviation (Feet). This is only used to calculate At Reference. (Default 0.083)
- Heading Tolerance -- The desired allowable rotation deviation (Degrees). This is only used to calculate At Reference. (Default 1.0)

Outputs:

- Ramsete EXE tuning -- Data cluster to be used as input to the Ramsete EXE ENG VI.

Ramsete_New



Construct a Ramsete unicycle controller. The default arguments for b and zeta of 2.0 and 0.7 have been well-tested to produce desireable results.

Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired pose consisting of x_d , y_d , and θ_d . We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVIZI e le TECnologie").

Ramsete_New_B_Z



Construct a Ramsete unicycle controller.

Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired

pose consisting of x_d , y_d , and θ_d . We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVizi e le TEcnologie").

Parameters:

- b - Tuning parameter ($b > 0$), radians 2 / meters 2 , for which larger values make convergence more aggressive like a proportional term. (Default 2.0)

- zeta - Tuning parameter ($0 < zeta < 1$), / radians, for which larger values provide more damping in response. (Default 0.7)

Returns:

- out Ramsete - Ramsete data structure

Ramsete_SINC



Returns $\sin(x) / x$

When x is close to zero a special approximation calculation is used. Otherwise $\sin(x)/x$ is used.

Parameters:

- X - Input value

Ramsete_SetEnabled



Enables and disables the controller for troubleshooting purposes.

Parameters:

- InRamsete - Ramsete data structure

- enabled - Set TRUE to enable closed loop Ramsete control.

Returns:

- OutRamsete - Updated Ramsete data structure
-

Ramsete_SetTolerance



Sets the pose error which is considered tolerable for use with atReference().

Parameters:

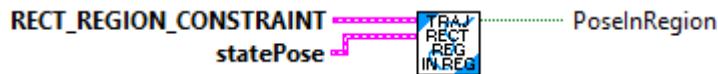
- in Ramsete - Ramsete data structure
- poseTolerance - Pose error which is tolerable.

Returns:

- out Ramsete - Updated Ramsete data structure

RectRegionConstraint

RectRegionConstraint_IsPoseInRegion



Determines if the robot pose is within the defined region.

Parameters:

- Rectangular Region Constraint -- Constraint data cluster.
- statePose - current traj state Pose

Returns

- PoseInRegion -- TRUE if pose is within the region.

RectRegionConstraint_New



Constructs a new EllipticalRegionConstraint.

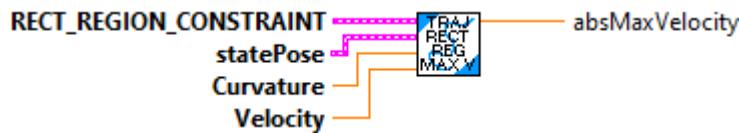
Parameters:

- BottomLeftPoint -- Translation holding the bottom left point of the rectangle.
- TopRightPoint -- Translation holding the top right point of the triangle.
- constraint -- The constraint to enforce when the robot is within the region.

Returns

- Rectangular Region Constraint - Constraint data structure.

RectRegionConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

Ensure each individual normalized wheel speed is within the defined maximum velocity.

Parameters:

- RectRegionConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

RectRegionConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values.

Parameters:

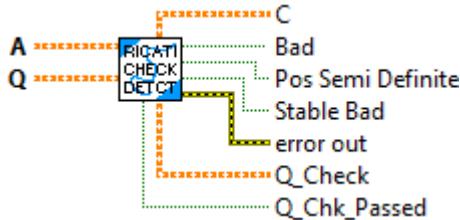
- RectRegionConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

Riccati

Riccati_Check_Detectable



This function is for internal use, by the riccati_input_check, ONLY.

This function check if (A,C) is a detectable pair, where $Q = C' * C$. (A,C) is detectable if and only if the unobservable eigenvalues of A , if any, have absolute values less than one, where an eigenvalue is unobservable if $\text{Rank}[\lambda I - A; C] < n$. Also, (A,C) is detectable if and only if (A',C') is stabilizable.

Inputs:

- A -- A matrix
- Q -- Q matrix

Outputs:

- Bad -- Returns TRUE, if the A,Q pair is not detectable or an error occurred.
- Error -- Error cluster that returns an error if A, Q is not detectable or some other error occurred.

Riccati_Check_Stabilizable



This function is for internal use, by the riccati_input_check, ONLY.

This function checks if (A,B) is a stabilizable pair. (A,B) is stabilizable if and only if the uncontrollable eigenvalues of A , if any, have absolute values less than one, where an eigenvalue is uncontrollable if $\text{Rank}[\lambda I - A, B] < n$. (n is the number of rows or columns of A , which must be a square matrix.)

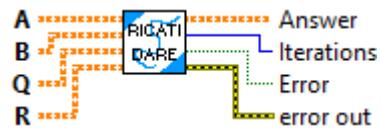
Inputs:

- A -- A matrix
- B -- B matrix

Outputs:

- Bad -- Returns TRUE, if the A,B pair is not stabilizable or an error occurred.
 - Error -- Error cluster that returns an error if A, B is not stabilizable or some other error occurred.
-

Riccati_DARE



Solves the discrete algebraic Riccati equation.

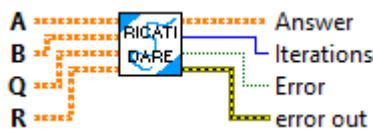
Input:

- A -- System matrix.
- B -- Input matrix.
- Q -- State cost matrix.
- R -- Input cost matrix.

Outputs:

- Answer -- Solution of DARE.
 - iterations -- Number of iterations
 - error -- If TRUE, an error occurred
-

Riccati_DARE_Choose



Solves the discrete algebraic Riccati equation.

Input:

- A -- System matrix.
- B -- Input matrix.
- Q -- State cost matrix.
- R -- Input cost matrix.

Outputs:

- Answer -- Solution of DARE.
- iterations -- Number of iterations
- error -- If TRUE, an error occurred

Riccati_DARE_Iterate



Solves the discrete algebraic Riccati equation.

This uses an iterative method to arrive at the solution. Run time is not deterministic and this routine should not be used during run-time, only during initialization.

Input:

- A -- System matrix.
- B -- Input matrix.

- Q -- State cost matrix.
- R -- Input cost matrix.

Outputs:

- Answer -- Solution of DARE.
- iterations -- Number of iterations
- error -- If TRUE, an error occurred

Riccati_DARE_N



Solves the discrete algebraic Riccati equation.

Input:

- A -- System matrix.
- B -- Input matrix.
- Q -- State cost matrix.
- R -- Input cost matrix.
- N -- State-input cross-term cost matrix.

Outputs:

- Answer -- Solution of DARE.
- iterations -- Number of iterations
- error -- If TRUE, an error occurred

See https://en.wikipedia.org/wiki/Linear%20quadratic_regulator#Infinite-horizon,_discrete-time_LQR for the change of variables used here.

Riccati_DARE_StructDoubling



Solves the discrete algebraic Riccati equation using the structured doubling method.

This uses an iterative method to arrive at the solution. While this is a quick algorithm, its run time is not deterministic and this routine should not be used during run-time, only during initialization.

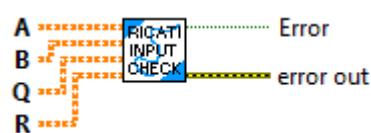
Input:

- A -- System matrix.
- B -- Input matrix.
- Q -- State cost matrix.
- R -- Input cost matrix.

Outputs:

- Answer -- Solution of DARE.
- iterations -- Number of iterations
- error -- If TRUE, an error occurred

Riccati_Input_Check



This is an internal function, called only by the DARE routines.

This routine checks the inputs for validity prior to calling DARE.

Inputs

- A -- A matrix
- B -- B matrix
- Q -- Q matrix
- R -- R matrix

Output:

- Error -- If TRUE, the input checks failed or an internal error occurred.
- Error out -- Error cluster

Rotation2d

Rotation2d_CreateAngle



Constructs a Rotation2d with the given radian value. The x and y don't have to be normalized.

Parameters:

- value - The value of the angle in radians.

Returns

- rotation data structure

Rotation2d_CreateAngleDegrees



Constructs a Rotation2d with the given angle value in degrees.

Parameters:

- value - The value of the angle in degrees..

Returns

- rotation data structure

Rotation2d_CreateAngleRotations



Constructs a Rotation2d with the given angle value in rotations.

Parameters:

- value - The value of the angle in rotations.

Returns

- rotation data structure
-
-

Rotation2d_CreateXY



Constructs a Rotation2d with the given x and y (cosine and sine) components. X and Y don't have to be normalized.

Parameters:

- X -The x component or cosine of the rotation.
- Y - The y component or sine of the rotation.

Returns:

Rotation data structure

Rotation2d_Div



Divides the current rotation (angle) by a scalar.

Parameters:

- IN ROTATION - This ROTATION data structure
- SCALAR - The value to divide the rotation by.

Returns:

- OUT ROTATION - The new ROTATION data structure.
-

Rotation2d_Equals



Checks equality between this Rotation2d and another object.

Parameters:

- in ROTATION - This Rotation data structure
- other ROTATION - The other Rotation to compare

Returns:

- equals - Whether the two objects are equal or not.
-

Rotation2d_GetAngleCosSin



Returns the angle, sine, and cosine of the rotation.

Parameters:

- in ROTATION - This ROTATION data structure

Returns:

- Angle - The angle of the ROTATION (radians)
- Cosine of the ROTATION

- Sine of the ROTATAION
-

Rotation2d_GetCos



Returns the cosine of the rotation.

Parameters:

- in ROTATION - This ROTATION data structure

Returns:

- Cosine of the ROTATAION
-

Rotation2d_GetDegrees



Get the angle in degrees from the rotation.

Parameters:

- ROTATION - The rotation data structure

Results:

- Angle in degrees.
-

Rotation2d_GetRadians



Get the angle in radians from the rotation.

Parameters:

- ROTATION - The rotation data structure

Results:

- Angle in radians.

Rotation2d_GetRotations



Get the angle in degrees from the rotation.

Parameters:

- ROTATION - The rotation data structure

Results:

- Angle in rotations.

Rotation2d_GetSin



Returns the sine of the rotation.

Parameters:

- in ROTATION - This ROTATION data structure

Returns:

- Sine of the ROTATAION

Rotation2d_GetTan



Returns the tangent of the rotation.

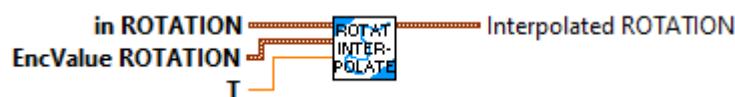
Parameters:

- in ROTATION - This ROTATION data structure

Returns:a

- Tangent of the ROTATAION

Rotation2d_Interpolate



Interpolate between 2 rotations

Parameters:

- in ROTATION - This rotation data structure
- EndValue ROTATION - The end value rotation
- T -- Value between 0 and 1.

Returns:

- Interpolated ROTATION - The interpolated rotation.

Rotation2d_Minus



Subtracts the new rotation from the current rotation and returns the new rotation.

For example, `Rotation2d.fromDegrees(10) - Rotation2d.fromDegrees(100) = Rotation2d{-pi/2}`

Parameters:

- in ROTATION - This rotation data structure
- other ROTATION - The rotation to subtract.

Returns:

- out ROTATION - The difference between the two rotations.

Rotation2d_Plus



Adds two rotations together, with the result being bounded between -pi and pi.

For example, `Rotation2d.fromDegrees(30) + Rotation2d.fromDegrees(60) = Rotation2d{-pi/2}`

This function is the same as RotateBy.

Parameters:

- IN ROTATION - This rotation data structure
- OTHER ROTATION - The rotation to add.

Returns:

- OUT ROTATION - The sum of the two rotations.

Rotation2d_RotateBy



Adds the new rotation to the current rotation using a rotation matrix.

The matrix multiplication is as follows:

$$[\cos_{\text{new}}] = [\text{other.cos}, -\text{other.sin}][\cos]$$

$$[\sin_{\text{new}}] = [\text{other.sin}, \text{other.cos}][\sin]$$

$$\text{value_new} = \text{atan2}(\cos_{\text{new}}, \sin_{\text{new}})$$

This is the same as Plus.

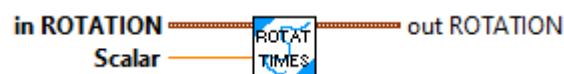
Parameters:

- IN ROTATION - This ROTATION data structure
- OTHER ROTATION - The rotation to rotate by

Returns:

- OUT ROTATION - The new rotated Rotation2d.

Rotation2d_Times



Multiples the current rotation (angle) by a scalar.

Parameters:

- IN ROTATION - This ROTATION data structure
- SCALAR - The value to multiply the rotation by.

Returns:

- OUT ROTATION - THe new ROTATION data structure.
-

Rotation2d_UnaryMinus



Takes the inverse of the current rotation. This is simply the negative of the current angular value.

Parameters:

- IN ROTATION - This ROTATION data structure

Results:

- OUT ROTATION - The negated ROTATION data structure.

Rotation3D

Rotation3D_Create_RotMatrix



Constructs a Rotation3d from a rotation matrix.

Inputs::

-- rotationMatrix -- The rotation matrix

Returns:

-- Rotation3d -- The created Rotation3d data cluster

-- Error -- TRUE indicates an error occurred.

Rotation3D_Create_RotVector



Constructs a Rotation3d from a rotation vector (3,1).

Inputs::

-- rotationVector -- The rotation vector

Returns:

-- Rotation3d -- The created Rotation3d data cluster

-- Error -- TRUE indicates an error occurred.

Rotation3d_Create_AxisAngle



Constructs a Rotation3d with the given axis-angle representation. The axis doesn't have to be normalized.

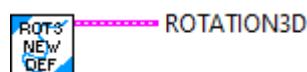
Parameters:

- Axis -- The rotation axis.
- Angle -- The rotation around the axis in radians.

Returns

-- Rotation3d -- The resulting rotation3d data cluster.

Rotation3d_Create_Default



Constructs a Rotation2d with the given radian value. The x and y don't have to be normalized.

Parameters:

- value - The value of the angle in radians.

Returns

- rotation data structure

Rotation3d_Create_InitialFinalVector



Constructs a Rotation3d that rotates the initial vector onto the final vector.

This is useful for turning a 3D vector (final) into an orientation relative to a coordinate system vector (initial).

Parameters:

- initial -- The initial vector.
- last -- The final vector

Returns

- rotation data structure
-

Rotation3d_Create_Quaternion



Constructs a Rotation3d from a quaternion.

Parameters:

- Quaternion - The quaternion to convert to Rotation3d.

Returns

- Rotation3d -- Rotation3d data structure
-

Rotation3d_Create_RollPitchYaw



Constructs a Rotation3d from extrinsic roll, pitch, and yaw.

Extrinsic rotations occur in that order around the axes in the fixed global frame rather than the body frame.

Parameters:

- Roll -- The counterclockwise rotation angle around the X axis (roll) in radians.
- Pitch -- The counterclockwise rotation angle around the Y axis (pitch) in radians.
- Yaw -- The counterclockwise rotation angle around the Z axis (yaw) in radians.

Returns

- Rotation3d -- The resulting rotation3d data cluster

Rotation3d_Div



Divides the current rotation by a scalar.

Parameters:

- IN ROTATION3D - This ROTATION3D data structure
- SCALAR - The value to divide the rotation by.

Returns:

- OUT ROTATION3D - THe new ROTATION3D data structure.

Rotation3d_Equals



Checks equality between this Rotation3d and another object.

Parameters:

- IN ROTATION3D -- This rotation3d data structure
- OTHER ROTATION3D -- The rotation3d to compare.

Returns:

- Equals -- Returns TRUE if the two rotation3ds are equals.
-

Rotation3d_GetAxisAngle



Returns the axis in the axis-angle representation of this rotation and the angle in radians in the axis-angle representation of this rotation.

Parameters:

- in ROTATION3D - This ROTATION3D data structure

Returns:

- Axis -- The axis in the axis-angle representation.
 - Angle -- The angle in radians in the axis-angle representation of this rotation
-

Rotation3d_GetQuaterion



Returns the quaternion representation of the Rotation3d.

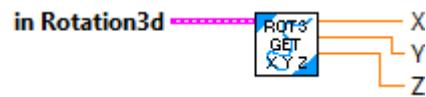
Parameters:

- in ROTATION3D - This ROTATION3D data structure

Returns:

- out Quaternion -- The quaternion representation of the Rotation3d.

Rotation3d_GetXYZ



Returns the counterclockwise rotation angle around the X axis (roll) in radians and the counterclockwise rotation angle around the Y axis (pitch) in radians and the counterclockwise rotation angle around the Z axis (yaw) in radians.

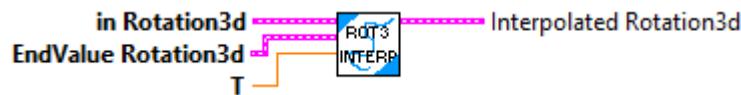
Parameters:

- in ROTATION3D - This ROTATION3D data structure

Returns:

- X -- The counterclockwise rotation angle around the X axis (roll) in radians.
- Y -- The counterclockwise rotation angle around the Y axis (pitch) in radians.
- Z -- The counterclockwise rotation angle around the Z axis (yaw) in radians.

Rotation3d_Interpolate



Interpolates between 2 Rotation3d.

Parameters:

- IN ROTATION3D -- This rotation3d data structure
- End Value ROTATION3D -- The end rotation3d
- T -- Interpolation value between 0 and 1.

Returns:

- Interpolated Rotation3d -- The Rotation3d interpolated between the 2 provided interpolations.

Rotation3d_Minus



Subtracts the new rotation from the current rotation and returns the new rotation.

Parameters:

- in ROTATION3D - This rotation data structure
- other ROTATION3D - The rotation to subtract.

Returns:

- out ROTATION3D - The difference between the two rotations.

Rotation3d_Plus



Adds two rotations together.

This function is the same as RotateBy.

Parameters:

- IN ROTATION3D - This rotation data structure
- OTHER ROTATION3D - The rotation to add.

Returns:

- OUT ROTATION3D - The sum of the two rotations.

Rotation3d_RotateBy



Adds the new rotation to the current rotation.

This is the same as Plus.

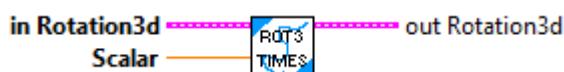
Parameters:

- IN ROTATION3D - This ROTATION data structure
- OTHER ROTATION3D - The rotation to rotate by

Returns:

- OUT ROTATION3D - The new rotated Rotation3d.

Rotation3d_Times



Multiplies the current rotation by a scalar.

Parameters:

- IN ROTATION3D - This ROTATION3D data structure
- SCALAR - The value to multiply the rotation by.

Returns:

- OUT ROTATION3D - The new ROTATION3D data structure.

Rotation3d_ToRotation2d



Returns a Rotation2d representing this Rotation3d projected into the X-Y plane

Parameters:

- IN ROTATION3D -- This ROTATION3D data structure

Results:

-- out Rotation2d -- A Rotation2d representing this Rotation3d projected into the X-Y plane.

Rotation3d_UnaryMinus



Takes the inverse of the current rotation.

Parameters:

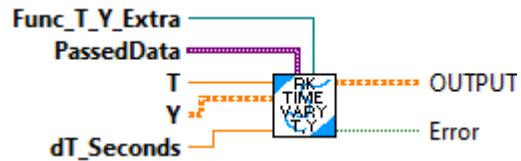
- IN ROTATION3D - This ROTATION3D data structure

Results:

- OUT ROTATION3D - The negated ROTATION3D data structure.

RungeKuttaTimeVarying

RungeKuttaTimeVarying_Rk4_Mat_T_Y



Performs 4th order Runge-Kutta integration of $dx/dt = f(t, \text{matrix } y)$ for dt .

Inputs:

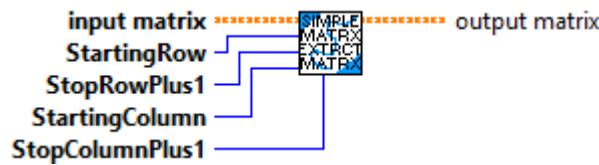
- Func_T_Y_Extra -- A strictly typed LabVIEW reference to a function to integrate as a function of matrix T, Y, and the extra passed data
- PassedData -- This is a variant that can be used to pass additional data to the function if needed. The data must be packed into the variant in the specific way expected by the function.
- T -- The initial value of T
- Y -- The initial value of Y.
- dT -- The time over which to integrate

Outputs:

- OUTPUT -- The integration of $dx/dt = f(t, y)$ for dt .
- Error -- If TRUE, an error occurred.

SimpleMatrix

SimpleMatrix_ExtractMatrix



Creates a new matrix which is a submatrix of this matrix.

$$s_{ij} = o_{ij} \text{ for all } y_0 < i < y_1 \text{ and } x_0 < j < x_1$$

where 's_{ij}' is an element in the submatrix and 'o_{ij}' is an element in the original matrix.

NOTE -- WPILIB has two functions called ExtractMatrix... be careful about the input parameters which are different. This one is SimpleMatrix.ExtractMatrix...

Inputs:

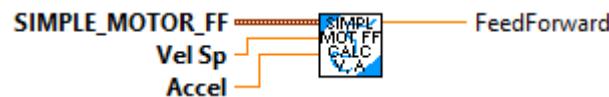
- Input Matrix -- The input from which to extract a subset
- StartingRow -- Starting row
- EndRowPlus1 -- Ending row + 1
- StartingColumn -- Starting column
- StopColumnPlus1 -- Ending column + 1

Outputs:

- Output Matrix -- The resulting extracted matrix

SimpleMotorFF

SimpleMotorFF_Calculate



Calculates the feedforward from the gains and setpoints.

Parameters:

- SimpleMotorFF -- Data structure containing constants
- velocity Setpoint -- The velocity setpoint.
- acceleration -- The setpoint acceleration. (Note that this could also be calculated from desired velocity, previous desired velocity (or actual velocity), and cycle time.)

Returns:

- FeedForward - The computed feedforward.

SimpleMotorFF_CalculateVelocityOnly



Calculates the feedforward from the gains and setpoints. (acceleration is assumed to be zero)

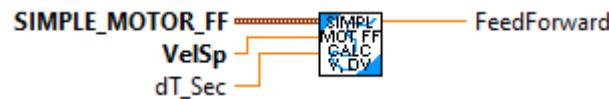
Parameters:

- SimpleMotorFF - Data structure containing constants
- velocity -- The velocity setpoint.

Returns:

- FeedForward - The computed feedforward.

SimpleMotorFF_Calculate_CalcAccel



Calculates the feedforward from the gains and setpoints. Instead of Acceleration Setpoint being supplied as a setpoint, the acceleration is calculated based on the change in the Velocity setpoint.

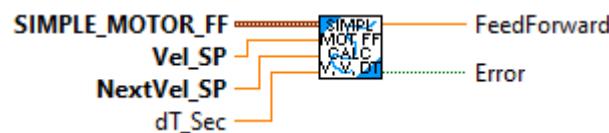
Parameters:

- SimpleMotorFF -- Data structure containing constants
- velocity Setpoint -- The velocity setpoint.
- dt -- The delta time (Seconds), default is 0.02 seconds.

Returns:

- FeedForward - The computed feedforward.

SimpleMotorFF_Calculate_NextV_Dt



Calculates the feedforward from the gains and setpoints. This VI calculates the acceleration by using a Linear System and a Linear Plant Inversion Feedforward. These systems are only created upon the first call or when one of the Kv or Ka values change.

Input parameters:

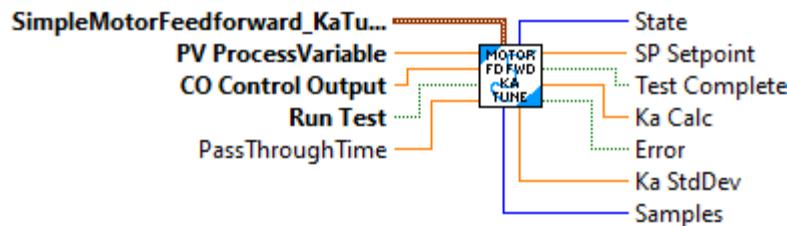
- SimpleMotorFF - Data structure containing constants
- Vel_SP -- The current velocity setpoint. (The velocity SP that was used last scan cycle)
- nextVel_SP -- The next velocity setpoint. (The velocity SP that we are using this scan cycle.)

- dt_Sec -- Time between velocity setpoints in seconds.

Returns:

- FeedForward - The computed feedforward.
- Error -- If TRUE, an error occurred calculating the acceleration value.

SimpleMotorFF_Ka_AutoTune



This function will ramp the SP up and down while collecting data. Once the ramp is complete Ka will be calculated.

Inputs:

- PackedParams -- SimpleMotorFF_KA_Tune_Params -- Data cluster containing the parameters defining the test.
- PV ProcessVariable -- Double -- The measured variable.
- CO Control Output -- Double -- The actuator output demand
- Run Test -- Boolean -- When TRUE, run the test.
- Pass Through Time -- Double -- Current time (seconds). If not specified, the FPGA time will be read and used.

Outputs:

- SP Setpoint -- Double - The desired while performing the test. When not testing the value is set to zero.
- Test Complete -- Boolean -- TRUE indicates the test has completed and a new Ka value has been calculated.

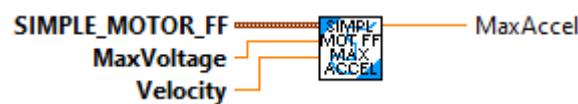
-- Ka Calc -- Double -- Newly calculated Ka value. Only valid when Test Complete is TRUE. This value is averaged from a number of individual Ka calculations made between the Min and Max eval SP values.

-- Error -- Boolean -- TRUE indicates an error has occurred.

-- Ka Std Dev -- Double -- Standard deviation of the array of Ka used to calculate the average Ka.

-- Samples -- Double -- The number of samples used to calculate Ka.

SimpleMotorFF_MaxAchieveAccel



Calculates the maximum achievable acceleration given a maximum voltage supply and a velocity. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the velocity constraint, and this will give you a simultaneously-achievable acceleration constraint.

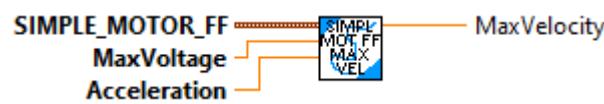
Parameters:

- Simple_Motor_FF - Data structure containing constants
- maxVoltage - The maximum voltage that can be supplied to the motor.
- velocity - The velocity of the motor.

Returns:

MaxAccel - The maximum possible acceleration at the given velocity.

SimpleMotorFF_MaxAchieveVel



Calculates the maximum achievable velocity given a maximum voltage supply and an acceleration. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the acceleration constraint, and this will give you a simultaneously-achievable velocity constraint.

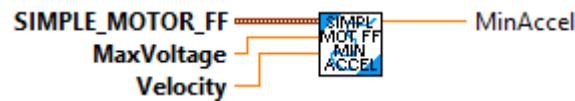
Parameters:

- Simple_Motor_FF - Data structure containing constants
- maxVoltage - The maximum voltage that can be supplied to the motor.
- Acceleration - The acceleration of the motor.

Returns:

MaxVelocity - The maximum possible velocity at the given acceleration.

SimpleMotorFF_MinAchieveAccel



Calculates the minimum achievable acceleration given a maximum voltage supply and a velocity. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable - enter the velocity constraint, and this will give you a simultaneously-achievable acceleration constraint.

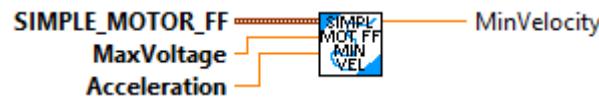
Parameters:

- Simple_Motor_FF - Data structure containing constants
- maxVoltage - The maximum voltage that can be supplied to the motor.
- velocity - The velocity of the motor.

Returns:

- MinAccel - The minimum possible acceleration at the given velocity.

SimpleMotorFF_MinAchieveVel



Calculates the minimum achievable velocity given a maximum voltage supply and an acceleration. Useful for ensuring that velocity and acceleration constraints for a trapezoidal profile are simultaneously achievable
- enter the acceleration constraint, and this will give you a simultaneously-achievable velocity constraint.

Parameters:

- Simple_Motor_FF - Data structure containing constants
- maxVoltage - The maximum voltage that can be supplied to the motor.
- Acceleration - The acceleration of the motor.

Returns:

- MinVelocity - The minimum possible velocity at the given acceleration.

SimpleMotorFF_New



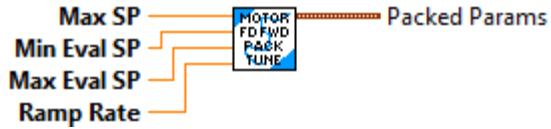
Creates a new SimpleMotorFeedforward with the specified gains. Units of the gain values will dictate units of the computed feedforward. (Generally units should be in meters to match other internal routines.)

These can be obtained by using the FRC Characterization routine to test your robot.

Parameters

- ks -- The static gain. (volts)
- kv -- The velocity gain. (volts seconds/meter)
- ka -- The acceleration gain. (volts seconds² / meter)
- Kn -- The normalization constant. When using the advanced PID, this constant helps match the output of this routine to input of the advanced PID.

SimpleMotorFF_Pack_Ka_Tune_Params



Pack individual parameters into the data cluster used by the Simple Motor Feed Forward Ka Auto Tune function. This function will ramp the SP up and down while collecting data. Once the ramp is complete Ka will be calculated.

Inputs:

- Max Sp -- Double -- The maximum value for the setpoint when ramping.
- Min Eval SP -- Double -- The lowest setpoint value to use when evaluating Ka
- Max Eval SP -- Double -- The highest setpoint value to use when evaluating Ka
- Ramp Rate -- Double -- The rate at which to ramp the setpoint (Units/Sec)

Outputs:

- PackedParams -- SimpleMotorFF_KA_Tune_Params -- The packed data cluster.

SlewRateLimiter

SlewRateLimiter_Calculate



Filters the input to limit its slew rate.

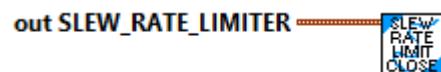
Inputs:

- in Slew Rate Limiter -- The Slew Rate Limiter data cluster
- input -- The input value whose slew rate is to be limited.

Outputs:

- out Slew Rate Limiter -- The modified Slew Rate Limiter data cluster
- output -- The filtered value, which will not change faster than the slew rate.

SlewRateLimiter_Close



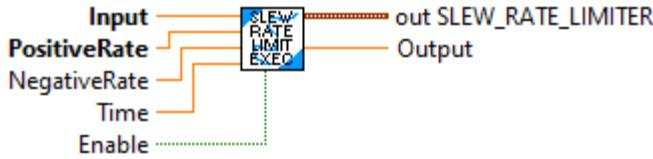
Releases resources used by a SlewRateLimiter data cluster.

(Since the Timer was removed, this no longer needs to be called. This is now a do nothing function.)

Inputs:

- in Slew Rate Limiter -- The Slew Rate Limiter data cluster

SlewRateLimiter_Execute



Convenience, single call, LabVIEW function. Creates and calculates the slew rate limiter function to filter the input to limit its slew rate. The optional enable parameter bypasses the filter calculation when "enabled" is false. When enabled is "false" the input is passed directly to the output and the internal buffer is continually filled with the current input value.

This limits the rate of change of an input value. Useful for implementing voltage, setpoint, and/or output ramps. A slew-rate limit is most appropriate when the quantity being controlled is a velocity or a voltage; when controlling a position, consider using a postion control algoritm or trapezoid profile instead.

Inputs:

- input -- The input value whose slew rate is to be limited.
- Positive Rate -- The new positive rate limit in units/second. This is expected to be ≥ 0 .
- Negative Rate -- The new negative rate limit in units/second. This is expected to be ≤ 0 . Optional. If not wired or the value is > 0 then the Positive Rate * -1 is used.
- enable -- Perform the rate limit calcuation when True. When False, output the input value unchanged. (Default=True)

Outputs:

- out Slew Rate Limiter -- The modified Slew Rate Limiter data cluster
- output -- The filtered value, which will not change faster than the slew rate.

SlewRateLimiter_GetRate



Gets the current rate limit value.

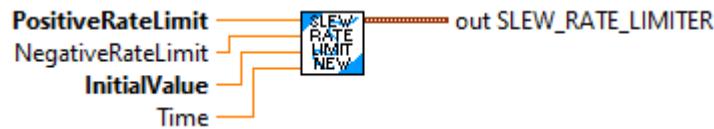
Inputs:

- in Slew Rate Limiter -- The Slew Rate Limiter data cluster

Outputs:

- PositiveRate -- The current positive rate limit in units/second. This is expected to be ≥ 0 .
- NegativeRate -- This current negative rate limit in units/second. This is expected to be ≤ 0 .

SlewRateLimiter_New



Creates a new SlewRateLimiter data cluster with the given rate limit and initial value.

This set of functions limits the rate of change of an input value. Useful for implementing voltage, setpoint, and/or output ramps. A slew-rate limit is most appropriate when the quantity being controlled is a velocity or a voltage; when controlling a position, consider using a position control algorithm or trapezoid profile instead.

Inputs:

- PositiveRateLimit -- The positive rate-of-change limit, in units per second.

The value is expected to be ≥ 0 .

- NegativeRateLimit -- The negative rate-of-change limit, in units per second.

The value is expected to be ≤ 0 . If the value is not wired or is > 0 then

the PositiveRateLimit * -1 is used.

- initialValue -- The initial value of the input.

Outputs:

- out Slew Rate Limiter -- The initialized Slew Rate Limiter data cluster
-

SlewRateLimiter_NewInitialZero

Creates a new SlewRateLimiter data cluster with the given rate limit. The initial value is set to zero.

This set of functions limits the rate of change of an input value. Useful for implementing voltage, setpoint, and/or output ramps. A slew-rate limit is most appropriate when the quantity being controlled is a velocity or a voltage; when controlling a position, consider using a postion control algoritm or trapezoid profile instead.

Inputs:

- PositiveRateLimit -- The positive rate-of-change limit, in units per second.

The value is expected to be ≥ 0 .

- NegativeRateLimit -- The negative rate-of-change limit, in units per second.

The value is expected to be ≤ 0 . If the value is not wired or is > 0 then

the PositiveRateLimit * -1 is used.

Outputs:

- out Slew Rate Limiter -- The initialized Slew Rate Limiter data cluster
-

SlewRateLimiter_Reset



Resets the slew rate limiter to the specified value; ignores the rate limit when doing so.

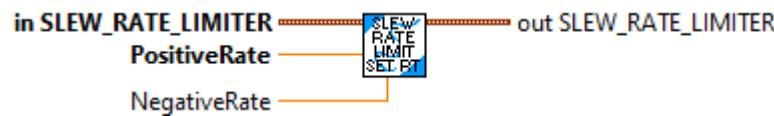
Inputs:

- in Slew Rate Limiter -- The Slew Rate Limiter data cluster
- ResetValue -- The value to reset to.

Outputs:

- out Slew Rate Limiter -- The modified Slew Rate Limiter data cluster

SlewRateLimiter_SetRate



Sets the current rate value.

Inputs:

- in Slew Rate Limiter -- The Slew Rate Limiter data cluster
- Positive Rate -- The new positive rate limit in units/second. This is expected to be ≥ 0 .
- Negative Rate -- The new negative rate limit in units/second. This is expected to be ≤ 0 . Optional. If not wired or the value is > 0 then the Positive Rate * -1 is used.

Outputs:

- out Slew Rate Limiter -- The modified Slew Rate Limiter data cluster

SngJntArmSim

SngJntArmSim_EstimateMOI



Calculates a rough estimate of the moment of inertia of an arm given its length and mass.

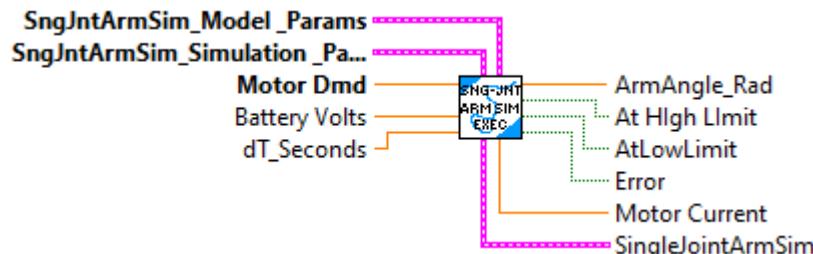
Inputs:

- lengthMeters -- The length of the arm.
- massKg -- The mass of the arm.

Outputs:

- MOI -- The calculated moment of inertia.

SngJntArmSim_Execute



Single call LabVIEW function to simulate a Single Jointed Arm system.

The single jointed arm linear system definition is:

- States
 - angle position (Rad)
 - velocity (Rad/Sec)

- Inputs:

- motor voltage
- Outputs:
 - angle position (rad)

Inputs:

- SngJntArm Model Params -- cluster -- Contains physical configuration for Single Jointed Arm system.
- SngJntArm Sim Simulation Params -- cluster -- Contains siulation configuration.
- Motor Dmd (CO) -- double -- motor demand value (+/- 1.0)
- Battery Voltage -- double -- Current simulated battery voltage (Volts)
- dT Sec -- double -- Update period (Seconds, Default: 0.02)

Outputs:

- ArmAngle Rad -- double -- Current angle of the arm (Radians)
- At High Limit -- boolean -- High limit of travel has been reached
- At Low Limit -- boolean -- Low limit of travel has been reached.
- Error -- boolean -- If TRUE an error occured.
- Motor Current -- double -- Current motor current (Amps)
- outSngJntArmSim -- cluster -- Current internal data cluster.

SngJntArmSim_GetAngleRads



Returns the current arm angle.

Inputs:

- SingleJointArmSim -- updated system data cluster

Outputs:

- ArmAngle-Radians -- The current arm angle (Radians).
-
-

SngJntArmSim_GetCurrentDraw



Returns the arm current draw.

Inputs:

- SingleJointArmSim -- updated system data cluster

Outputs:

- Current_Amps -- The arm current draw. (Amps)
-
-

SngJntArmSim_GetVelocityRadsPerSec



Returns the current arm velocity.

Inputs:

- SingleJointArmSim -- updated system data cluster

Outputs:

- Velocity_RadPerSec -- The current arm velocity (Radians/Sec).
-
-

SngJntArmSim_HasHitLowerLimit



Returns whether the arm has hit the lower limit.

Inputs:

- SingleJointArmSim -- system data cluster

Outputs:

- AtLowLimit -- Whether the arm has hit the lower limit.

SngJntArmSim_HasHitUpperLimit



Returns whether the arm has hit the upper limit.

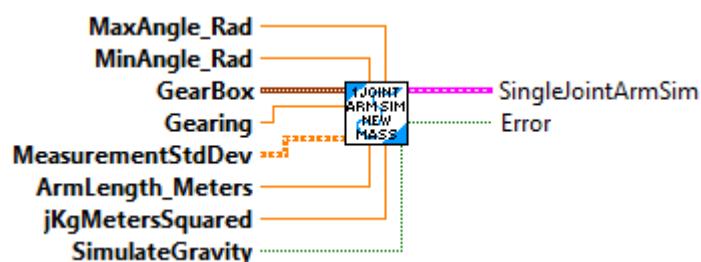
Inputs:

- SingleJointArmSim -- updated system data cluster

Outputs:

- AtHighLimit -- Whether the arm has hit the upper limit.

SngJntArmSim_New



Creates a simulated arm mechanism.

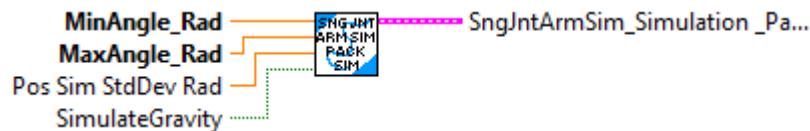
Inputs:

- maxAngleRads -- The maximum angle that the arm is capable of.
- minAngleRads -- The minimum angle that the arm is capable of.
- gearbox -- The type of and number of motors in the arm gearbox.
- gearing -- The gearing of the arm (numbers greater than 1 represent reductions).
- armMassKg -- The mass of the arm.
- measurementStdDevs -- The standard deviations of the measurements.
- armLengthMeters -- The length of the arm.
- jKgMetersSquared -- The moment of inertia of the arm; can be calculated from CAD software.
- simulateGravity -- Whether gravity should be simulated or not.

Outputs:

- SingleJointArmSim -- system data cluster\
- error -- If TRUE, an error occurred.

SngJntArmSim_Pack_Simulation_Params



Pack simulation parameters used by the Single Jointed Arm Simulation Execute routine.

Inputs:

- Min Angle -- double -- Minimum physical arm angle (radians)
- Max Angle -- double -- Maximum physical arm angle (radians)
- Pos Sim Std Dev -- double -- Standard deviation for the arm position (radians Default: 0.09)

- SimulateGravity -- boolean -- Simulate gravity (Default: True)

Outputs:

- SngJntArm Sim Simulation -- cluster -- Packed data for Execute function

SngJntArmSim_Rkf45_Func



Custom function to calculate the newX used for integrating to get updated HatX

Inputs:

- Variant -- extra data used by function

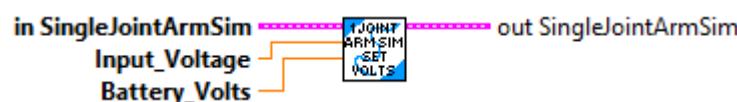
for this function, the extra data is; Matrix A, Matrix B, Arm length, Simulator gravity flag.

- X - X matrix
- U - U matrix

Outputs:

- NewX -- X value as a function of (X, U, extra data)
- error -- If TRUE, an error occurred.

SngJntArmSim_SetInputVoltage



Sets the input voltage for the arm.

Inputs:

- SingleJointArmSim -- system data cluster
- volts -- The input voltage.
- Battery_Volts -- current battery voltage

Outputs:

- SingleJointArmSim -- updated system data cluster

SngJntArmSim_SetState



Sets the system state.

Inputs:

- SngJntArmSim -- Data cluster
- state -- The new state.

Outputs:

- OutSngJntArmSim -- Updated data cluster
- SizeCoerced -- If TRUE, an error occurred. Execution may continue.

SngJntArmSim_Update



Updates the simulation.

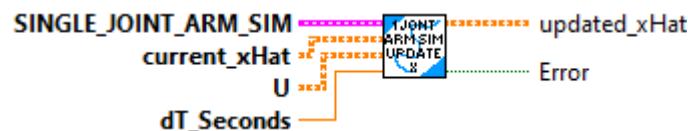
Inputs:

- SngJntArmSim -- Data cluster
- dtSeconds -- The time between updates.

Outputs:

- OutSngJntArmSim -- Updated data cluster
- Error -- If TRUE, an error occurred.

SngJntArmSim_UpdateX



Updates the state of the arm.

Inputs:

- currentXhat -- The current state estimate.
- u -- The system inputs (voltage).
- dtSeconds -- The time difference between controller updates.

Outputs:

- updated_Xhat -- Updated current state estimate
- error -- If TRUE, an error occurred.

SngJntArmSim_WouldHitLowerLimit



Returns whether the arm would hit the lower limit.

Inputs:

- SingleJointArmSim -- updated system data cluster
- currentAngleRads -- The current arm height.

Outputs:

- WouldHitLowLimit -- Whether the arm would hit the lower limit.

SngJntArmSim_WouldHitUpperLimit



Returns whether the arm has hit the upper limit.

Inputs:

- SingleJointArmSim -- updated system data cluster
- currentAngleRads -- The current arm height.

Outputs:

- WouldHitHighLimit -- Whether the arm would hit the upper limit.

Spline

Spline_getPoint



Gets the pose and curvature at some point t on the spline.

Parameters:

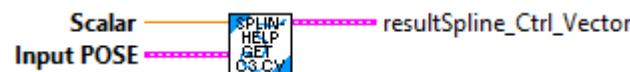
- Input Spline - Spline data structure
- Time - The point in time to evaluate. (between 0 and 1)

Returns:

- PoseWCurve - The pose and curvature at that point.

SplineHelp

SplineHelp_GetCubicCtrlVector



SplineHelp_GetCubicCtrlVectorsFromWayPts



Returns 2 cubic control vectors from a set of exterior waypoints and interior translations.

Parameters:

- startPose - The starting pose.
- interiorWaypoints - The interior waypoints.
- endPose - The ending pose.

Returns:

- SplineControlVectors - Array of cubic control vectors.

SplineHelp_GetCubicCtrlVectorsFromWeightedWayPts



Returns cubic spline control vectors and interior waypoints from a set of exterior weighted waypoints.

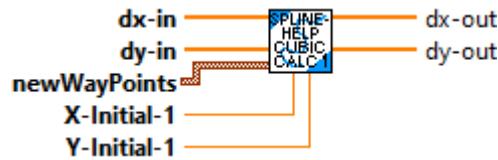
Parameters:

- WeightedWaypoints -- The weighted waypoints.
- UseWeights -- boolean indicating that weights should be used.

Returns:

- SplineControlVectors - Array of cubic control vectors.
 - InteriorWaypoints -- Array of interior waypoints.
-
-

SplineHelp_GetCubicSpline_Calc1



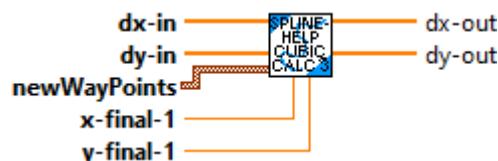
Internal routine called by: SplineHelp_getCubicSplinesFromControlVectors.vi

SplineHelp_GetCubicSpline_Calc2



Internal routine called by: SplineHelp_getCubicSplinesFromControlVectors.vi

SplineHelp_GetCubicSpline_Calc3



Internal routine called by: SplineHelp_getCubicSplinesFromControlVectors.vi

SplineHelp_GetQuinticCtrlVector



SplineHelp_GetQuinticSplinesFromWayPts



Returns quintic splines from a set of waypoints.

Parameters:

- waypoints - The array of waypoints (Translation)

Returns:

- Splines - Array of splines

SplineHelp_GetQuinticSplinesFromWeightedWayPts



Returns quintic splines from a set of weighted waypoints.

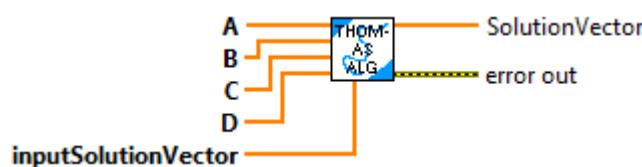
Parameters:

- weighted waypoints - The array of weighted waypoints

Returns:

- Splines - Array of splines

SplineHelp_ThomasAlgorithm



Thomas algorithm for solving tridiagonal systems $Af = d$.

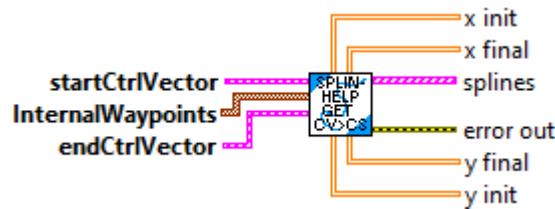
Parameters:

- a - the values of A above the diagonal
- b - the values of A on the diagonal
- c - the values of A below the diagonal
- d - the vector on the rhs
- InputSolutionVector - the unknown (solution) vector

Returns:

- SolutionVector - The modified solution vector
- Error Out - Returned error cluster

SplineHelp_getCubicSplinesFromControlVectors



Returns a set of cubic splines corresponding to the provided control vectors. The user is free to set the direction of the start and end point. The directions for the middle waypoints are determined automatically to ensure continuous curvature throughout the path.

Parameters:

- startCtrlVector - The starting control vector.
- waypoints - The middle waypoints. This can be left blank if you only wish to create a path with two waypoints.
- endCtrlVector - The ending control vector.

Returns:

- A vector of cubic hermite splines that interpolate through the provided waypoints and control vectors.
 - Error Out - Output error cluster
 - X init - For diagnostics only (these may be deleted in the future)
 - X final - For diagnostics only (these may be deleted in the future)
 - Y init - For diagnostics only (these may be deleted in the future)
 - Y final - For diagnostics only (these may be deleted in the future)
-

SplineHelp_getQuinticSplinesFromControlVectors



Returns a set of quintic splines corresponding to the provided control vectors. The user is free to set the direction of all control vectors. Continuous curvature is guaranteed throughout the path.

Parameters:

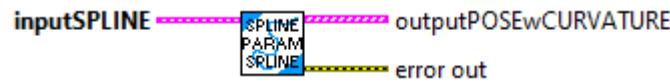
- controlVectors - The control vectors.

Returns:

- Splines - A vector of quintic hermite splines that interpolate through the provided waypoints.
- Error Out - The returned error cluster

SplineParam

SplineParam_Spline



Parameterizes the spline. This method breaks up the spline into various arcs until their dx, dy, and dtheta are within specific tolerances.

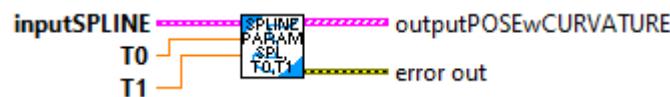
Parameters:

- InputSpline - The spline to parameterize.

Returns:

- outputPoseWCurvature - A list of poses and curvatures that represents various points on the spline.
- Error Out - Output error cluster

SplineParam_Spline_T0_T1



Parameterizes the spline. This method breaks up the spline into various arcs until their dx, dy, and dtheta are within specific tolerances.

Parameters:

- InputSpline - The spline to parameterize.
- T0 - Starting internal spline parameter. It is recommended to use 0.0.
- T1 - Ending internal spline parameter. It is recommended to use 1.0.

Returns:

- OutputPoseWCurvature - A list of poses and curvatures that represents various points on the spline.
- Error Out - Returned error cluster

SplineParam_StackGet



Internal routine used by SplineParam_Spline_T0_T1. This routine gets, but does not remove, the item from the top of the stack.

Parameters:

- InputStack - Stack data structure

Returns:

- T0
 - T1
-

SplineParam_StackPop



Internal routine used by SplineParam_Spline_T0_T1. This routine "pops", the item from the top of the stack and decrements the items on the stack.

Parameters:

- InputStack - Stack data structure

Returns:

- OutputStack - Updated Stack data structure
 - T0
 - T1
-

SplineParam_StackPush



Internal routine used by SplineParam_Spline_T0_T1. This routine "pushes" the item to the top of the stack and increments the number of items on the stack.

Parameters:

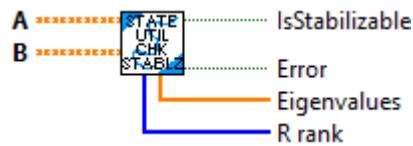
- InputStack - Stack data structure
- T0
- T1

Returns:

- OutputStack - Updated Stack data structure

StateSpaceUtil

StateSpaceUtil_Check_Stabilizable



THIS IS AN INTERNAL ROUTINE. USE IsStabilizable INSTEAD.

Returns true if (A, B) is a stabilizable pair.

(A,B) is stabilizable if and only if the uncontrollable eigenvalues of A, if any, have absolute values less than one, where an eigenvalue is uncontrollable if $\text{rank}(\Lambda - A, B) < n$ where n is number of states.

Inputs:

- A -- System matrix.
- B -- Input matrix.

Outputs:

- IsStabilizable -- If the system is stabilizable.

StateSpaceUtil_ClampInputMaxMagnitude



Clamp the input u to the min and max.

Inputs:

- u -- The input to clamp.
- umin -- The minimum input magnitude.

- u_{\max} -- The maximum input magnitude.

Outputs:

- ClampedU -- The clamped input.
- sizeCoerced -- If TRUE, an error occurred.

StateSpaceUtil_IsDetectable



Returns true if (A, C) is a detectable pair.

(A, C) is detectable if and only if the unobservable eigenvalues of A , if any, have absolute values less than one, where an eigenvalue is unobservable if $\text{rank}(\Lambda - A; C) < n$ where n is the number of states.

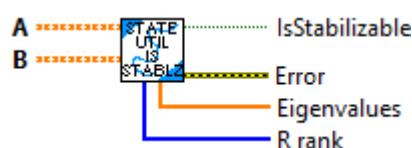
Inputs:

- A -- System matrix.
- C -- Output matrix.

Outputs:

- IsDetectable -- If the system is detectable.

StateSpaceUtil_IsStabilizable



Returns true if (A, B) is a stabilizable pair.

(A, B) is stabilizable if and only if the uncontrollable eigenvalues of A , if any, have absolute values less than one, where an eigenvalue is uncontrollable if $\text{rank}(\Lambda - A, B) < n$ where n is number of states.

Inputs:

- A -- System matrix.
- B -- Input matrix.

Outputs:

- IsStabilizable -- If the system is stabilizable.

StateSpaceUtil_MakeCostMatrix



Creates a cost matrix from the given vector for use with LQR.

The cost matrix is constructed using Bryson's rule. The inverse square of each tolerance is placed on the cost matrix diagonal. If a tolerance is infinity, its cost matrix entry is set to zero.

Inputs:

- Elements -- representing the number of system states or inputs
- tolerances -- For a Q matrix, its elements are the maximum allowed excursions of the states from the reference. For an R matrix, its elements are the maximum allowed excursions of the control inputs from no actuation.

Outputs:

- CostsMatrix -- State excursion or control effort cost matrix.
- sizeCoerced -- If TRUE, an error occurred.

StateSpaceUtil_MakeCovarianceMatrix



Creates a covariance matrix from the given vector for use with Kalman filters.

Each element is squared and placed on the covariance matrix diagonal.

Inputs:

- states -- the number of states of the system.
- stdDevs -- For a Q matrix, its elements are the standard deviations of each state from how the model behaves. For an R matrix, its elements are the standard deviations for each output measurement.

Outputs:

- CovarianceMatrix -- Process noise or measurement noise covariance matrix.
- sizeCoerced -- If TRUE, an error occurred.

StateSpaceUtil_MakeWhiteNoiseVector



Creates a vector of normally distributed white noise with the given noise intensities for each element.

Inputs:

- stdDevs -- A matrix whose elements are the standard deviations of each element of the noise vector.

Outputs:

- WhiteNoise -- White noise vector.
- sizeCoerced -- If TRUE, an error occurred.

StateSpaceUtil_NormalizeInputVector



Normalize all inputs if any exceeds the maximum magnitude. Useful for systems such as differential drivetrains.

Inputs:

- J -- The number of inputs.
- u -- The input vector.
- maxMagnitude -- The maximum magnitude any input can have.

Outputs:

- NormalizedU -- The normalizedInput
- sizeCoerced -- If TRUE, an error occurred.

StateSpaceUtil_PoseTo3dVector



Convert a Pose2d to a vector of [x, y, theta], where theta is in radians.

Inputs:

- pose -- A pose to convert to a vector.

Outputs:

- PoseVector -- The given pose in vector form, with the third element, theta, in radians.

StateSpaceUtil_PoseTo4dVector



Convert a Pose2d}to a vector of [x, y, cos(theta), sin(theta)], where theta is in radians.

Inputs:

- pose -- A pose to convert to a vector.

Outputs:

- PoseVector -- The given pose in as a 4x1 vector of x, y, cos(theta), and sin(theta).
-

StateSpaceUtil_PoseToVector



Convert a Pose2d to a vector of [x, y, theta], where theta is in radians.

Inputs:

- pose -- A pose to convert to a vector.

Outputs:

- PoseVector -- The given pose in vector form, with the third element, theta, in radians.

SwerveDriveKinematicsConstraint

SwerveDriveKinematicsConstraint_New



Constructs a swerve drive dynamics constraint.

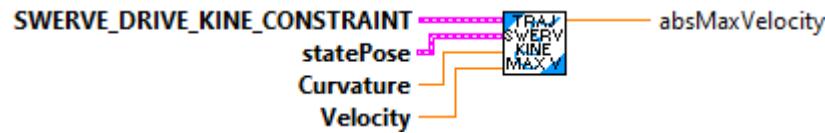
Parameters:

- maxVelocity - Maximum allowed velocity.
- SwerveDriveKinematics - Data structure

Returns

- SwerveDriveKinematicsConstraint - Constraint data structure

SwerveDriveKinematicsConstraint_getMaxVelocity



Return the maximum allowed velocity given the provided conditions.

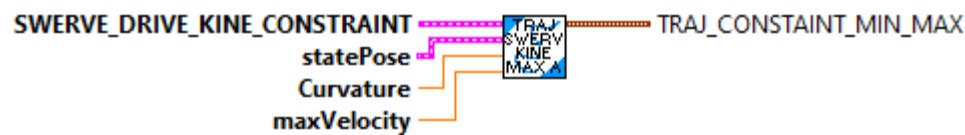
Parameters:

- SwerveDriveKinematicsConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

SwerveDriveKinematicsConstraint_getMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

It appears that this routine doesn't do anything. It returns default values.

Parameters:

- SwerveDriveKinematicsConstraint - Constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- TrajConstraint_Min_Max - Data structure with Min / Max acceleration.

SwerveDrivePoseEst

SwerveDrivePoseEst_AddVisionMeasurement



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Add a vision measurement to the Unscented Kalman Filter. This will correct the odometry pose estimate while still accounting for measurement noise.

This method can be called as infrequently as you want, as long as you are calling `SwerveDrivePoseEstimator_update` every loop.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

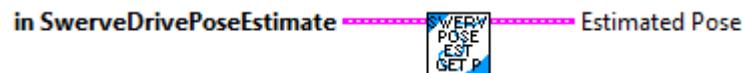
Inputs:

- `SwervePoseEst` -- Data cluster containing `SwervePoseEst` data
- `visionRobotPoseMeters` -- The pose of the robot as measured by the vision camera.
- `timestampSeconds` -- The timestamp of the vision measurement in seconds. Note that if you don't use your own time source by calling `SwerveDrivePoseEstimator_updateWithTime` then you must use a timestamp with an epoch since FPGA startup (i.e. the epoch of this timestamp is the same epoch as `Timer.getFPGATimestamp`.) This means that you should use `Timer.getFPGATimestamp` as your time source or sync the epochs.

Outputs:

- outSwevePoseEst -- Updated data cluster
 - Error -- If TRUE, an error occurred.
-

SwerveDrivePoseEst_GetEstimatedPosition



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Gets the pose of the robot at the current time as estimated by the Unscented Kalman Filter.

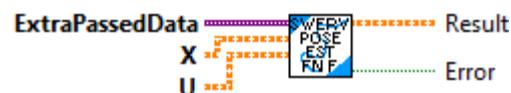
Inputs:

- SwervePoseEst -- Data cluster containing SwervePoseEst data

Outputs:

- EstimatedPose - The estimated robot pose in meters.
-

SwerveDrivePoseEst_Kalman_F_Callback



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Swerve Drive Pose Estimator, Kalman Filter F function.

This function returns the U matrix.

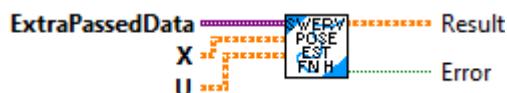
Input:

- ExtraPassedData -- Variant containing extra data for the callback For this function the extra data should be empty..
- X Matrix -- X matrix
- U Matrix -- U matrix

Output:

- Result -- Matrix resulting from calculation
 - Error -- If TRUE an error occurred
-

SwerveDrivePoseEst_Kalman_H_Callback



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Swerve Drive Pose Estimator, Kalman Filter H function.

This function returns row 2 of the X matrix.

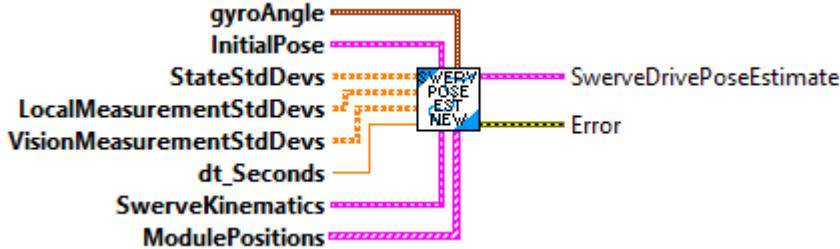
Input:

- ExtraPassedData -- Variant containing extra data for the callback For this function the extra data should be empty..
- X Matrix -- X matrix
- U Matrix -- U matrix

Output:

- Result -- Matrix resulting from calculation
 - Error -- If TRUE an error occurred
-

SwerveDrivePoseEst_New



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Constructs a SwerveDrivePoseEstimator data cluster.

The functions that use this data cluster wrap an UnscentedKalmanFilter Unscented Kalman Filter to fuse latency-compensated vision measurements with swerve drive encoder velocity measurements. It will correct for noisy measurements and encoder drift. It is intended to be an easy but more accurate drop-in for SwerveDriveOdometry.

The generic arguments to this data cluster define the size of the state, input and output vectors used in the underlying UnscentedKalmanFilter Unscented Kalman Filter. Num States must be equal to the module count + 3. Num Inputs must be equal to the module count + 3. Num Outputs must be equal to the module count + 1.

SwerveDrivePoseEstimator_update should be called every robot loop. If your loops are faster or slower than the default of 20 ms, then you should change the nominal delta time using the secondary constructor: SwerveDrivePoseEstimator_SwerveDrivePoseEstimator(Nat, Nat, Nat, Rotation2d, Pose2d, SwerveModulePosition[], SwerveDriveKinematics, Matrix, Matrix, Matrix, double).

SwerveDrivePoseEstimator_addVisionMeasurement can be called as infrequently as you want; if you never call it, then this class will behave mostly like regular encoder odometry.

The state-space system used internally has the following states (x), inputs (u), and outputs (y):

$$x = [x, y, \theta, s_0, \dots, s_n]t$$

in the field coordinate system containing x position, y position, and heading, followed by the distance travelled by each wheel.

$u = [v_x, v_y, \omega, v_0, \dots, v_n]t$

containing x velocity, y velocity, and angular rate in the field coordinate system, followed by the velocity measured at each wheel.

$y = [x, y, \theta]t$

from vision containing x position, y position, and heading; or

$y = [\theta, s_0, \dots, s_n]t$

containing gyro heading, followed by the distance travelled by each wheel.

Inputs:

- gyroAngle -- The current gyro angle.
- initialPoseMeters -- The starting pose estimate.
- modulePositions -- The current distance measurements and rotations of the swerve modules.
- kinematics -- A correctly-configured kinematics object for your drivetrain.
- stateStdDevs -- Standard deviations of model states. Increase these numbers to trust your model's state estimates less. This matrix is in the form $[x, y, \theta, s_0, \dots, s_n]T$, with units in meters and radians, then meters.
- localMeasurementStdDevs -- Standard deviations of the encoder and gyro measurements. Increase these numbers to trust sensor readings from encoders and gyros less. This matrix is in the form $[\theta, s_0, \dots, s_n]$, with units in radians followed by meters.
- visionMeasurementStdDevs -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form $[x, y, \theta]?$, with units in meters and radians.

Outputs:

- outSwevePoseEst -- Updated data cluster
 - Error -- If TRUE, an error occurred.
-

SwerveDrivePoseEst_ResetPosition



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Resets the robot's position on the field.

(NOTE -- For LabVIEW version, this may not be needed.) You NEED to reset your encoders (to zero) when calling this method.

The gyroscope angle does not need to be reset in the user's robot code. The library automatically takes care of offsetting the gyro angle.

Inputs:

- SwervePoseEst -- Data cluster containing SwervePoseEst data
- poseMeters -- The position on the field that your robot is at.
- gyroAngle -- The angle reported by the gyroscope.
- modulePositions -- The current distance measurements and rotations of the swerve modules.

Outputs:

- outSwevePoseEst -- Updated data cluster
-

SwerveDrivePoseEst_SetVisionMeasurementStdDevs



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Sets the pose estimator's trust of global measurements. This might be used to change trust in vision measurements after the autonomous period, or to change trust as distance to a vision target increases.

Inputs:

- SwervePoseEst -- Data cluster containing SwervePoseEst data
- visionMeasurementStdDevs -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form

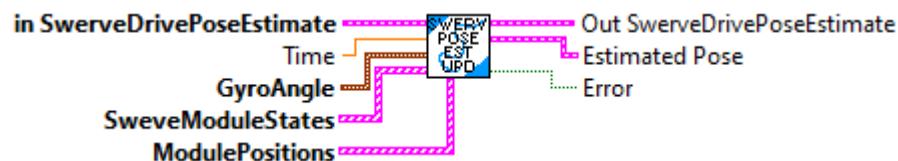
$[x, y, \theta]^T$,

with units in meters and radians.

Outputs:

- outSwevePoseEst -- Updated data cluster
- SizeCoereced -- If TRUE, an unexpected error occurred. Execution may continue.

SwerveDrivePoseEst_Update



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Updates the the Unscented Kalman Filter using only wheel encoder information. This should be called every loop, and the correct loop period must be passed into the constructor of this class.

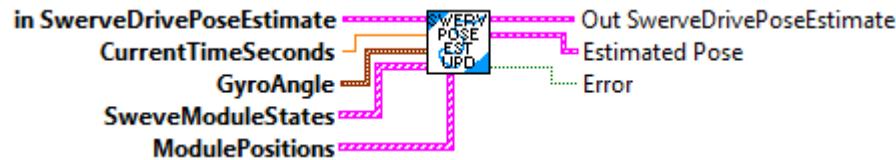
Inputs:

- In Swerve Drive Pose Est -- data cluster
- Time -- (Optional) System continuous time counter.
- gyroAngle -- The current gyro angle.
- moduleStates -- The current velocities and rotations of the swerve modules.
- modulePositions -- The current distance measurements and rotations of the swerve modules.

Outputs:

- outSwervePoseEst -- Updated data cluster
- EstimatedPose -- The estimated pose of the robot in meters.
- Error -- If TRUE, an error occurred.

SwerveDrivePoseEst_UpdateWithTime



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Updates the the Unscented Kalman Filter using only wheel encoder information. This should be called every loop, and the correct loop period must be passed into the constructor of this class.

Inputs:

- SwervePoseEst -- Data cluster containing SwervePoseEst data
- currentTimeSeconds Time at which this method was called, in seconds.
- gyroAngle The current gyroscope angle.
- moduleStates The current velocities and rotations of the swerve modules.
- modulePositions -- The current distance measurements and rotations of the swerve modules.

Outputs:

- outSwevePoseEst -- Updated data cluster
 - EstimatedPose -- The estimated pose of the robot in meters.
 - Error -- If TRUE, an error occurred.
-

SwerveDrivePoseEst_VisionCorrect_Callback



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Swerve Drive Pose Estimator function used to update Kalman Filter.

This function calls the UnscentedKalmanFilter Correct function with U, Y, VisionContR matrices.

The parameters of this function are fixed since it's reference is passed to other functions. This is why the VisionContR matrix needs to be passed as extra data.

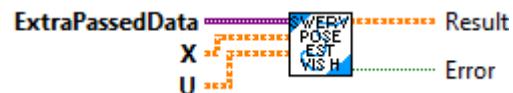
Inputs:

- Unscented Kalman Filter -- Input data cluster
- U -- U matrix
- Y -- Y matrix
- ExtraData -- Variant containing extra data. For this call the extra data must contain the VisionContR Matrix.

Outputs:

- Out Unscented Kalman Filter -- Updated data cluster
-

SwerveDrivePoseEst_VisionCorrect_Kalman_H_Callback



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

Swerve Drive Pose Estimator, Kalman Filter H function for vision correction. This function passes the X matrix to the resulting matrix. No calculations are done.

Input:

- ExtraPassedData -- Variant containing extra data for the callback. For this function the extra data should be empty..
- X Matrix -- X matrix
- U Matrix -- U matrix

Output:

- Result -- Matrix resulting from calculation
- Error -- If TRUE an error occurred

SwerveDrivePoseEst2

SwerveDrivePoseEst2_AddVisionMeasurement



Add a vision measurement to the Unscented Kalman Filter. This will correct the odometry pose estimate while still accounting for measurement noise.

This method can be called as infrequently as you want, as long as you are calling `SwerveDrivePoseEstimator_update` every loop.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

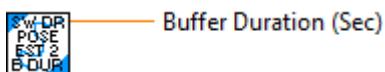
Inputs:

- `InSwerveDrivePoseEstimate` -- Data cluster for this system
- `visionRobotPoseMeters` -- The pose of the robot as measured by the vision camera.
- `timestampSeconds` -- The timestamp of the vision measurement in seconds. Note that if you don't use your own time source by calling `SwerveDrivePoseEstimator_updateWithTime` then you must use a timestamp with an epoch since FPGA startup (i.e. the epoch of this timestamp is the same epoch as `Util_GetFPGATime`.) This means that you should use `Util_GetFPGATime` as your time source in this case.

Outputs:

- `OutSwerveDrivePoseEstimate2` -- Data cluster for this system
- `Error` -- Returns TRUE if an error occurred.

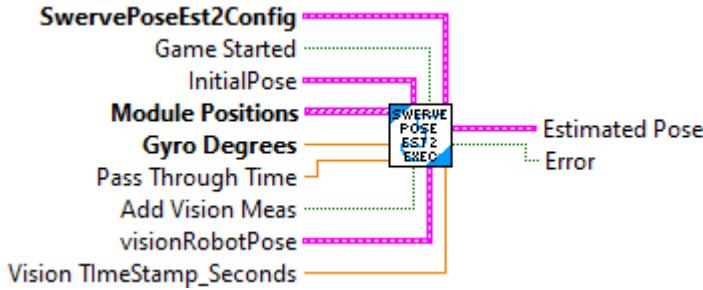
SwerveDrivePoseEst2_BufferDuration



Gets the maximum duration for the Time Interp Variant buffer.

This is an internal routine. It should NOT be called by the end user.

SwerveDrivePoseEst2_Execute



This single call LabVIEW function wraps Swerve Drive Odometry to fuse latency-compensated vision measurements with swerve drive module measurements. It is intended to be a drop-in replacement for SwerveDrvOdom2; in fact, if you never call SwerveDrvPoseEst2_AddVisionMeasurement and only call SwerveDrvPoseEst2_Update then this will behave exactly the same as SwerveDrvOdom2.

This function should be called every robot loop, or perhaps put into a separate loop.

When the "Game Started" input is false, the position is reset using the initial pose, wheel distances, and gyro position. Use this to wait processing changes in measurements until the game starts.

When a new vision measurement is available it can be passed into this function to merge it with the other odometry data. This is done by setting the "Add Vision Meas" input to TRUE. To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

The default standard deviations of the model states are 0.1 meters for x, 0.1 meters for y, and 0.1 radians for heading. The default standard deviations of the vision measurements are 0.9 meters for x, 0.9 meters for y, and 0.9 radians for heading.

Inputs:

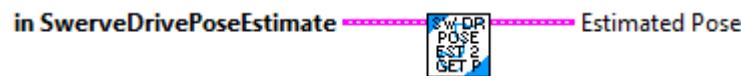
- **SwervePoseEst2Config** -- cluster -- Packed configuration data used to define how this function operates. It includes information used to define kinematics and position standard deviations.
- **Game Started** -- boolean -- When FALSE, resets the initial position. (Default: TRUE)
- **InitialPose** -- Pose2d -- Initial position. (meters, meters, rotation2d)

- ModulePositions -- SwerveModulePositions array -- Array of distances measured by the drive wheel encoders. (Meters)
- gyroAngle -- Rotation2d -- The current gyro angle. (Degrees)
- Pass Through Time -- double -- Current FPGA time. If input is unwired, FPGA time will be read internally.
- AddVisionMeas -- boolean -- When TRUE the provided vision measurement is merged with the odometry data.
- VisionRobotPose -- Pose2d -- The pose from vision measurements.
- Vision Time Stamp -- double -- The FPGA time stamp of the vision measurement. This helps to account for latency of the vision measurement.

Outputs:

- EstimatedPose -- Pose2d -- Current estimated pose (meters, rotation2d)
- Error -- boolean -- If TRUE, an error occurred.

SwerveDrivePoseEst2_GetEstimatedPosition



Gets the pose of the robot at the current time as estimated by the Unscented Kalman Filter.

Inputs:

- SwerveDrivePoseEst2 - System data cluster

Outputs:

- EstimatedPose - The estimated robot pose in meters.

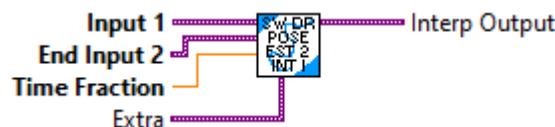
SwerveDrivePoseEst2_InterpRecord_ExtractFromVar



This extracts the specific data cluster of Swerve Drive Pose Estimator2 Interpolation Record from a Variant.

This is an internal routine. It should NOT be called by the end user.

SwerveDrivePoseEst2_InterpRecord_Interp



Return the interpolated record. This object is assumed to be the starting position, or lower bound.

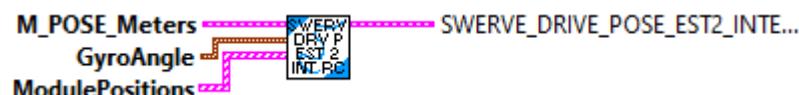
Inputs:

- Input 1 -- The lower bound data cluster
- end Input 2 -- The upper bound, or end, data cluster
- time frac -- How far between the lower and upper bound we are. This should be bounded in [0, 1].
- extra -- variant -- extra data needed for the calculation.

Returns:

- The interpolated cluster value.
-
-

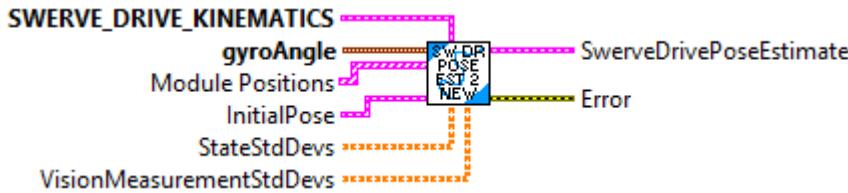
SwerveDrivePoseEst2_InterpRecord_New



Create a new SwerveDrivePoseEstimator2 Interpolation record.

This is an internal routine. It should NOT be used by the end user.

SwerveDrivePoseEst2_New



This class wraps Swerve Drive Odometry to fuse latency-compensated vision measurements with differential drive encoder measurements. It is intended to be a drop-in replacement for SwerveDrvOdometry; in fact, if you never call SwerveDrvPoseEst2_AddVisionMeasurement and only call SwerveDrvPoseEst2_Update then this will behave exactly the same as SwerveDrvOdometry.

SwerveDrvPoseEst2_Update should be called every robot loop.

SwerveDrvPoseEst2_AddVisionMeasurement can be called as infrequently as you want. If you never call it then this set of VI will behave exactly like regular encoder odometry.

Constructs a SwerveDrivePoseEstimator2.

The default standard deviations of the model states are 0.1 meters for x, 0.1 meters for y, and 0.1 radians for heading. The default standard deviations of the vision measurements are 0.9 meters for x, 0.9 meters for y, and 0.9 radians for heading.

Inputs:

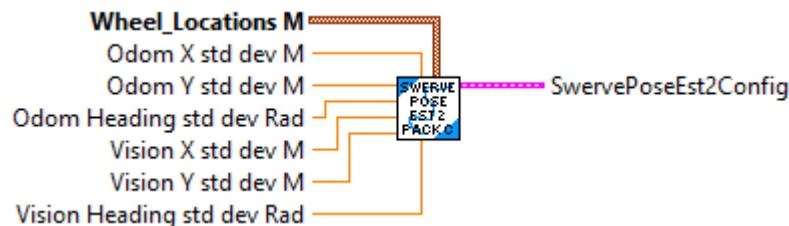
- kinematics -- SwerveDriveKinematics -- A correctly-configured kinematics data cluster for your drivetrain.
- gyroAngle -- Rotation2d -- The current gyro angle.
- ModulePositions -- ModulePositions -- The current module positions.
- initialPoseMeters -- Pose2d -- The starting pose estimate.
- stateStdDevs -- <3,1> matrix -- Standard deviations of the pose estimate (x position in meters, y position in meters, and heading in radians). Increase these numbers to trust your state estimate less.

- visionMeasurementStdDevs -- <3,1> matrix -- Standard deviations of the vision pose measurement (x position in meters, y position in meters, and heading in radians). Increase these numbers to trust the vision pose measurement less.

Outputs:

- SwerveDrivePoseEst2 -- SwerveDrivePoseEst2 -- Created data cluster.

SwerveDrivePoseEst2_Pack_Config



Pack individual values into the cluster required by the SwerveDrivePoseEst2_Execute function.

Inputs

- Wheel Locations M -- Translation2d array -- Locations of module wheels in relation to center of robot. (meters)
- Odom X Std Dev M -- double -- Odometry X position standard deviation (Default: 0.02) (meters)
- Odom Y Std Dev M -- double -- Odometry Y position standard deviation (Default: 0.02) (meters)
- Odom Heading Std Dev M -- double -- Odometry Heading (gyro) position standard deviation (Default: 0.01) (radians)
- Vision X Std Dev M -- double -- Vision X position standard deviation (Default: 0.1) (meters)
- Vision Y Std Dev M -- double -- Vision Y position standard deviation (Default: 0.1) (meters)
- Vision Heading Std Dev M -- double -- Vision Heading (gyro) position standard deviation (Default: 0.05) (radians)

Outputs:

- SwervePoseEst2Config -- cluster -- Packed data values.

SwerveDrivePoseEst2_ResetPosition



Resets the robot's position on the field.

The gyroscope angle does not need to be reset here on the user's robot code. The library automatically takes care of offsetting the gyro angle.

Inputs:

- inSwerveDrvPoseEst2 -- SwerveDrvPoseEst2 -- Data cluster
- gyroAngle -- Rotation2d -- The angle reported by the gyroscope.
- ModulePositions -- ModulePositions -- The current module positions.
- poseMeters -- Pose2d -- The position on the field that your robot is at.

Outputs:

- outSwerveDrvPoseEst2 -- SwerveDrvPoseEst2 -- Updated data cluster

SwerveDrivePoseEst2_SetVisionMeasurementStdDevs



Sets the pose estimator's trust of global measurements. This might be used to change trust in vision measurements after the autonomous period, or to change trust as distance to a vision target increases.

Inputs:

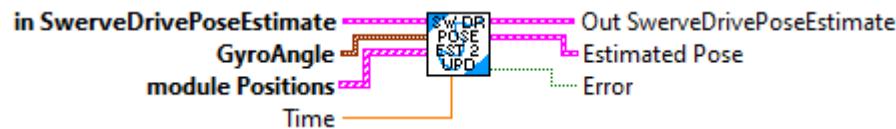
- inSwerveDrvPoseEst2 -- SwerveDrvPoseEst2 -- Data cluster

- VisionMeasurementStdDevs -- <3,1> Matrix -- Standard deviations of the vision measurements. Increase these numbers to trust global measurements from vision less. This matrix is in the form [x, y, theta]Time, with units in meters and radians.

Outputs:

- outSwerveDrvPoseEst2 -- SwerveDrvPoseEst2 -- Updated data cluster
 - sizeCooerced -- boolean -- If TRUE, then the size of the vision measurement standard deviations was not 3,1. The size was modified to allow this routine to complete.
-
-

SwerveDrivePoseEst2_Update



Updates the the robot odometry using only wheel encoder information. Note that this should be called every loop.

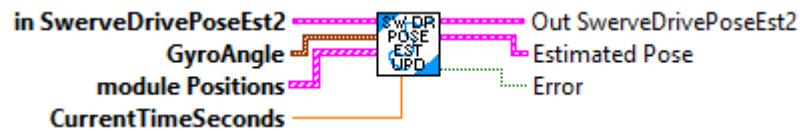
Inputs:

- inSwerveDrivePoseEst -- system data cluster
- gyroAngle -- The current gyro angle. (radians)
- ModulePositions -- Array Module Position -- The current module positions (distance and angle)
- currentTime -- Time at which this method was called, in seconds.

Outputs:

- outSwerveDrivePoseEst -- system data cluster
 - EstimatedPose -- The estimated pose of the robot in meters.
-
-

SwerveDrivePoseEst2_UpdateWithTime



Updates the the robot odometry using only wheel encoder information. Note that this should be called every loop.

Inputs:

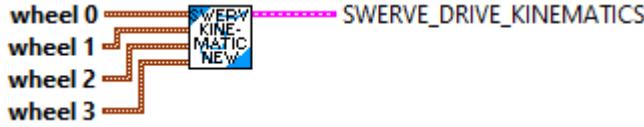
- inSwerveDrivePoseEst -- system data cluster
- gyroAngle -- The current gyro angle. (radians)
- ModulePositions -- Array Module Position -- The current module positions (distance and angle)
- currentTime -- Time at which this method was called, in seconds.

Outputs:

- outSwerveDrivePoseEst -- system data cluster
- EstimatedPose -- The estimated pose of the robot in meters.

SwerveKinematics

SwerveKinematics_New4



Constructs a swerve drive kinematics object. This takes in a variable number of wheel locations as Translation2ds. The order in which you pass in the wheel locations is the same order that you will receive the module states when performing inverse kinematics. It is also expected that you pass in the module states in the same order when calling the forward kinematics methods. This subVI is customized for 4 wheel modules.

This is a helper class that converts a chassis velocity (dx , dy , and $d\theta$ components) into individual module states (speed and angle).

The inverse kinematics (converting from a desired chassis velocity to individual module states) uses the relative locations of the modules with respect to the center of rotation. The center of rotation for inverse kinematics is also variable. This means that you can set your center of rotation in a corner of the robot to perform special evasion maneuvers.

Forward kinematics (converting an array of module states into the overall chassis motion) is performed the exact opposite of what inverse kinematics does. Since this is an overdetermined system (more equations than variables), we use a least-squares approximation.

The inverse kinematics: $[moduleStates] = [moduleLocations][chassisSpeeds]$ We take the Moore-Penrose pseudoinverse of $[moduleLocations]$ and then multiply by $[moduleStates]$ to get our chassis speeds.

Forward kinematics is also used for odometry -- determining the position of the robot on the field using encoders and a gyro.

Parameters:

- wheel 0 - Translation data structures indicating the location of a wheel relative to the physical center of the robot.
- wheel 1 - Translation data structures indicating the location of a wheel relative to the physical center of the robot. (Meters)

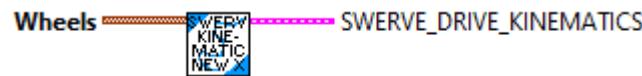
- wheel 2 - Translation data structures indicating the location of a wheel relative to the physical center of the robot. (Meters)

- wheel 3 - Translation data structures indicating the location of a wheel relative to the physical center of the robot. (Meters)

Returns:

- SwerveDriveKinematics - The initialized data structure

SwerveKinematics_NewX



Constructs a swerve drive kinematics object. This takes in a variable number of wheel locations as Translation2ds. The order in which you pass in the wheel locations is the same order that you will receive the module states when performing inverse kinematics. It is also expected that you pass in the module states in the same order when calling the forward kinematics methods.

This is a helper class that converts a chassis velocity (dx , dy , and $d\theta$ components) into individual module states (speed and angle).

The inverse kinematics (converting from a desired chassis velocity to individual module states) uses the relative locations of the modules with respect to the center of rotation. The center of rotation for inverse kinematics is also variable. This means that you can set your center of rotation in a corner of the robot to perform special evasion maneuvers.

Forward kinematics (converting an array of module states into the overall chassis motion) performs the exact opposite of what inverse kinematics does. Since this is an overdetermined system (more equations than variables), we use a least-squares approximation.

The inverse kinematics: $[moduleStates] = [moduleLocations][chassisSpeeds]$ We take the Moore-Penrose pseudoinverse of $[moduleLocations]$ and then multiply by $[moduleStates]$ to get our chassis speeds.

Forward kinematics is also used for odometry -- determining the position of the robot on the field using encoders and a gyro.

Parameters:

- wheels - Array of Translation data structures indicating the locations of the wheels relative to the physical center of the robot. (Meters)

Returns:

- SwerveDriveKinematics - The initialized data structure

SwerveKinematics_NormalizeChassisSpeeds



Normalizes the wheel speeds using some max attainable speed. Sometimes, after inverse kinematics, the requested speed from a/several modules may be above the max attainable speed for the driving motor on that module. To fix this issue, one can "normalize" all the wheel speeds to make sure that all requested module speeds are below the absolute threshold, while maintaining the ratio of speeds between modules.

Parameters:

- moduleStates - Array of module states.
- attainableMaxSpeed - The absolute max speed that a module can reach. (Meters/Sec)

Returns:

- out Wheel States -- Array of normalized Wheel States

SwerveKinematics_NormalizeWheelSpeedsX



Normalizes the wheel speeds using some max attainable speed. Sometimes, after inverse kinematics, the requested speed from a/several modules may be above the max attainable speed for the driving motor on that module. To fix this issue, one can "normalize" all the wheel speeds to make sure that all requested module speeds are below the absolute threshold, while maintaining the ratio of speeds between modules.

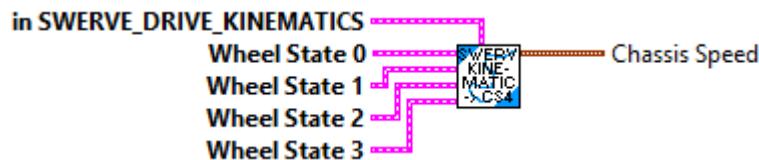
Parameters:

- moduleStates - Array of module states.
- attainableMaxSpeed - The absolute max speed that a module can reach. (Meters/Sec)

Returns:

- out Wheel States -- Array of normalized Wheel States

SwerveKinematics_ToChassisSpeeds4



Performs forward kinematics to return the resulting chassis state from the given module states. This method is often used for odometry -- determining the robot's position on the field using data from the real-world speed and angle of each module on the robot. This subVI is customized for 4 individual wheel states.

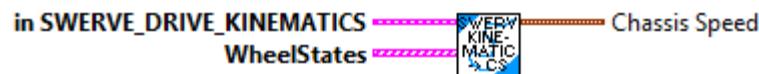
Parameters:

- in Swerve Drive Kinematics - Swerve drive kinematics data structure
- wheelState 0 - The state of a module (as a SwerveModuleState type) as measured from respective encoders and gyros. The order of the swerve module states should be same as passed into the constructor of this class.
- wheelState 1 - Another wheel state
- wheelState 2 - Another wheel state
- wheelState 3 - Another wheel state

Returns:

- The resulting chassis speed.

SwerveKinematics_ToChassisSpeedsX



Performs forward kinematics to return the resulting chassis state from the given module states. This method is often used for odometry -- determining the robot's position on the field using data from the real-world speed and angle of each module on the robot.

Parameters:

- in Swerve Drive Kinematics - Swerve drive kinematics data structure
- wheelStates - The state of the modules (as a SwerveModuleState type) as measured from respective encoders and gyros. The order of the swerve module states should be same as passed into the constructor of this class.

Returns:

- The resulting chassis speed.

SwerveKinematics_ToSwerveModuleStates



Performs inverse kinematics to return the module states from a desired chassis velocity. This method is often used to convert joystick values into module speeds and angles.

This function also supports variable centers of rotation. During normal operations, the center of rotation is usually the same as the physical center of the robot; therefore, the argument is defaulted to that use case. However, if you wish to change the center of rotation for evasive maneuvers, vision alignment, or for any other use case, you can do so.

In the case that the desired chassis speeds are zero (i.e. the robot will be stationary), the previously calculated module angle will be maintained.

Parameters:

- chassisSpeeds -- The desired chassis speed.
- centerOfRotation -- The center of rotation. For example, if you set the center of rotation at one corner of the robot and provide a chassis speed that only has a dtheta component, the robot will rotate around that corner. (Meters)

Returns:

- ModuleStates - An array of data structures containing the module states. Use caution because these module states are not normalized. Sometimes, a user input may cause one of the module speeds to go above the attainable max velocity. Use the {@link #normalizeWheelSpeeds(SwerveModuleState[], double) normalizeWheelSpeeds} function to rectify this issue.

SwerveKinematics_ToSwerveModuleStatesZeroCenter



Performs inverse kinematics to return the module states from a desired chassis velocity. This method is often used to convert joystick values into module speeds and angles.

This uses the robot center as the center of rotation. During normal operations, the center of rotation is usually the same as the physical center of the robot; therefore, the argument is defaulted to that use case. However, if you wish to change the center of rotation for evasive manuevers, vision alignment, or for any other use case, you can do so.

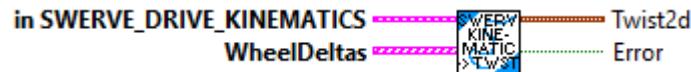
Parameters:

- chassisSpeeds -- The desired chassis speed.

Returns:

- ModuleStates - An array of data structures containing the module states. Use caution because these module states are not normalized. Sometimes, a user input may cause one of the module speeds to go above the attainable max velocity. Use the {@link #normalizeWheelSpeeds(SwerveModuleState[], double) normalizeWheelSpeeds} function to rectify this issue.

SwerveKinematics_ToTwist2dX



Performs forward kinematics to return the resulting chassis state from the given module states. This method is often used for odometry -- determining the robot's position on the field using data from the real-world speed and angle of each module on the robot.

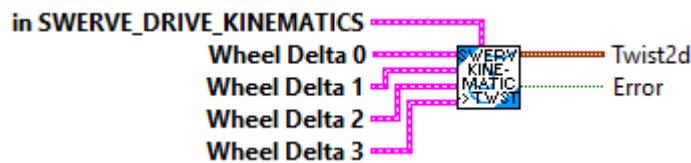
Parameters:

- in Swerve Drive Kinematics - Swerve drive kinematics data structure
- wheelDeltas - An array of the latest change in position of the modules (as a SwerveModulePosition type) as measured from respective encoders and gyros. The order of the swerve module states should be same as passed into the constructor of this class.

Returns:

- The resulting twist2d.

SwerveKinematics_ToTwst2d4



Performs forward kinematics to return the resulting chassis state from the given module states. This method is often used for odometry -- determining the robot's position on the field using data from the real-world speed and angle of each module on the robot.

Parameters:

- in Swerve Drive Kinematics - Swerve drive kinematics data structure
- wheelDelta 0 - The latest change in position of the modules (as a SwerveModulePosition type) as measured from respective encoders and gyros. The order of the swerve module states should be same as passed into the constructor of this class.
- wheelDelta 1 - Another wheel position
- wheelDelta 2 - Another wheel position

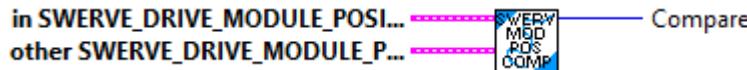
- wheelDelta 3 - Another wheel position

Returns:

- The resulting twist2d.

SwerveModulePosition

SwerveModulePosition_CompareTo



Compares two swerve module positions. One swerve module is "greater" than the other if its position is higher than the other.

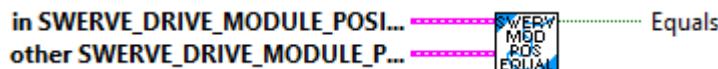
Parameters:

- this Module Position
- other Module Position - The other swerve module.

Returns:

- compare - comparison result where
 - 1 if this is greater,
 - 0 if both are equal,
 - 1 if other is greater.

SwerveModulePosition_Equals



Compares two swerve module positions. Returns TRUE if both are equal

Parameters:

- this Module Position
- other Module Position - The other swerve module.

Returns:

- equal -- boolean - Returns TRUE if both are equal.

SwerveModulePosition_Get



Get the individual position (distance) (meters) and angle (radians) components of the Module Position data cluster.

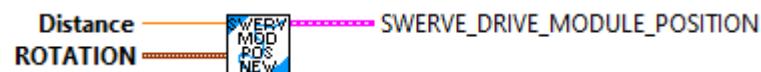
Parameters:

- Module Position -- data cluster

Returns:

- Distance -- Module distance (Meters)
- Angle -- Module angle (Radians)

SwerveModulePosition_New



Constructs a SwerveModulePosition.

Parameters:

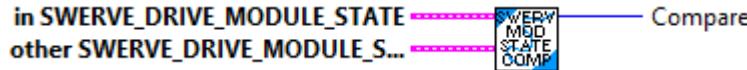
- distance - The distance of the wheel of the module. (Meters)
- rotation - The angle of the module.

Returns:

- SwerveDriveModulePosition - Data structure

SwerveModuleState

SwerveModuleState_CompareTo



Compares two swerve module states. One swerve module is "greater" than the other if its speed is higher than the other.

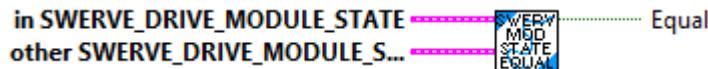
Parameters:

- this Module State
- other Module State - The other swerve module.

Returns:

- compare - comparison result where
 - 1 if this is greater,
 - 0 if both are equal,
 - 1 if other is greater.

SwerveModuleState_Equal



Compares two swerve module states. Returns TRUE if they are equal.

Parameters:

- this Module State
- other Module State - The other swerve module.

Returns:

- equal -- boolean -- Returns TRUE if both module states are equal.

SwerveModuleState_Get



Get the individual speed (meters/sec) and angle (radians) components of the Module State data cluster.

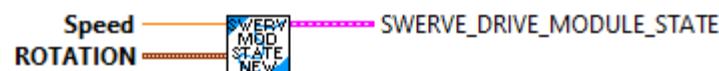
Parameters:

- Module State

Returns:

- Speed -- Module speed (Meters/Sec)
- Angle -- Module angle (Radians)

SwerveModuleState_New



Constructs a SwerveModuleState.

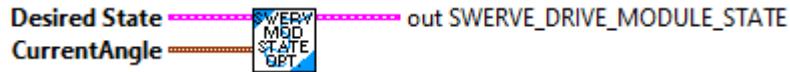
Parameters:

- speed - The speed of the wheel of the module. (Meters/Sec)
- rotation -The angle of the module.

Returns:

- SwerveDriveModuleState - Data structure

SwerveModuleState_Optimize



Minimize the change in heading the desired swerve module state would require by potentially reversing the direction the wheel spins. If this is used with the PIDController class's continuous input functionality, the furthest a wheel will ever rotate is 90 degrees.

Parameters:

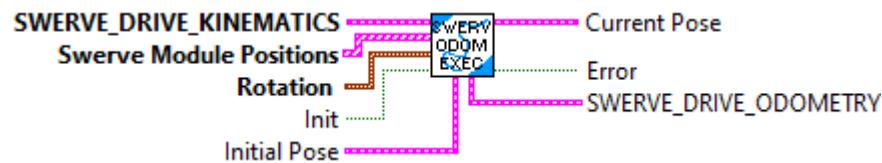
- desired Module State
- current module angle (rotation)

Returns:

- optimized Module State

SwerveOdometry

SwerveOdometry_Execute



SIngle call LabVIEW function to execute Swerve Drive Odometry to calculate a relative or absolute POSE.

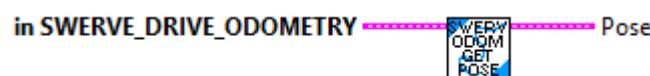
Inputs:

- Swerve_Drive_Kinematics -- cluster -- swerve drive kinematics
- Swerve Module Positions -- array of cluster -- current module positions.
- Gyro Value -- rotation2d -- current gyro reading.
- Init -- boolean -- Reset position to the value of Initial Pose. Odometry is calculated from this position.
- Initial Pose -- Pose2d -- Pose to use when Init is TRUE. (Optional. Default: 0,0,0)

Returns

- Current Pose -- Pose2d -- Current position calculated by Swerve Odometry
- Error -- boolean -- If TRUE an error has occurred
- Swerve_Drive_Odometry -- cluster -- current value of the swerve drive odometry cluster.

SwerveOdometry_GetPosition



Returns the position of the robot on the field.

Parameters:

- In SwerveDriveOdometry - Input data structure

Returns:

- Pose - The pose of the robot (x and y are in meters).

SwerveOdometry_New



Constructs a SwerveDriveOdometry object.

Parameters:

- gyroAngle - The angle reported by the gyroscope.
- ModulePostions -- Array of module positions (distances)
- initialPose The starting position of the robot on the field.

Returns

- SwerveDriveOdometry - Data Structure

SwerveOdometry_NewZeroCenter



Constructs a SwerveDriveOdometry object with the default pose at the origin.

Parameters:

- gyroAngle - The angle reported by the gyroscope.
- ModulePostions -- Array of module positions (distances)

Returns:

- SwerveDriveOdometry - Data Structure

SwerveOdometry_ResetPosition



Resets the robot's position on the field.

The gyroscope angle does not need to be reset here on the user's robot code. The library automatically takes care of offsetting the gyro angle.

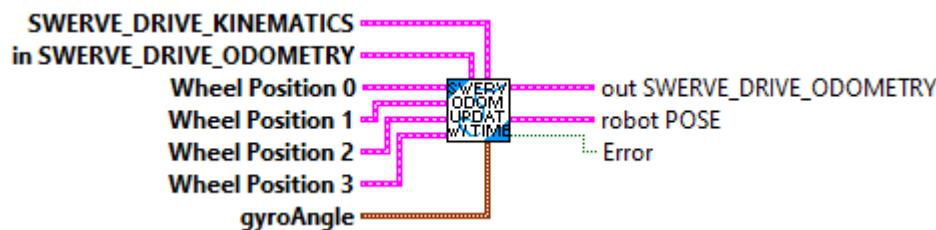
Parameters:

- in SwerveDriveOdometry -- input SwerveDriveOdometry data structure
- pose - The position on the field that your robot is at.
- gyroAngle - The angle reported by the gyroscope.

Returns:

- out SwerveDriveOdometry -- Updated data structure

SwerveOdometry_Update4



Updates the robot's position on the field using forward kinematics and integration of the pose over time. This also takes in an angle parameter which is used instead of the angular rate that is calculated from forward kinematics.

This subVI is customized for 4 swerve modules.

Parameters:

- InSwerveDriveOdometry - This data structure
- gyroAngle - The angle reported by the gyroscope.
- modulePosition 0 - The current position (distance) of a swerve module. Please provide the states in the same order in which you instantiated your SwerveDriveKinematics.
- modulePosition 1 - The current position (distance) of a swerve module.
- modulePosition 2 - The current position (distance) of a swerve module.
- modulePosition 3 - The current position (distance) of a swerve module.

Returns:

- OutSwerveDriveOdometry - Updated data structure
- The new pose of the robot.
- Error -- TRUE if an error occurred (number of modules doesn't match)

SwerveOdometry_UpdateX



Updates the robot's position on the field using forward kinematics and integration of the pose over time. This method takes in the current time as a parameter to calculate period (difference between two timestamps). The period is used to calculate the change in distance from a velocity. This also takes in an angle parameter which is used instead of the angular rate that is calculated from forward kinematics.

Parameters:

- InSwerveDriveKinematics - The data structure

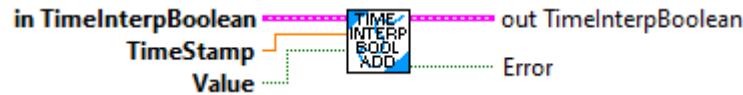
- InSwerveDriveOdometry - This data structure
- gyroAngle - The angle reported by the gyroscope.
- modulePositions - The current positions (distances) of all swerve modules. Please provide the states in the same order in which you instantiated your SwerveDriveKinematics.

Returns:

- OutSwerveDriveOdometry - Updated data structure
- The new pose of the robot.
- Error -- TRUE if an error occurred (number of modules doesn't match)

TimeInterpBoolean

TimeInterpBoolean_AddSample



Add a sample to the buffer.

Inputs:

- In TimeInterpBoolean -- Time Interp data cluster
- Time -- The time stamp of the sample (seconds)
- Value -- The boolean value

Outputs:

- out TimeInterpBoolean -- Updated Time Interp data cluster

TimeInterpBoolean_CleanUp



Removes samples older than our current history size.

Inputs:

- In TimeInterpBoolean -- Input data cluster.

Outputs:

- Out TimeInterpBoolean -- Updated data cluster.
- Error -- Returns TRUE if an error occurred.

TimeInterpBoolean_Clear



Removes all samples from the history buffer.

Inputs:

- In TimeInterpBoolean -- Input data cluster.

Outputs:

- Out TimeInterpBoolean -- Updated data cluster.

TimeInterpBoolean_GetNewestSample



Get the newest sample

Inputs:

- TimeInterpBoolean -- Input data cluster

Outputs

- OutputBoolean -- Newest Boolean.
- TimeStamp -- Time for this sample (seconds)
- IsPresent -- Returns TRUE if the buffer contains any data

TimeInterpBoolean_GetSample



Sample the buffer at the given time.

This returns the value sample at or before the requested time. (boolean values can't be interpolated.)

Inputs:

- TimeInterpBoolean -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputBoolean -- Sampled boolean. This is an exact value if there is a sample in the buffer at this time. Otherwise it is the next previous sample.
 - IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.
-

TimeInterpBoolean_New



Create the data cluster for a TIME_INTERPOLATABLE_BOOLEAN.

The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_BOOLEAN stores and returns boolean values.

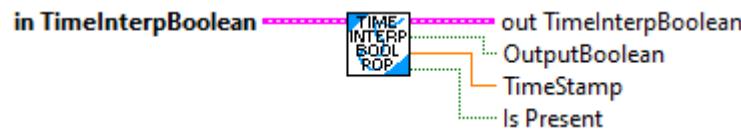
Inputs:

- Max Time -- The maximum time (seconds) of data to store in the buffer.

Outputs:

- Time Inter Boolean -- Created data structure cluster
-

TimeInterpBoolean_PopOldestSample



Sample the buffer at the given time.

Inputs:

- TimeInterpVariant -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputPose -- Sampled Variant. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated. A custom routine will need to be called to extract the custom data from the variant.
 - IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.
-

TimeInterpBoolean_SetMaxTime



Set the maximum time period that should be stored in the buffer

Inputs:

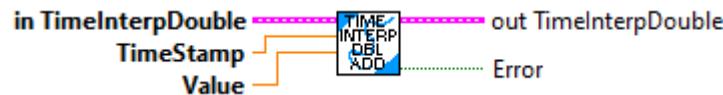
- In TimeInterpBoolean -- Input data cluster.
- Max Time -- Maximum time period to store in the buffer (seconds)

Outputs:

- Out TimeInterpBoolean -- Updated data cluster.

TimeInterpDouble

TimeInterpDouble_AddSample



Add a sample to the buffer.

Inputs:

- In TimeInterpDouble -- Time Interp data cluster
- Time -- The time stamp of the sample (seconds)
- Value -- The double value

Outputs:

- out TimeInterpDouble -- Updated Time Interp data cluster
-
-

TimeInterpDouble_CleanUp



Removes samples older than our current history size.

Inputs:

- In TimeInterpDouble -- Input data cluster.

Outputs:

- Out TimeInterpDouble -- Updated data cluster.
 - Error -- Returns TRUE if an error occurred.
-
-

TimeInterpDouble_Clear



Removes all samples from the history buffer.

Inputs:

- In TimeInterpDouble -- Input data cluster.

Outputs:

- Out TimeInterpDouble -- Updated data cluster.
-
-

TimeInterpDouble_GetNewestSample



Get the newest sample

Inputs:

- TimeInterpDouble -- Input data cluster

Outputs

- OutputDouble -- Newest Double.
 - TimeStamp -- Time for this sample (seconds)
 - IsPresent -- Returns TRUE if the buffer contains any data
-
-

TimeInterpDouble_GetSample



Sample the buffer at the given time.

Inputs:

- TimeInterpDouble -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputDouble -- Sampled Double. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated.

- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.

TimeInterpDouble_GetTimeForValue



Find the time for a particular value in the buffer. The first instance of a value is found. If an exact match isn't found, interpolation is used to calculate the time for the value. If there multiple instances of a value exist, only the first one is returned.

Inputs:

- TimeInterpDouble -- Input data cluster
- Value -- The value to search for.

Outputs

- TimeState -- Time stamp for the input value.
- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.

TimeInterpDouble_New



Create the data cluster for a TIME_INTERPOLATABLE_DOUBLE.

The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_DOUBLE stores and returns DOUBLE values.

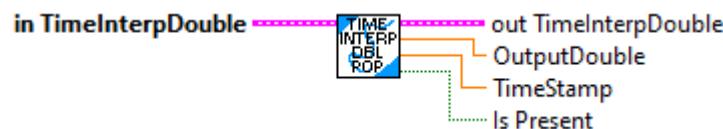
Inputs:

- Max Time -- The maximum time (seconds) of data to store in the buffer.

Outputs:

- Time Inter Double -- Created data structure cluster

TimeInterpDouble_PopOldestSample



Sample the buffer at the given time.

Inputs:

- TimeInterpVariant -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputPose -- Sampled Variant. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated. A custom routine will need to be called to extract the custom data from the variant.
- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.

TimeInterpDouble_SetMaxTime



Set the maximum time period that should be stored in the buffer

Inputs:

- In TimeInterpDouble -- Input data cluster.
- Max Time -- Maximum time period to store in the buffer (seconds)

Outputs:

- Out TimeInterpDouble -- Updated data cluster.

TimeInterpPose2d

TimeInterpPose2d_AddSample



Add a sample to the buffer.

Inputs:

- In TimeInterpPose2d -- Time Interp data cluster
- Time -- The time stamp of the sample (seconds)
- Value -- The Pose2d value

Outputs:

- out TimeInterpPose2d -- Updated Time Interp data cluster

TimeInterpPose2d_CleanUp



Removes samples older than our current history size.

Inputs:

- In TimeInterpPose2d -- Input data cluster.

Outputs:

- Out TimeInterpPose2d -- Updated data cluster.
- Error -- Returns TRUE if an error occurred.

TimeInterpPose2d_Clear



Removes all samples from the history buffer.

Inputs:

- In TimeInterpPose2d -- Input data cluster.

Outputs:

- Out TimeInterpPose2d -- Updated data cluster.

TimeInterpPose2d_GetNewestSample



Get the newest sample

Inputs:

- TimeInterpPose2d -- Input data cluster

Outputs

- OutputPose2d -- Newest Pose2d.
- TimeStamp -- Time for this sample (seconds)
- IsPresent -- Returns TRUE if the buffer contains any data

TimeInterpPose2d_GetSample



Sample the buffer at the given time.

Inputs:

- TimeInterpPose2d -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputPose -- Sampled Pose2d. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated.

- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.

TimeInterpPose2d_New

Create the data cluster for a TIME_INTERPOLATABLE_POSE2D.

The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_POSE2D stores and returns POSE2D values.

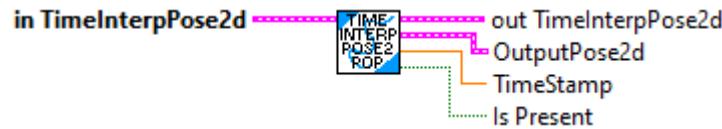
Inputs:

- Max Time -- The maximum time (seconds) of data to store in the buffer.

Outputs:

- Time Inter Pose2d -- Created data structure cluster

TimeInterpPose2d_PopOldestSample



Sample the buffer at the given time.

Inputs:

- TimeInterpVariant -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputPose -- Sampled Variant. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated. A custom routine will need to be called to extract the custom data from the variant.
 - IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.
-

TimeInterpPose2d_SetMaxTime



Sets the maximum time period that should be stored in the buffer.

Inputs:

- In TimeInterpPose2d -- Input data cluster.
- Max Time -- Maximum time period to store in the buffer (seconds)

Outputs:

- Out TimeInterpPose2d -- Updated data cluster.

TimeInterpRotation2d

TimeInterpRotation2d_AddSample



Add a sample to the buffer.

Inputs:

- In TimeInterpRotation2d -- Time Interp data cluster
- Time -- The time stamp of the sample (seconds)
- Value -- The Rotation2d value

Outputs:

- out TimeInterpRotation2d -- Updated Time Interp data cluster

TimeInterpRotation2d_CleanUp



Removes samples older than our current history size.

Inputs:

- In TimeInterpRotation2d -- Input data cluster.

Outputs:

- Out TimeInterpRotation2d -- Updated data cluster.
- Error -- Returns TRUE if an error occurred.

TimeInterpRotation2d_Clear



Removes all samples from the history buffer.

Inputs:

- In TimeInterpRotation2d -- Input data cluster.

Outputs:

- Out TimeInterpRotation2d -- Updated data cluster.

TimeInterpRotation2d_GetNewestSample



Get the newest sample

Inputs:

- TimeInterpRotatin2d -- Input data cluster

Outputs

- OutputRotatin2d -- Newest Rotation2d.
- TimeStamp -- Time for this sample (seconds)
- IsPresent -- Returns TRUE if the buffer contains any data

TimeInterpRotation2d_GetSample



Sample the buffer at the given time.

Inputs:

- TimeInterpRotation2d -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputRotation -- Sampled Rotation2d. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated.

- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.

TimeInterpRotation2d_New

Create the data cluster for a TIME_INTERPOLATABLE_ROTATION2D.

The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_ROTATION2D stores and returns Rotation2d values.

Inputs:

- Max Time -- The maximum time (seconds) of data to store in the buffer.

Outputs:

- Time Inter Rotation2d -- Created data structure cluster
-
-

TimeInterpRotation2d_PopOldestSample



Sample the buffer at the given time.

Inputs:

- TimeInterpVariant -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputPose -- Sampled Variant. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated. A custom routine will need to be called to extract the custom data from the variant.
 - IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.
-

TimeInterpRotation2d_SetMaxTime



Sets the maximum time period that should be stored in the buffer.

Inputs:

- In TimeInterpRotation2d -- Input data cluster.
- Max Time -- Maximum time period to store in the buffer (seconds)

Outputs:

- Out TimeInterpRotation2d -- Updated data cluster.

TimeInterpVariant

TimeInterpVariant_AddSample



Add a sample to the buffer. Only one value for a given time is allowed to exist. New values provided at the exact time of an existing value replace the existing value.

Inputs:

- In TimeInterpVariant -- Time Interp data cluster
- Time -- The time stamp of the sample (seconds)
- Value -- The Variant value. Any data type can be provided to this terminal. It is converted to a variant.

Outputs:

- out TimeInterpVariant -- Updated Time Interp data cluster

TimeInterpVariant_CleanUp



Removes samples older than our current history size.

Inputs:

- In TimeInterpPose2d -- Input data cluster.

Outputs:

- Out TimeInterpPose2d -- Updated data cluster.
- Error -- Returns TRUE if an error occurred.

TimeInterpVariant_Clear



Removes all samples from the history buffer.

Inputs:

- In TimeInterpPose2d -- Input data cluster.

Outputs:

- Out TimeInterpPose2d -- Updated data cluster.

TimeInterpVariant_GetNewestSample



Get the newest sample

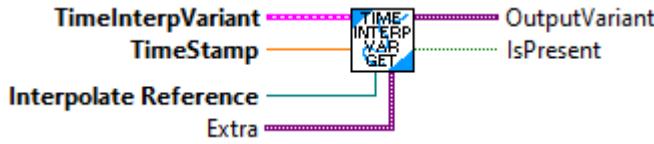
Inputs:

- TimeInterpVariant -- Input data cluster

Outputs

- OutputVariant -- Newest Variant.
- TimeStamp -- Time for this sample (seconds)
- IsPresent -- Returns TRUE if the buffer contains any data

TimeInterpVariant_GetSample



Sample the buffer at the given time.

Inputs:

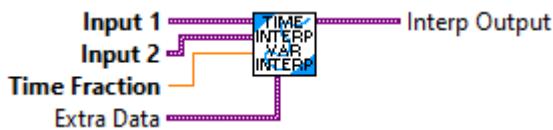
- TimeInterpVariant -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputPose -- Sampled Variant. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated. A custom routine will need to be called to extract the custom data from the variant.

- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.

TimeInterpVariant_Interpolate



TimeInterpVariant_New



Create the data cluster for a TIME_INTERPOLATABLE_VARIANT

The TimeInterpolatableVariant provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_VARIANT stores and returns Variant values. Because this routine stores and retrieves Variant values, several custom routines will need to be provided to effectively use this set of functions:

- A TypeDef for the cluster containing the data to be sorted.
- A routine to pack data into the custom TypeDef
- A routine to unpack the variant data back to the custom data type.
- A routine to interpolate two values. A sample routine,

TimeInterpVariant_Interpolate.vi, is provided to use as a template to create the custom interpolation routine.

Inputs:

- Max Time -- The maximum time (seconds) of data to store in the buffer.

Outputs:

- Time InterpVariant -- Created data structure cluster

TimeInterpVariant_PopOldestSample



Sample the buffer at the given time.

Inputs:

- TimeInterpVariant -- Input data cluster
- TimeStamp -- Time at which to sample (seconds)

Outputs

- OutputVariant -- Sampled Variant. This is an exact value if there is a sample in the buffer at this time. Otherwise the value is interpolated. A custom routine will need to be called to extract the custom data from the variant.

- IsPresent -- Returns TRUE if the buffer contains data as far back as TimeStamp.
-

TimeInterpVariant_SetMaxTime



Sets the maximum time period that should be stored in the buffer.

Inputs:

- In TimeInterpVariant -- Input data cluster.
- Max Time -- Maximum time period to store in the buffer (seconds)

Outputs:

- Out TimeInterpVariant -- Updated data cluster.

Timer

Timer_Close

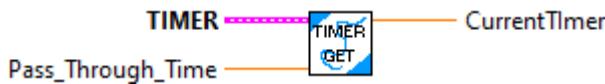


Releases resources used by a Timer data cluster. Specifically the semaphore is deleted.

Inputs:

- Timer -- The Timer data cluster
-
-

Timer_Get



Get the current time from the timer. If the clock is running it is derived from the current system clock the start time stored in the timer class. If the clock is not running, then return the time when it was last stopped.

Inputs:

- Timer -- The Timer data cluster

Outputs:

- CurrentTimer -- Current time value for this timer in seconds
-
-

Timer_GetAndReset



Synchronized call to get and reset the timer. Get the current time from the timer. If the clock is running it is derived from the current system clock the start time stored in the timer class. If the clock is not running, then return the time when it was last stopped. Also reset the timer by setting the time to 0. Make the timer startTime the current time so new requests will be relative now.

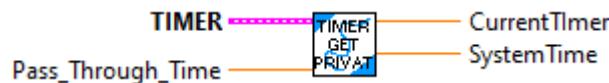
Inputs:

- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster
- CurrentTimer -- Current time value for this timer in seconds

Timer_GetInternal



THIS IS AN INTERNAL ROUTINE. DO NOT USE.

Get the current time from the timer. If the clock is running it is derived from the current system clock the start time stored in the timer class. If the clock is not running, then return the time when it was last stopped.

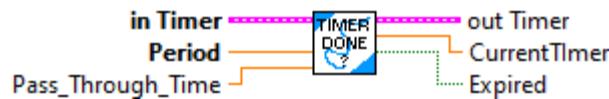
Inputs:

- Timer -- The Timer data cluster

Outputs:

- CurrentTimer -- Current time value for this timer in seconds
- SystemTime -- Current system time

Timer_HasPeriodPassed



Check if the period specified has passed and if it has, advance the start time by that period. This is useful to decide if it's time to do periodic work without drifting later by the time it took to get around to checking.

Inputs:

- Timer -- The Timer data cluster
- period -- The period to check for (in seconds).

Outputs:

- Timer -- The modified Timer data cluster
- CurrentTimer -- Current time value for this timer in seconds
- Expired -- True if the period has passed.

Timer_HasPeriodPassedOnce



Check if the period specified has passed and if it has, the timer is stopped and reset. This is useful for timers that have to be timers that have to be specifically re-started after they have expired.

Inputs:

- Timer -- The Timer data cluster
- period -- The period to check for (in seconds).

Outputs:

- Timer -- The modified Timer data cluster
 - CurrentTimer -- Current time value for this timer in seconds
 - Expired -- True if the period has passed.
-
-

Timer_New

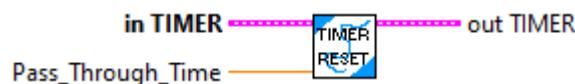


Creates and initializes a Timer data cluster.

Outputs:

- Timer -- The initialized Timer data cluster
-
-

Timer_Reset



Reset the timer by setting the time to 0. Make the timer startTime the current time so new requests will be relative now

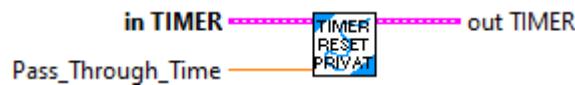
Inputs:

- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster
-
-

Timer_ResetInternal



THIS IS AN INTERNAL ROUTINE. DO NOT USE.

Reset the timer by setting the time to 0. Make the timer startTime the current time so new requests will be relative now

Inputs:

- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster

Timer_Restart



Restart the timer by stopping the timer, if it is not already stopped, resetting the accumulated time, then starting the timer again. If you want an event to periodically reoccur at some time interval from the start time, consider using HasPeriodPassed instead.

Inputs:

- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster

Timer_Start



Start the timer running. Just set the running flag to true indicating that all time requests should be relative to the system clock.

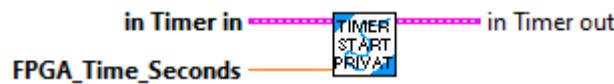
Inputs:

- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster
-
-

Timer_Start_Internal



Start Timer Internal -- This is a private internal routine. It should NEVER be called by users.

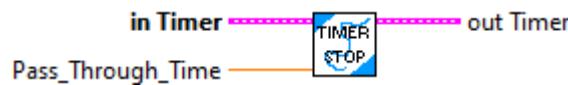
Input:

- in Timer -- data cluster
- PassThroughTime -- fpga time

Output

- out Timer -- data cluster
-
-

Timer_Stop



Stop the timer. This computes the time as of now and clears the running flag, causing all subsequent time requests to be read from the accumulated time rather than looking at the system clock.

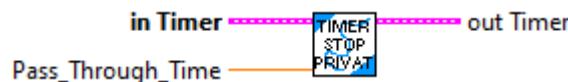
Inputs:

- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster

Timer_StopInternal



THIS IS AN INTERNAL ROUTINE. DO NOT USE.

Stop the timer. This computes the time as of now and clears the running flag, causing all subsequent time requests to be read from the accumulated time rather than looking at the system clock.

Inputs:

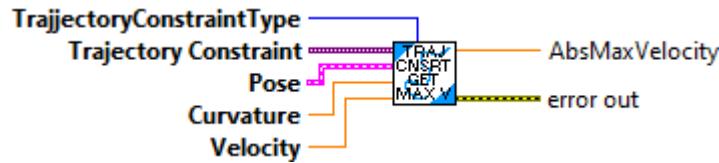
- Timer -- The Timer data cluster

Outputs:

- Timer -- The modified Timer data cluster

TrajConstraint

TrajConstraint_GetMaxVelocity



Calculate the Maximum velocity for the provided constraint and conditions.

Note that this only works for regular constraints. Constraints that contain constraints, such as the Rectangular Region and Elliptical Region, are not calculated by this routine.

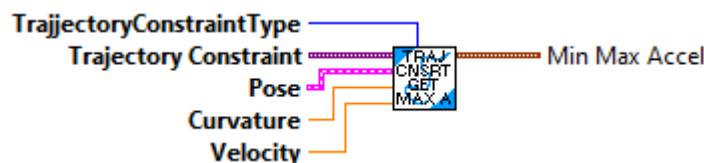
Parameters:

- TrajectoryConstraintType -- Enum indicating the type of constraint being passed.
- Constraint - Variant containing the constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- maxVelocity - Maximum allowed velocity.

TrajConstraint_GetMinMaxAccel



Return the minimum and maximum allowed acceleration given the provided conditions.

Note that this only works for regular constraints. Constraints that contain constraints, such as the Rectangular Region and Elliptical Region, are not calculated by this routine.

Parameters:

- TrajectoryConstraintType -- Enum indicating the type of constraint being passed.
- Constraint - Variant containing the constraint data structure
- statePose - current traj state Pose
- curvature - current traj curvature
- maxVelocity - current traj max velocity

Returns

- MinMaxVelocity - Data cluster containing the Minimum and Maximum acceleration..
-

TrajConstraint_GetType

Determines the type of a passed trajectory constraint.

Parameters:

- Constraint - Variant containing the constraint to determine the type.

Returns:

- TrajectoryConstraintType -- Enum containing the constraint type
- Type Found -- Returns TRUE if the variant is a valid constraint type.

Trajectory

Trajectory_Concatenate



Concatenates another trajectory to the current trajectory. The user is responsible for making sure that the end pose of this trajectory and the start pose of the other trajectory match (if that is the desired behavior).

Inputs:

- Trajectory 1 -- Initial trajectory
- Trajectory 2 -- Trajectory to add to end of Trajectory 1

Outputs:

- OutTrajectory -- Concatenated trajectory

Trajectory_Equals



Determines if two trajectories are equal.

Parameters:

- Trajectory - Trajectory data structure
- Other Trajectory - Trajectory data structure

Returns:

- Equals - boolean indicating if the two trajectories are equal.

Trajectory_GetStates



Gets the array of trajectory states for this trajectory

Parameters:

- Trajectory - Trajectory data structure

Returns:

- States - Array of trajectory states for this trajectory

Trajectory_GetTotalTime



Gets the total time in seconds for this trajectory

Parameters:

- Trajectory - Trajectory data structure

Returns:

- TotalTime_SEC - Total trajectory time in seconds.

Trajectory_New



Constructs a trajectory from a vector of states.

Represents a time-parameterized trajectory. The trajectory contains of various States that represent the pose, curvature, time elapsed, velocity, and acceleration at that point.

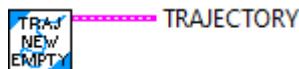
Parameters:

- TrajStates - A vector of states.

Returns:

- Trajectory - Trajectory data structure
-

Trajectory_New_Empty



Constructs an empty trajectory.

Represents a time-parameterized trajectory. The trajectory contains of various States that represent the pose, curvature, time elapsed, velocity, and acceleration at that point.

Parameters:

-

Returns:

- Trajectory - Trajectory data structure
-

Trajectory_RelativeTo



Transforms all poses in the trajectory so that they are relative to the given pose. This is useful for converting a field-relative trajectory into a robot-relative trajectory.

Parameters:

- InputTrajectory - Trajectory data structure
- pose -The pose that is the origin of the coordinate frame that the current trajectory will be transformed into.

Returns:

- OutputTrajectory - The transformed trajectory.
-

Trajectory_Sample



Sample the trajectory at a point in time.

Parameters:

- InputTrajectory - The Trajectory data structure
- time - The point in time since the beginning of the trajectory to sample. (seconds)

Returns:

- OutputTrajState - The state at that point in time.
-

Trajectory_SampleReverse



Sample the trajectory at a point in time. The trajectory is sampled in reverse such that the ending time becomes 0 and the beginning time is the maximum time. The selected trajectory state is then transformed to be relative to end of the trajectory.

For this subVI to work correctly, sampling must start at time 0.0.

The result is a relative trajectory which starts at 0,0,0. This assumes that the robot follows the trajectory such that the orientation of the robot equals the ending state of the trajectory.

This routine does not exist in WPILIB

Parameters:

- InputTrajectory - The Trajectory data structure
- time - The point in time since the beginning of the trajectory to sample. (seconds)

Returns:

- OutputTrajState - The state at that point in time.
-
-

Trajectory_TransformBy



Transforms all poses in the trajectory by the given transform. This is useful for converting a robot-relative trajectory into a field-relative trajectory. This works with respect to the first pose in the trajectory.

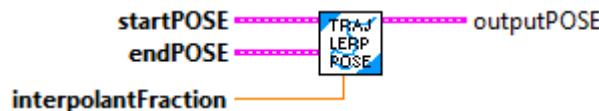
Parameters:

- InputTrajectory - Trajectory data structure
- transform - The transform to transform the trajectory by.

Returns:

- OutputTrajectory - The transformed trajectory.
-
-

Trajectory_lerp_Pose



This is an internal routine. Linearly interpolates between two values.

Parameters:

- startPose - The start Pose.
- endPose - The end Pose.
- fraction - The fraction for interpolation.

Return:

- outputPose - The interpolated Pose.
-
-

Trajectory_lerp_double



This is an internal routine. Linearly interpolates between two values.

Parameters:

- startValue - The start value.
- endValue - The end value.
- fraction - The fraction for interpolation.

Return:

- outputValue - The interpolated value.

TrajectoryConfig

TrajectoryConfig_AddConstraint



Add a constraint to this Trajectory Configuration

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure
- Constraint -- Trajectory constraint converted to a variant

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryConfig_AddConstraints



Add an array of constraints to this Trajectory Configuration

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure
- Constraints -- Array of Trajectory constraint converted to a variant

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryConfig_Create



Constructs the trajectory configuration class.

Represents the configuration for generating a trajectory. This class stores the start velocity, end velocity, max velocity, max acceleration, custom constraints, and the reversed flag.

The cluster must be constructed with a max velocity and max acceleration. The other parameters (start velocity, end velocity, constraints, reversed) have been defaulted to reasonable values (0, 0, {}, false). These values can be changed via the setXXX methods.

It also contains the data for each constraint and a flag indicating if a particular constraint is active. As new constraints are added, this cluster will be modified to contain them.

Parameters:

- maxVelocity - The max velocity for the trajectory. (Meters/Sec)
- maxAcceleration - The max acceleration for the trajectory. (Meters/Sec²)

Returns:

- TrajectoryConfig - TrajectoryConfig data structure

TrajectoryConfig_GetCentripetalAccelConstraint



Retrieves the Centripetal Acceleration constraint from the Trajectory Configuration.

Note: If more than one instance of the Centripetal Acceleration constraint exists only the first one is returned. (Only one of these should be defined.)

Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- MaxCentAccel -- The data cluster, which contains the maximum centripetal acceleration (meters/sec²)
 - Constraint Exists -- Returns TRUE if this constraint is defined in the trajectory configuration.
-

TrajectoryConfig_GetConstraints



Return the array of Constraints contained in the Trajectory Configuration.

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- Constraints -- Array of variants containing the data clusters for trajectory constraints.
 - OutTrajectoryConfig - Modified trajectory configuration data structure
-

TrajectoryConfig_GetEndVelocity



Retrieve the end velocity from the Trajectory Configuration

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- End Velocity -- The desired velocity at the end of the trajectory.

TrajectoryConfig_GetKinematicsDiffDriveConstraint



Retrieves the Differential Drive Kinematics constraint from the Trajectory Configuration.

Note: If more than once instance of the Differential Drive Kinematics constraint exists only the first one is returned. (Only one of these should be defined.)

Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- Diff Drive Kinematics Constraint -- The data cluster, which contains the constraint.
- Constraint Exists -- Returns TRUE if this constraint is defined in the trjaectory configuration.

TrajectoryConfig_GetKinematicsMecanumDriveConstraint



Retrieves the Mecanum Kinematics constraint from the Trajectory Configuration.

Note: If more than once instance of the Mecanum Kinematics constraint exists only the first one is returned. (Only one of these should be defined.)

Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- Mecanum Kinematics Constraint -- The data cluster, which contains the constraint
- Constraint Exists -- Returns TRUE if this constraint is defined in the trjaectory configuration.

TrajectoryConfig_GetKinematicsSwerveDriveConstraint



Retrieves the Swerve Drive Kinematics constraint from the Trajectory Configuration.

Note: If more than once instance of the Swerve Drive Kinematics constraint exists only the first one is returned. (Only one of these should be defined.)

Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- Swerve Drive Kinematics Constraint -- The data cluster containing the constraint
- Constraint Exists -- Returns TRUE if this constraint is defined in the trajectory configuration.

TrajectoryConfig_GetMaxVelAccel



Retrieves the Maximum Velocity and Acceleration from the Trajectory Configuration.

Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- Maximum Velocity -- Maximum velocity defined in configuration
- Maximum Acceleration -- Maximum acceleration defined in configuration.

TrajectoryConfig_GetStartVelocity



Retrieves the desired Starting Velocity from the Trajectory Configuration.

Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- Starting Velocity -- The desired starting velocity from the configuration

TrajectoryConfig_GetVoltageDiffDriveConstraint



Retrieves the Differential Drive voltage constraint from the Trajectory Configuration.

Note: If more than once instance of the Differential Drive Voltage constraint exists only the first one is returned. (Only one of these should be defined.)

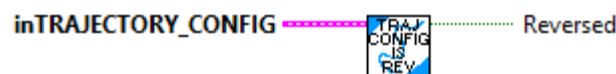
Parameters:

- InTrajectoryConfig -- The trajectory configuration data structure

Returns:

- Diff Drive Voltage Constraint -- The data cluster containing the constraint.
- Constraint Exists -- Returns TRUE if this constraint is defined in the trjaectory configuration.

TrajectoryConfig_IsReversed



Retrieves the reversed flag from the Trajectory Config data structure.

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- Reversed - Whether the trajectory should be reversed or not.

TrajectoryConfig_SetEndVelocity



Updates the desired End Velocity in the Trajectory Config data structure.

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure
- End Velocity - Desired velocity at the completion of the trajectory..

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryConfig_SetStartVelocity



Updates the desired Start Velocity in the Trajectory Config data structure.

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure
- Start Velocity -- The desired trajectory starting velocity..

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryConfig_setCentripetalAccel



Adds and enables a centripetal acceleration constraint to ensure that the rotating capability of the robot is not exceeded.

Parameters:

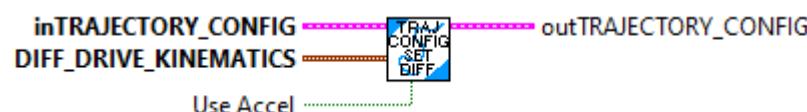
- MaxCentAccel - The maximum centripetal acceleration (meters/sec²)
- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

modified to contain them.

TrajectoryConfig_setKinematicsDiffDrive



Adds and enables a differential drive kinematics constraint to ensure that no wheel velocity of a differential drive goes above the max velocity. The TrajectoryConfiguration max velocity is used for the differential drive max individual wheel velocity.

Parameters:

- DiffDriveKinematics - The differential drive kinematics data structure
- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure
-
-

TrajectoryConfig_setKinematicsMecanumDrive

Adds and enables a mecanum drive kinematics constraint to ensure that no wheel velocity of a mecanum drive goes above the max velocity. The TrajectoryConfiguration max velocity is used for the mecanum drive max individual wheel velocity.

Parameters:

- MecanumDriveKinematics - The mecanum drive kinematics data structure
- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure
-
-

TrajectoryConfig_setKinematicsSwerveDrive

Adds and enables a swerve drive kinematics constraint to ensure that no wheel velocity of a swerve drive goes above the max velocity. The TrajectoryConfiguration max velocity is used for the swerve drive max individual wheel velocity.

Parameters:

- SwerveDriveKinematics - The swerve drive kinematics data structure
- InTrajectoryConfig - The trajectory configuration data structure

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryConfig_setReversed



Updates the reversed flag in the Trajectory Config data structure.

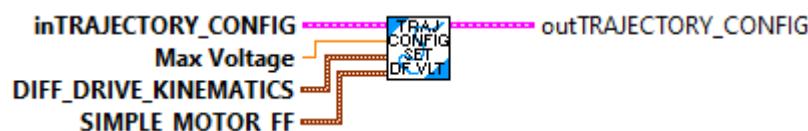
Parameters:

- InTrajectoryConfig - The trajectory configuration data structure
- Reversed - Whether the trajectory should be reversed or not.

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryConfig_setVoltageDiffDrive



Adds and enables a differential drive voltage constraint to ensure that no wheel velocity or acceleration of a differential drive goes above the attainable values.

Parameters:

- InTrajectoryConfig - The trajectory configuration data structure

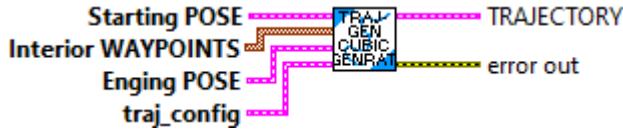
- maxVoltage - The maximum voltage available to the motors while following the path. Should be somewhat less than the nominal battery voltage (12V) to account for "voltage sag" due to current draw.
- DiffDriveKinematics - A kinematics component describing the drive geometry.
- SimpleMotorFeedforward - A feedforward component describing the behavior of the drive.

Returns:

- OutTrajectoryConfig - Modified trajectory configuration data structure

TrajectoryGenerate

TrajectoryGenerate_Make_Cubic



Generates a trajectory from the given waypoints and config. This method uses clamped cubic splines -- a method in which the initial pose, final pose, and interior waypoints are provided. The headings are automatically determined at the interior points to ensure continuous curvature.

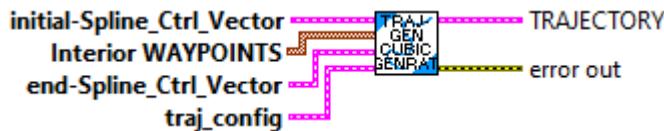
Parameters:

- startingPose - The initial Pose.
- interiorWaypoints - The interior waypoints.
- ending Pose - The ending Pose.
- Trajectory Config - The configuration for the trajectory.

Returns:

- Trajectory -- Created trajectory data structure
 - Error Out - Returned error cluster
-
-

TrajectoryGenerate_Make_Cubic_CtrlVect



Generates a trajectory from the given control vectors and config. This method uses clamped cubic splines -- a method in which the exterior control vectors and interior waypoints are provided. The headings are automatically determined at the interior points to ensure continuous curvature.

Parameters:

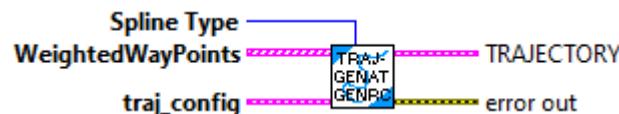
- initialCtrlVector - The initial control vector.

- interiorWaypoints - The interior waypoints.
- endCtrlVector - The ending control vector.
- Trajectory Config - The configuration for the trajectory.

Returns:

- Trajectory -- Created trajectory data structure
 - Error Out - Returned error cluster
-

TrajectoryGenerate_Make_Generic



Generates a trajectory from the given waypoints and config. The type of spline used and whether to use weights or auto-calcualte weights is provided as an input.

Parameters:

- Spline Type -- An enumerated value indicating what type of spline to use when creating the trajectory.
- WeightedWaypoints - The array of waypoints. If weights are not to be used, set the values to zero..
- Trajectory Config - The configuration for the trajectory.

Returns:

- Trajectory -- Created trajectory data structure
 - Error Out - Returned error cluster
-

TrajectoryGenerate_Make_Quintic



Generates a trajectory from the given waypoints and config. This method uses quintic hermite splines -- therefore, all points must be represented by Pose2d objects. Continuous curvature is guaranteed in this method.

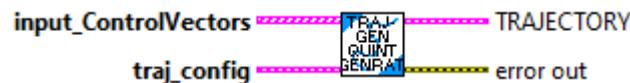
Parameters:

- WaypointPose - Array of waypoints (Pose).
- Trajectory Config - The configuration for the trajectory.

Returns:

- Trajectory -- Created trajectory data structure
 - Error Out - Returned error cluster
-
-

TrajectoryGenerate_Make_Quintic_CtrlVect



Generates a trajectory from the given quintic control vectors and config. This method uses quintic hermite splines -- therefore, all points must be represented by control vectors. Continuous curvature is guaranteed in this method.

Parameters:

- ControlVectors - Array of quintic control vectors.
- Trajectory Config - The configuration for the trajectory.

Returns:

- Trajectory -- Created trajectory data structure
 - Error Out - Returned error cluster
-
-

TrajectoryGenerate_Make_Quintic_Weighted



Generates a trajectory from the given weighted waypoints and config. This method uses quintic hermite splines -- therefore, all points must be represented by Pose2d objects. Continuous curvature is guaranteed in this method.

Parameters:

- WeightedWaypointPose - Array of weighted waypoints .
- Trajectory Config - The configuration for the trajectory.

Returns:

- Trajectory -- Created trajectory data structure
- Error Out - Returned error cluster

TrajectoryGenerate_splinePointsFromSplines



Generate spline points from a vector of splines by parameterizing the splines.

Parameters:

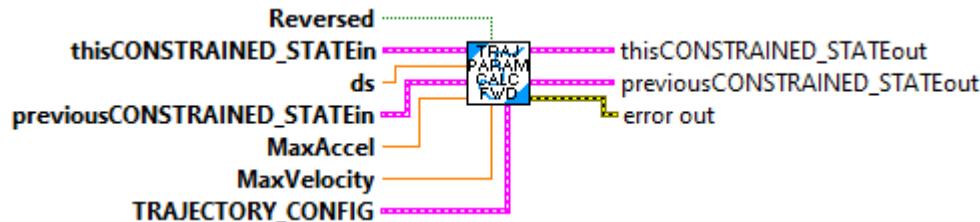
- inputSpline - Array of splines to parameterize.

Returns:

- ArrayPoseWithCurvature - The spline points for use in time parameterization of a trajectory.
- Error Out - Returned error cluster

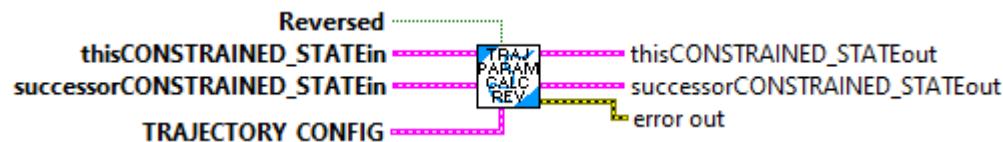
TrajectoryParam

TrajectoryParam_calcStuffFwd



Internal routine called by TrajectoryParam_timeParam.VI

TrajectoryParam_calcStuffRev



Internal routine called by TrajectoryParam_timeParam.VI

TrajectoryParam_enforceAccel



Internal routine called by TrajectoryParam_timeParam.VI

This routine enforces acceleration constraints and updates the ConstrainedState value based on the calculated limits.

This routine will need to be updated whenever a new constrained state type is added.

Parameters:

- InConstrainedState
- Reversed
- TrajectoryConfig

Returns

- OutConstrainedState

TrajectoryParam_enforceVelocity



Internal routine called by TrajectoryParam_timeParam.VI

This routine enforces velocity constraints and updates the ConstrainedState value based on the calculated limits.

This routine will need to be updated whenever a new constrained state type is added.

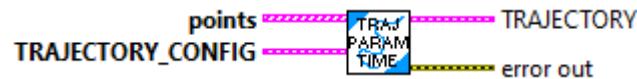
Parameters:

- InConstrainedState
- TrajectoryConfig

Returns

- OutConstrainedState
- error out

TrajectoryParam_timeParam



Parameterize the trajectory by time. This is where the velocity profile is generated.

The derivation of the algorithm used can be found here
"<http://www2.informatik.uni-freiburg.de/~lau/students/Sprunk2008.pdf>"

- points - The spline points.
- TrajectoryConfig - Trajectory Config data structure (it contains - start velocity, end velocity, max velocity, max acceleration, reversed, and constraint data)

Returns:

- Trajectory - Trajectory data structure
- Error Out - Returned error cluster

TrajectoryState

TrajectoryState_Equals



Determines if two Trajectory States are equal

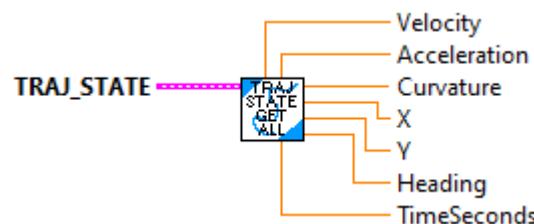
Parameters:

- TrajState - TrajectoryState data structure
- Other TrajState - Other TrajectoryState data structure

Returns:

- Equals - boolean indicating if the states are equal.

TrajectoryState_GetAll



Get individual items from the trajectory state cluster

Parameters:

- TrajState - TrajectoryState data structure

Returns:

- Velocity -- chassis velocity M/S

- Acceleration -- chassis acceleration M/S²
 - Curvature -- chassis curvature RADIANS/METER
 - X -- chassis X position M
 - Y -- chassis Y position M
 - Heading -- chassis heading RADIANS
 - Time -- Sample time SECONDS
-

TrajectoryState_GetPose



Get POSE from the trajectory state cluster

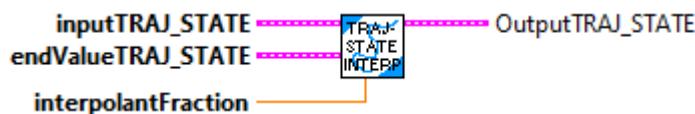
Parameters:

- TrajState - TrajectoryState data structure

Returns:

- TrajStatePose -- POSE data cluster for this trajectory state (X,Y,Heading)

TrajectoryState_Interpolate



Interpolates between two States.

Parameters:

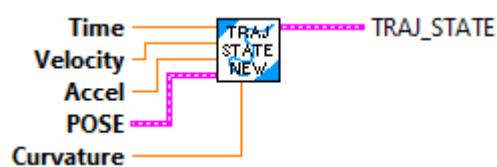
- inputTrajState - The current trajectory state data structure

- endValueTrajState - The end value for the interpolation.
- interpolantFactor - The interpolant (fraction).

Returns:

- TrajState - The interpolated state.

TrajectoryState_New



Constructs a State with the specified parameters.

Parameters:

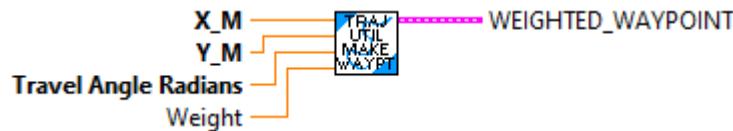
- time - The time elapsed since the beginning of the trajectory. (seconds)
- velocity - The speed at that point of the trajectory. (meters/sec)
- acceleration - The acceleration at that point of the trajectory. (meters/sec²)
- pose - The pose at that point of the trajectory. (meter, radians)
- curvature - The curvature at that point of the trajectory.(radans/meter)

Returns:

- TrajState - TrajectoryState data structure

TrajectoryUtil

TrajectoryUtil_MakeWeightedWayPoint



Creates a weighted waypoint data structure from individual inputs.

Inputs are all SI units (Meters, Radians)

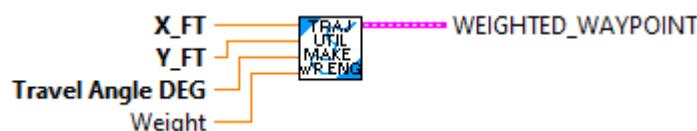
Inputs:

- X_M -- X position (meters)
- Y_M -- Y position (meters)
- Travel Angle -- Desired direction of travel (Radians)
- Weight -- Weight value -- This is an indication of how straight the robot is going when it reaches a waypoint. Larger values specify less curvature at the waypoint.

Outputs:

- WeightedWaypoint -- Data cluster containing created waypoint

TrajectoryUtil_MakeWeightedWayPoint_ENG



Creates a weighted waypoint data structure from individual inputs.

Inputs are all ENG units (Feet, Degrees)

Inputs:

- X_M -- X position (Feet)
- Y_M -- Y position (Feet)
- Travel Angle -- Desired direction of travel (Degrees)
- Weight -- Weight value -- This is an indication of how straight the robot is going when it reaches a waypoint. Larger values specify less curvature at the waypoint.

Outputs:

- WeightedWaypoint -- Data cluster containing created waypoint. The created waypoint uses SI units which are compatible with the internals of the trajectory functions.

TrajectoryUtil_fromPathWeaverJSON



Imports a Trajectory from a PathWeaver-style JSON file.

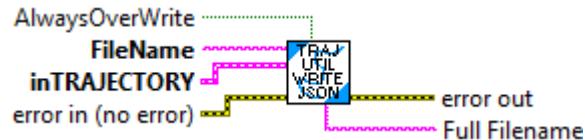
Parameters:

- FileName - File name string of the json file to import from
- Error In - Error cluster (not used)

Returns:

- Trajectory - Trajectory data structure
- Error out - Returned error cluster

TrajectoryUtil_toPathWeaverJSON



Exports a Trajectory to a PathWeaver-style JSON file.

Parameters:

- Always Overwrite -- This optional terminal, if TRUE, will not prompt on WINDOWS to overwrite an existing file. On the RoboRIO the file is always overwritten. (Default: False)
- Trajectory - Trajectory data structure
- FileName - File name string of the export json file
- Error In - Error cluster (not used)

Returns:

- Error out - Returned error cluster

Transform2d

Transform2d_Create_PosePose



Constructs the transform that maps the initial pose to the final pose.

This routine rotates the difference between the translations using a clockwise rotation matrix. This transforms the global delta into a local delta (relative to the initial pose).

Parameters:

- initial - The initial pose for the transformation.
- last - The final pose for the transformation.

Returns:

- Transform - The TRANSFORM data structure

Transform2d_Create_TransRot



Constructs a transform with the given translation and rotation components.

Parameters:

- translation - Translational component of the transform.
- rotation - Rotational component of the transform.

Result:

- transform - TRANSFORM data structure

Transform2d_Div



Divides the transform by the scalar.

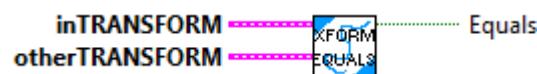
Parameters:

- IN TRANSFORM - This TRANSFORM data structure
- SCALAR - The scalar value to divide the transform by

Results:

- OUT TRANSFORM - The multiplied transform data structure

Transform2d_Equals



Checks equality between this Transform2d and another object.

Parameters:

- IN TRANSFORM - This TRANSFORM data structure
- OTHER TRANSFORM - The other TRANSFORM to compare

Results:

- EQUALS - Returns TRUE when both TRANSFORMs are the same.

Transform2d_GetRotation



Returns the rotational component of the transformation.

Parameters:

- IN TRANSFORM - The TRANSFORM data structure

Returns:

- ROTATION - The rotational component of the transform.

Transform2d_GetTranslation



Returns the translation component of the transformation.

Parameters:

- IN TRANSFORM - This TRANSFORM data structure

Returns:

- TRANSLATION - The translational component of the transform.

Transform2d_GetXY



Returns the X, Y elements of the translation component of the transformation.

Parameters:

- IN TRANSFORM - THis TRANSFORM data structure

Returns:

- X - The X element of the translational component of the transform.
- Y - The Y element of the translational component of the transform.

Transform2d_GetXYAngle



Returns the X, Y, angle elements of the translation component of the transformation.

Parameters:

- IN TRANSFORM - THis TRANSFORM data structure

Returns:

- X - The X element of the translational component of the transform.
- Y - The Y element of the translational component of the transform.
- angle - The angle element of the translational component of the transform.

Transform2d_Inverse



Invert the transformation. This is useful for undoing a transformation.

Parameters:

- IN TRANSFORM - This TRANSFORM data structure

Results:

- OUT TRANSFORM - The inverted transformation.
-
-

Transform2d_Plus



Composes two transformations.

Parameters:

- IN TRANSFORM - This TRANSFORM data structure
- other Transform -- The transform to compose with this one.

Results:

- OUT TRANSFORM - The composition of the two transformations.
-
-

Transform2d_Times



Scales the transform by the scalar.

Parameters:

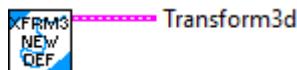
- IN TRANSFORM - This TRANSFORM data structure
- SCALAR - The scalar value to multiply the transform by

Results:

- OUT TRANSFORM - The multiplied transform data structure

Transform3d

Transform3d_Create_Default



Constructs the identity transform -- maps an initial pose to itself.

Parameters:

-- none --

Result:

- transform3d -- TRANSFORM3d data structure

Transform3d_Create_Pose3dPose3d



Constructs the transform that maps the initial pose to the final pose.

Parameters:

- initial pose3d -- The initial pose for the transformation.
- last pose3d -- The final pose for the transformation.

Returns:

- Transform 3d -- The TRANSFORM3d data structure

Transform3d_Create_Trans3dRot3d



Constructs a transform with the given translation and rotation components.

Parameters:

- translation3d -- Translational component of the transform.
- rotation3e -- Rotational component of the transform.

Result:

- transform3d -- TRANSFORM3d data structure

Transform3d_Div



Divides the transform3d by the scalar.

Parameters:

- IN TRANSFORM3d -- This TRANSFORM data structure
- SCALAR -- The scalar value to divide the transform by

Results:

- OUT TRANSFORM3d -- The multiplied transform data structure

Transform3d_Equals



Checks equality between this Transform3d and another Transform3d.

Parameters:

- IN TRANSFORM3d -- This TRANSFORM data structure
- OTHER TRANSFORM3d -- The other TRANSFORM to compare

Results:

- EQUALS - Returns TRUE when both TRANSFORMs are the same.

Transform3d_GetRotation3d



Returns the rotational3d component of the transformation3d.

Parameters:

- IN TRANSFORM3d -- The TRANSFORM3d data structure

Returns:

- ROTATION3d -- The rotational component of the transform.

Transform3d_GetTranslation3d



Returns the translation3d component of the transformation3d.

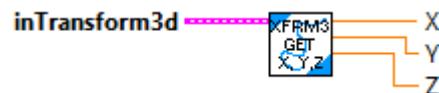
Parameters:

- IN TRANSFORM3d -- THis TRANSFORM data structure

Returns:

- TRANSLATION3d -- The translational component of the transform.
-

Transform3d_GetXYZ



Returns the X, Y, Z elements of the translation3d component of the transformation3d.

Parameters:

- IN TRANSFORM3d -- THis TRANSFORM data structure

Returns:

- X - The X element of the translational component of the transform.
 - Y - The Y element of the translational component of the transform.
 - Z - The Z element of the translational component of the transform.
-

Transform3d_Inverse



Invert the transformation3d. This is useful for undoing a transformation.

Parameters:

- IN TRANSFORM3D -- This TRANSFORM3d data structure

Results:

- OUT TRANSFORM3D -- The inverted transformation.

Transform3d_Plus



Composes two transformations.

Parameters:

- IN TRANSFORM3d -- This TRANSFORM data structure
- other Transform3d -- The transform to compose with this one.

Results:

- OUT TRANSFORM3d -- The composition of the two transformations.

Transform3d_Times



Scales the transform3d by the scalar.

Parameters:

- IN TRANSFORM3d -- This TRANSFORM data structure
- SCALAR -- The scalar value to multiply the transform by

Results:

- OUT TRANSFORM3d -- The multiplied transform data structure

Translation2d

Translation2d_Create



Constructs a Translation2d with the X and Y components equal to the provided values.

Represents a translation in 2d space. This object can be used to represent a point or a vector.

This assumes that you are using conventional mathematical axes. When the robot is placed on the origin, facing toward the X direction, moving forward increases the X, whereas moving to the left increases the Y.

Parameters:

- X - The x component of the translation.
- Y - The y component of the translation.

Returns

- TRANSLATION - The TRANSLATION data structure

Translation2d_Create_DistAng



Constructs a Translation2d with the provided distance and angle. This is essentially converting from polar coordinates to Cartesian coordinates.

Represents a translation in 2d space. This object can be used to represent a point or a vector.

This assumes that you are using conventional mathematical axes. When the robot is placed on the origin, facing toward the X direction, moving forward increases the X, whereas moving to the left increases the Y.

Parameters:

- Dist - Distance.
- Ang - Rotation representing the angle.

Returns

- TRANSLATION - The TRANSLATION data structure

Translation2d_Div



Divides the translation by a scalar and returns the new translation.

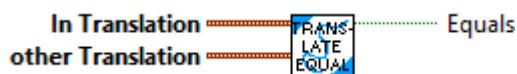
Parameters:

- In Translation - This translation data structure
- scalar - The value to divide the translation by

Result:

- Out Translation - The resulting translation

Translation2d_Equals



Checks equality between this Translation2d and another object.

Parameters:

- IN TRANSLATION - This TRANSLATION data structure
- OTHER TRANSLATION - The other TRANSLATION data structure

Returns:

- EQUALS - Value is set to TRUE if the two TRANSLATIONS are equal.
-

Translation2d_GetAngle



Returns the angle this translation forms with the positive X axis.

Parameters:

- IN TRANSLATION2d - This TRANSLATION2 data structure

Returns:

- Angle -- The angle of the translation (Radians)
-

Translation2d_GetDistance



Calculates the distance between two translations in 2d space.

This function uses the pythagorean theorem to calculate the distance. $\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Parameters:

- IN TRANSLATION - This TRANSLATION data structure
- OTHER TRANSLATION - The translation to compute the distance to.

Returns:

- Distance - The distance between the two translations.

Translation2d_GetNorm



Returns the norm, or distance from the origin to the translation.

Parameters:

- IN TRANSLATION - This translation data structure

Returns:

- Norm - The norm of the translation.

Translation2d_GetX



Returns the X component of the translation.

Parameters

- IN Translation - This Translation data structure

Returns:

- X - The x component of the translation.

Translation2d_GetXY



Returns the X and Y components of the translation.

Parameters

- IN Translation - This Translation data structure

Returns:

- X- The X component of the translation.
- Y- The Y component of the translation.

Translation2d_GetY



Returns the Y component of the translation.

Parameters

- IN Translation - This Translation data structure

Returns:

- Y- The Y component of the translation.

Translation2d_Interpolate



Interpolate between this and an End Value translation

Parameters:

- IN Translation - This translation data structure
- EndValue Translation - The translation data structure

Returns

- Interpolated Translation - The interpolated translation.
-

Translation2d_Minus



Subtracts the other translation from the other translation and returns the difference.

For example, $\text{Translation2d}\{5.0, 4.0\} - \text{Translation2d}\{1.0, 2.0\} = \text{Translation2d}\{4.0, 2.0\}$

Parameters:

- IN Translation - This translation data structure
- other Translation - The translation data structure to subtract

Returns

- Out Translation - The difference between the two translations.
-

Translation2d_NearestTo



Returns the nearest Translation2d from a list of translations. If two or more translations in the list have the same distance from this translation, return the first translation found.

Parameter:

- IN Translation -- Translation2d -- The translation to compare against the OTHER TRANSLATIONS to find the closest pose.
- OTHER Translatios -- Translation2d array -- The list of translations to search for the closest translation.

Returns:

- Nearest Translation -- Translation2d -- The translation closest to IN TRANSLATION.
 - Found -- boolean -- True if a TRANSLATION was found. This should only be false if the Other Translation array is empty.
-
-

Translation2d_Plus



Adds two translations in 2d space and returns the sum. This is similar to vector addition.

For example, $\text{Translation2d}\{1.0, 2.5\} + \text{Translation2d}\{2.0, 5.5\} = \text{Translation2d}\{3.0, 8.0\}$

Parameters:

- IN Translation - This translation data structure
- other Translation - The translation data structure to add

Returns

- Out Translation - The summation of the two translations.
-
-

Translation2d_RotateBy



Applies a rotation to the translation in 2d space.

This multiplies the translation vector by a counterclockwise rotation matrix of the given angle.

$$[x_{\text{new}}] = [\text{other}.cos, -\text{other}.sin][x]$$

$$[y_{\text{new}}] = [\text{other}.sin, \text{other}.cos][y]$$

For example, rotating a Translation2d of {2, 0} by 90 degrees will return a Translation2d of {0, 2}.

Parameters:

- In Translation - This data structure
- Other Rotation - The rotation to rotate the translation by.

Returns:

- Out Translation - The resulting Translation data structure

Translation2d_Times



Multiples the translation by a scalar and returns the new translation.

For example, $\text{Translation2d}\{2.0, 2.5\} * 2 = \text{Translation2d}\{4.0, 5.0\}$

Parameters:

- In Translation - This translation data structure
- scalar - The value to multiply the translation by

Result:

- Out Translation - The resulting translation

Translation2d_UnaryMinus



Returns the inverse of the current translation. This is equivalent to rotating by 180 degrees, flipping the point over both axes, or simply negating both components of the translation.

Parameters:

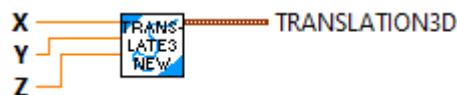
- In Translation - This translation data structure

Result:

- Out Translation - The inverse of the current translation.

Translation3d

Translation3d_Create



Constructs a Translation3d with the X, Y, and Z components equal to the provided values.

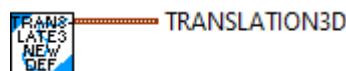
Parameters:

- X - The x component of the translation.
- Y - The y component of the translation.
- Z - The z component of the translation.

Returns

- TRANSLATION3D - The TRANSLATION3D data structure
-
-

Translation3d_Create_Default



Constructs a Translation3d with X, Y, and Z components equal to zero.

Parameters:

- None --

Returns

- TRANSLATION3D - The TRANSLATION3D data structure
-
-

Translation3d_Create_DistAng



Constructs a Translation3d with the provided distance and angle. This is essentially converting from polar coordinates to Cartesian coordinates.

Parameters:

- Dist - Distance.
- Ang - Rotation3d representing the angle.

Returns

- TRANSLATION3d - The TRANSLATION3d data structure

Translation3d_Div



Returns the translation divided by a scalar.

For example, $\text{Translation3d}(2.0, 2.5, 4.5) / 2 = \text{Translation3d}(1.0, 1.25, 2.25)$.

Parameters:

- In Translation3d - This translation3d data structure
- scalar - The value to divide the translation by

Result:

- Out Translation3d - The resulting translation

Translation3d_Equals



Checks equality between this Translation3d and another object.

Parameters:

- IN TRANSLATION3d - This TRANSLATION3d data structure
- OTHER TRANSLATION3d - The other TRANSLATION3d data structure

Returns:

- EQUALS - Value is set to TRUE if the two TRANSLATIONS are equal.
-
-

Translation3d_GetDistance



Calculates the distance between two translations in 3D space.

The distance between translations is defined as $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2+(z_2-z_1)^2}$.

Parameters:

- IN TRANSLATION3d - This TRANSLATION3d data structure
- OTHER TRANSLATION3d - The translation3d to compute the distance to.

Returns:

- Distance - The distance between the two translations.
-
-

Translation3d_GetNorm



Returns the norm, or distance from the origin to the translation3d.

Parameters:

- IN TRANSLATION3D - This translation3d data structure

Returns:

- Norm - The norm of the translation.
-
-

Translation3d_GetXYZ



Returns the X, Y and Z components of the translation.

Parameters

- IN Translation3d - This Translation3d data structure

Returns:

- X- The X component of the translation.
 - Y- The Y component of the translation.
 - Z- The Z component of the translation.
-
-

Translation3d_Interpolate



Inerpolate between this and an End Value translation3d

Parameters:

- IN Translation3d - This translation3d data structure
- EndValue Translation3d - The translation3d data structure
- T -- The fraction, between 0 and 1 to interpolate between the two translations.

Returns

- Interpolated Translation3d - The interpolated translation.

Translation3d_Minus



Returns the difference between two translations.

For example, $\text{Translation3d}(5.0, 4.0, 3.0) - \text{Translation3d}(1.0, 2.0, 3.0) = \text{Translation3d}(4.0, 2.0, 0.0)$.

Parameters:

- IN Translation3d - This translation3d data structure
- other Translation3d - The translation3d data structure to subtract

Returns

- Out Translation3d - The difference between the two translations.

Translation3d_NearestTo



Returns the nearest Translation3d from a list of translations. If two or more translations in the list have the same distance from this translation, return the first translation found.

Parameter:

- IN Translation -- Translation3d -- The translation to compare against the OTHER TRANSLATIONS to find the closest pose.

- OTHER Translatios -- Translation3d array -- The list of translations to search for the closest translation.

Returns:

- Nearest Translation -- Translation3d -- The translation closest to IN TRANSLATION.

- Found -- boolean -- True if a TRANSLATION was found. This should only be fasle if the Other Translation array is empty.

Translation3d_Plus

Adds two translations in 3d space and returns the sum.

For example, $\text{Translation3d}(1.0, 2.5, 3.5) + \text{Translation3d}(2.0, 5.5, 7.5) = \text{Translation3d}\{3.0, 8.0, 11.0\}$.

Parameters:

- IN Translation3d - This translation3d data structure

- other Translation3d - The translation3d data structure to add

Returns

- Out Translation3d - The summation of the two translations.

Translation3d_RotateBy

Applies a rotation to the translation in 3D space.

For example, rotating a Translation3d of $<2, 0, 0>$ by 90 degrees around the Z axis will return a Translation3d of $<0, 2, 0>$.

Parameters:

- in Trandlation3d -- Translation3d data structure
- Other Rotation3d -- The rotation to rotate the translation by.

Returns

- TRANSLATION3D - The TRANSLATION3D data structure
-
-

Translation3d_Times



Returns the translation multiplied by a scalar.

For example, $\text{Translation3d}(2.0, 2.5, 4.5) * 2 = \text{Translation3d}(4.0, 5.0, 9.0)$.

Parameters:

- In Translation3d - This translation3d data structure
- scalar - The value to multiply the translation by

Result:

- Out Translation 3d - The resulting translation
-
-

Translation3d_ToTranslation2d



Returns a Translation2d representing this Translation3d projected into the X-Y plane.

Parameters

- IN Translation3d - This Translation3d data structure

Returns:

- out Translation2d -- A Translation2d representing this Translation3d projected into the X-Y
-
-

Translation3d_UnaryMinus



Returns the inverse of the current translation. This is equivalent to negating all components of the translation.

Parameters:

- In Translation3d - This translation3d data structure

Result:

- Out Translation3d - The inverse of the current translation.

TrapProfConstraint

TrapProfConstraint_New



Construct constraint data cluster for a TrapezoidProfileConstraint

Inputs:

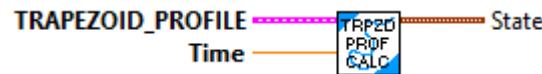
- maxVelocity -- maximum velocity
- maxAcceleration -- maximum acceleration

Outputs:

- Trapezoid_Profile_Constraint -- Updated Trapezoid_Profile_Constraint data cluster

TrapProfile

TrapProfile_Calculate



Calculate the correct position and velocity for the profile at a time t where the beginning of the profile was at time t = 0.

Inputs:

- Trapezoid_Profile -- The Trapezoid_Profile data cluster
- t -- The time since the beginning of the profile. (If the Trapezoid_Profile is freshly created every execution cycle, set t to be the time between calls.)

Outputs:

- Trapezoid_Profile -- Initialized Trapezoid_Profile data cluster
- state -- desired position and velocity values

TrapProfile_Direct



DO NOT USE. THIS IS AN INTERNAL ONLY ROUTINE.

Flip the sign of the velocity and position if the profile is inverted

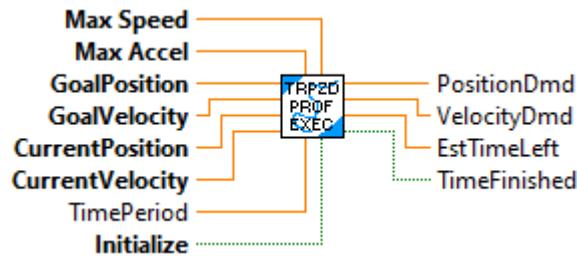
Inputs:

- State -- Trapezoid Profile State
- Direction -- Sign value (-1, 1)

Outputs:

- State -- Updated Trapezoid Profile State

TrapProfile_Execute



Convenience, single call, LabVIEW function. Creates and calculates a Trapezoid Profile. Call this routine periodically to calculate the newest output for the provided inputs.

One use of a Trapezoid Profile is to calculate a velocity SP (Setpoint) for a position movement so that the velocity SP creates a trapezoid shape to allow for smoother operation and reduced overshoot.

Inputs:

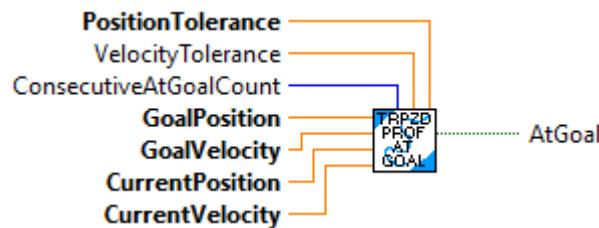
- Max Speed -- Maximum allowable speed
- Max Accel -- Maximum allowable acceleration
- Goal Position -- Desired final position
- Goal Velocity -- Desired velocity when goal position is reached. (Might often be zero.)
- Current Position -- Current position (This could be a measured value.)
- Current Velocity -- Current velocity (This could be a measured value.)
- TimePeriod -- Period between calls in seconds.
- Initialize -- If TRUE, the Goal values are reset to the inputs and the initial position and velocity are set to the current position and velocity.

Outputs:

- PositionDmd -- Current position demand (This could be used to calculate position error, or used in control.)

- VelocityDmd -- Current velocity demand. (This could be used as a setpoint to drive an actuator or motor.)
 - EstTimeLeft -- Estimated time remaining until goal is achieved.
 - TimeFinished -- The goal should have been reached because the estimated calculated time has expired.
-
-

TrapProfile_Execute_AtGoal



Convenience function to determine if the trapezoid profile goal has been reached.

Inputs:

- Position Tolerance -- Allowed position tolerance
- Velocity Tolerance -- Allowed velocity tolerance (Default 9.9E+30)
- ConsecutiveAtGoalCount -- How many times the position and velocity errors have to be within the tolerance to be considered "At Goal"
- Goal Position -- Desired final position
- Goal Velocity -- Desired velocity when goal position is reached. (Might often be zero.)
- Current Position -- Current position (This could be a measured value.)
- Current Velocity -- Current velocity (This could be a measured value.)

Outputs:

- AtGoal -- The position and velocity errors have been within defined tolerance for at least the defined consecutive scan times.
-
-

TrapProfile_IsFinished



Returns true if the profile has reached the goal.

The profile has reached the goal if the time since the profile started has exceeded the profile's total time.

Inputs:

- Trapezoid_Profile -- The Trapezoid_Profile data cluster
- t -- The time since the beginning of the profile.

Outputs:

- Finished -- True if the profile has reached the goal based solely on time being exceeded.

TrapProfile_New



A trapezoid-shaped velocity profile.

While this class can be used for a profiled movement from start to finish, the intended usage is to filter a reference's dynamics based on trapezoidal velocity constraints. To compute the reference obeying this constraint, do the following.

Initialization:

- Create a new trapezoid profile constraint and provide Max Velocity and Max Acceleration
- Create an initial previous profile reference state and provide current position (distance, angle, or other) and Velocity

Run on update:

- Create a new trapezoid profile given the constraints, unprofiled (current) reference and the previous profile reference
- Calculate providing the time since last update, the result is a new previous profile reference

where `unprofiledReference` is free to change between calls. Note that when the unprofiled reference is within the constraints, `calculate()` returns the unprofiled reference unchanged.

Otherwise, a timer can be started to provide monotonic values for `calculate()` and to determine when the profile has completed via `isFinished()`.

Construct a TrapezoidProfile data cluster and performs initial calculations to fill data cluster

Inputs:

- constraints -- The constraints on the profile, like maximum velocity.
- goal -- The desired state when the profile is complete.
- initial -- The initial state (usually the current state).

Outputs:

- Trapezoid_Profile -- Initialized Trapezoid_Profile data cluster

TrapProfile_New_DefInitial



A trapezoid-shaped velocity profile.

While this class can be used for a profiled movement from start to finish, the intended usage is to filter a reference's dynamics based on trapezoidal velocity constraints. To compute the reference obeying this constraint, do the following.

Initialization:

- Create a new trapezoid profile constraint and provide Max Velocity and Max Acceleration
- Create an initial previous profile reference state and provide current position (distance, angle, or other) and Velocity

Run on update:

- Create a new trapezoid profile given the constraints, unprofiled (current) reference and the previous profile reference
- Calculate providing the time since last update, the result is a new previous profile reference

where `unprofiledReference` is free to change between calls. Note that when the unprofiled reference is within the constraints, `calculate()` returns the unprofiled reference unchanged.

Otherwise, a timer can be started to provide monotonic values for `calculate()` and to determine when the profile has completed via `isFinished()`.

Construct a TrapezoidProfile data cluster and performs initial calculations to fill data cluster. A position and velocity of zero are used for as the initial state.

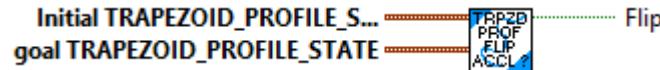
Inputs:

- constraints -- The constraints on the profile, like maximum velocity.
- goal -- The desired state when the profile is complete.

Outputs:

- Trapezoid_Profile -- Initialized Trapezoid_Profile data cluster

TrapProfile_ShouldFlipAcceleration



DO NOT USE. THIS IS AN INTERNAL ONLY ROUTINE.

Returns true if the profile inverted.

The profile is inverted if goal position is less than the initial position.

Inputs:

- initial The initial state (usually the current state).
- goal The desired state when the profile is complete.

Outputs:

- flip -- True if the profile is inverted.

TrapProfile_TimeLeftUntil



Returns the time left until a target distance in the profile is reached.

Inputs:

- Trapezoid_Profile -- The Trapezoid_Profile data cluster
- target -- The target distance.

Outputs:

- timeLeft -- Time remaining to reach target (seconds)

TrapProfile_TotalTime



Returns the total time the profile takes to reach the goal.

Inputs:

- Trapezoid_Profile -- The Trapezoid_Profile data cluster

Outputs:

- TotalTime -- Total time to reach goal (seconds)

TrapProfState

TrapProfState_Equals



Compares the values of two Trapezoid Profile States. If both are EXACTLY equal, True is returned.

Inputs:

- Trapezoid_Profile_state -- The first state to compare
- Trapezoid_Profile_state_2 -- The second state to compare

Outputs:

- Equal -- True if both states are identical.

TrapProfState_New



Creates a new Trapzoid Profile State data cluster. This could be a demand, actual, or goal (setpoint) data cluster depending on how it is used.

Inputs:

- Position -- Value for position
- Velocity -- Value for velocity.

Outputs::

- Trapezoid_Profile_State -- Initialized data cluster

Twist2d

Twist2d_Create



Constructs a Twist2d with the given values.

A change in distance along arc since the last pose update. We can use ideas from differential calculus to create new Pose2ds from a Twist2d and vice versa.

Parameters:

- dx - Change in x direction relative to robot.
- dy - Change in y direction relative to robot.
- dtheta - Change in angle relative to robot.

Results:

- TWIST - TWIST data structure

Twist2d_Equals



Checks equality between this Twist2d and another object.

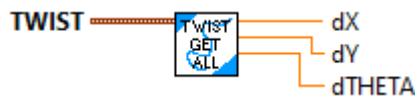
Parameters:

- this twist - This TWIST data structure
- other twist - The other TWIST data structure to compare to

Returns:

- equals - Returns TRUE if both TWISTS are equal

Twist2d_GetAll



Get the individual components of the TWIST data structure

Parameters:

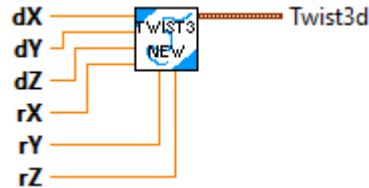
- in twist - TWIST data structure

Returns:

- dx - Linear "dx" component.
- dy - Linear "dy" component.
- dtheta - Angular "dtheta" component (radians).

Twist3d

Twist3d_Create



Constructs a Twist3d with the given values.

Parameters:

- dx - Change in x direction relative to robot.
- dy - Change in y direction relative to robot.
- dz - Change in z direction relative to robot.
- rx - Rotation vector x component.
- ry - Rotation vector y component.
- rz - Rotation vector z component.

Results:

- TWIST3D - TWIST3D data structure

Twist3d_Equals



Checks equality between this Twist3d and another Twist3d.

Parameters:

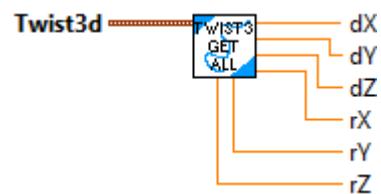
- this twist3d -- This TWIST data structure

- other twist3d -- The other TWIST data structure to compare to

Returns:

- equals - Returns TRUE if both TWISTS are equal

Twist3d_GetAll



Get the individual components of the TWIST3D data structure

Parameters:

- in twist3D -- TWIST3D data structure

Returns:

- dx - Linear "dx" component.
- dy - Linear "dy" component.
- dz - Linear "dz" component.
- rx -- Rotation vector x component (radians)
- ry -- Rotation vector y component (radians)
- rz -- Rotation vector z component (radians)

Units

Units_DegreesToRadians



Convert a value from degrees to radians

Inputs:

- Degrees - Angle (Degrees)

Outputs:

- Radians - Angle (Radians)

Units_DegreesToRotations



Convert a value from degrees to rotations

Inputs:

- Degrees - Angle (Degrees)

Outputs:

- Rotations - Angle (Rotations)

Units_FeetToMeters



Convert a value from feet to meters

Inputs:

- Feet - Distance (Feet)

Outputs:

- Meters - Distance (Meters)
-
-

Units_InchesToMeters



Convert a value from inches to meters

Inputs:

- Inches - Distance (Inches)

Outputs:

- Meters - Distance (Meters)
-
-

Units_MetersToFeet



Convert a value from meters to feet

Inputs:

- Meters - Distance (Meters)

Outputs:

- Feet - Distance (Feet)

Units_MetersToInches



Convert a value from meters to inches

Inputs:

- Meters - Distance (Meters)

Outputs:

- Inches - Distance (Inches)

Units_MillisecondsToSeconds



Convert a value from milliseconds to seconds

Inputs:

- Milliseconds

Outputs:

- Seconds

Units_RadiansPerSecondToRotationsPerMinute



Convert a value from radians/sec to revolutions per minute (RPM)

Inputs:

- RadiansPerSec - Speed (Radians / Second)

Outputs:

- RevolutionsPerMinute - Speed (RPM)
-
-

Units_RadiansToDegrees



Convert a angle value from radians to degrees

Inputs:

- Radians - Angle (Radians)

Outputs:

- Degrees - Angle (Degrees)
-
-

Units_RadiansToRotations



Convert a angle value from rotations to degrees

Inputs:

- Rotations - Angle (Rotations)

Outputs:

- Degrees - Angle (Degrees)

Units_RotationsPerMinuteToRadiansPerSecond



Convert a value from revolutions per minute (RPM) to radians/sec

Inputs:

- RevolutionsPerMinute - Speed (RPM)

Outputs:

- RadiansPerSec - Speed (Radians / Second)
-
-

Units_RotationsToDegrees



Convert Rotations to Degrees

Units_RotationsToRadians



Convert a value from rotations to radians

Inputs:

- Rotations - Angle (rotations)

Outputs:

- Radians - Angle (Radians)
-
-

Units_SecondsToMilliseconds



Convert a value from seconds to milliseconds

Inputs:

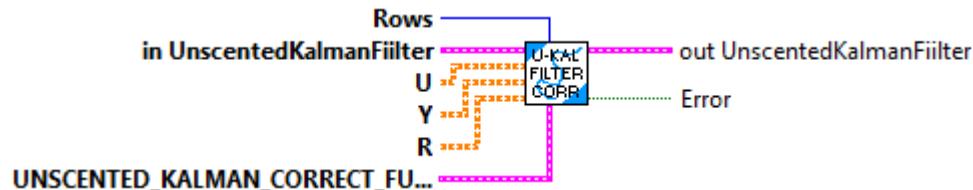
- Seconds

Outputs:

- Milliseconds

UnscentedKalmanFilter

UnscentedKalmanFilter_Correct



Correct the state estimate \hat{x} using the measurements in y .

This is useful for when the measurements available during a timestep's `Correct()` call vary.

The $h(x, u)$ passed to the constructor is used if one is not provided (the two-argument version of this function).

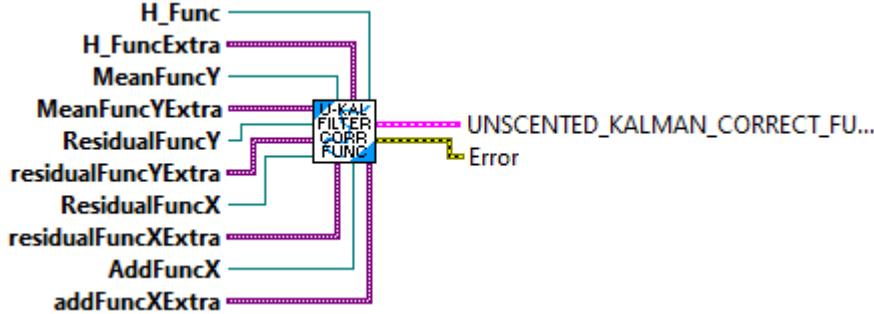
Inputs:

- rows -- Number of rows in y .
- inUnscentedKalmanFilter -- filter data cluster
- u -- Same control input used in the predict step.
- y -- Measurement vector.
- R -- Measurement noise covariance matrix (continuous-time).
- FuncGroup -- Packed data cluster containing the callback functions used by this routine.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- Error -- If TRUE, an error occurred.

UnscentedKalmanFilter_Correct_FuncGroup



Creates a new packed function group cluster to pass to the Correct subVI.

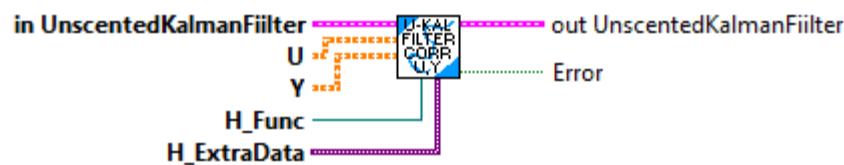
Inputs:

- `H_Func` -- A vector-valued function of x and u that returns the measurement vector.
- `H_ExtraData` -- Variant containing extra data used by the `H` function. The contents of the variant are specific to the `H` function.
- `meanFuncY` -- A strict function reference that computes the mean of 2 States + 1 measurement vectors using a given set of weights.
- `meanFuncYExtra` -- Variant containing extra data used by the `meanY` function. The contents of the variant are specific to the `meanY` function.
- `residualFuncY` -- A strict function reference that computes the residual of two measurement vectors (i.e. it subtracts them.)
- `residualFuncYExtra` -- Variant containing extra data used by the `residualY` function. The contents of the variant are specific to the `residualY` function.
- `residualFuncX` -- A strict function reference that computes the residual of two state vectors (i.e. it subtracts them.)
- `residualFuncXExtra` -- Variant containing extra data used by the `residualX` function. The contents of the variant are specific to the `residualX` function.
- `addFuncX` -- A strict function reference that adds two state vectors.
- `addFuncXExtra` -- Variant containing extra data used by the `addY` function. The contents of the variant are specific to the `addY` function.

Outputs:

- `UnscentedKalmanFilterCorrectFuncGroup` -- Cluster containing the functions and extra data.
- `Error` -- If TRUE, an error occurred.

UnscentedKalmanFilter_Correct_OnlyUY



Correct the state estimate \hat{x} using the measurements in y .

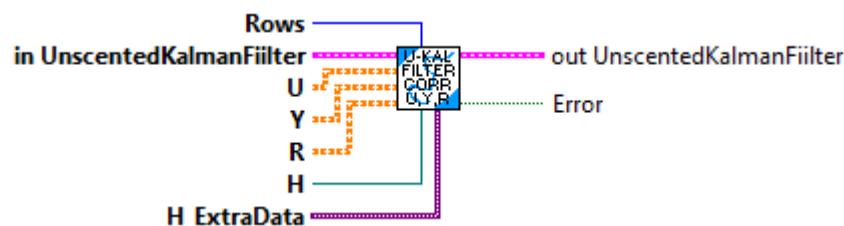
Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- u -- Same control input used in the predict step.
- y -- Measurement vector.
- H_Func -- A vector-valued function of x and u that returns the measurement vector.
- H_ExtraData -- Variant containing extra data used by the H function. The contents of the variant are specific to the H function.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- Error -- If TRUE, an error occurred.

UnscentedKalmanFilter_Correct_OnlyUYR



Correct the state estimate \hat{x} using the measurements in y .

This is useful for when the measurements available during a timestep's `Correct()` call vary.

The `h(x, u)` passed to the constructor is used if one is not provided (the two-argument version of this function).

Inputs:

- `inUnscentedKalmanFilter` -- filter data cluster
- `rows` -- Number of rows in `y`.
- `u` -- Same control input used in the predict step.
- `y` -- Measurement vector.
- `R` -- Measurement noise covariance matrix (continuous-time).
- `H_Func` -- A vector-valued function of `x` and `u` that returns the measurement vector.
- `H_ExtraData` -- A variant containing extra data used by the `H` function. The contents are specific to each different `H` function.

Outputs:

- `outUnscentedKalmanFilter` -- updated filter data cluster
- `Error` -- If TRUE, an error occurred.

UnscentedKalmanFilter_GetP



Returns the error covariance matrix `P`.

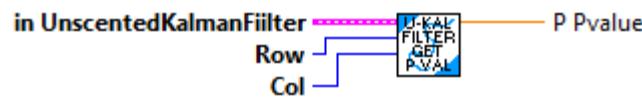
Inputs:

- `inUnscentedKalmanFilter` -- filter data cluster

Outputs:

- `P` -- the error covariance matrix `P`.

UnscentedKalmanFilter_GetP_Single



Returns an element of the error covariance matrix P.

Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- row -- Row of P.
- col -- Column of P.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- P_Value -- the value of the error covariance matrix P at (i, j).

UnscentedKalmanFilter_GetXHat



Returns the state estimate x-hat.

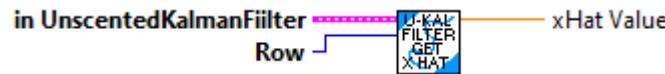
Inputs:

- inUnscentedKalmanFilter -- filter data cluster

Outputs:

- xHat -- the state estimate x-hat.

UnscentedKalmanFilter_GetXHat_Single



Returns an element of the state estimate x-hat.

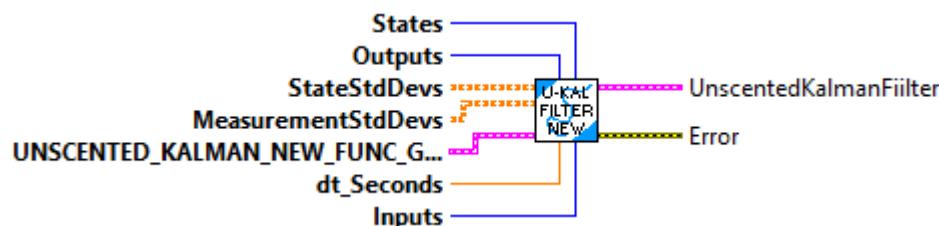
Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- row -- Row of x-hat.

Outputs:

- xHat_Value -- the value of the state estimate x-hat at i.

UnscentedKalmanFilter_New



A Kalman filter combines predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

The Unscented Kalman filter is similar to the Kalman filter, except that it propagates carefully chosen points called sigma points through the non-linear model to obtain an estimate of the true covariance (as opposed to a linearized version of it). This means that the UKF works with nonlinear systems.

Constructs an unscented Kalman filter with custom mean, residual, and addition functions. Using custom functions for arithmetic can be useful if you have angles in the state or measurements, because they allow you to correctly account for the modular nature of angle arithmetic.

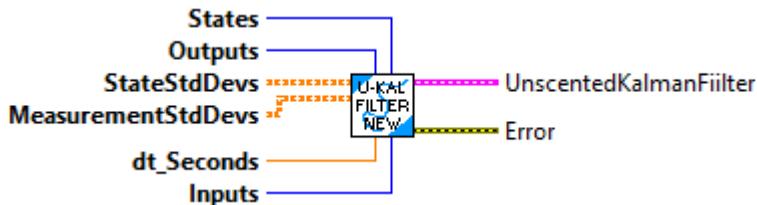
Inputs:

- states -- The number of states.
- outputs -- The number of outputs.
- sstateStdDevs -- Standard deviations of model states.
- measurementStdDevs -- Standard deviations of measurements.
- nominalDtSeconds Nominal discretization timestep.
- Inputs -- The number of inputs.
- FunctionGroup -- A packed cluster containing the callback functions.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- Error -- If TRUE, and error occurred.

UnscentedKalmanFilter_New_Default



A Kalman filter combines predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

The Unscented Kalman filter is similar to the Kalman filter, except that it propagates carefully chosen points called sigma points through the non-linear model to obtain an estimate of the true covariance (as opposed to a linearized version of it). This means that the UKF works with nonlinear systems.

Constructs an unscented Kalman filter with custom mean, residual, and addition functions. Using custom functions for arithmetic can be useful if you have angles in the state or measurements, because they allow you to correctly account for the modular nature of angle arithmetic.

Constructs an Unscented Kalman Filter.

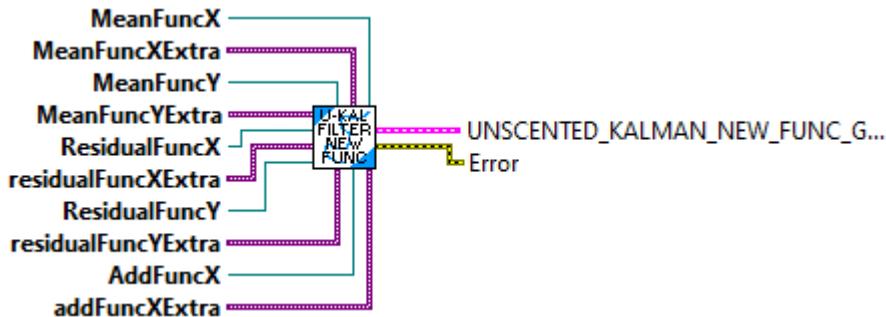
Inputs:

- states -- The number of states.
- outputs -- The number of outputs.
- stateStdDevs -- Standard deviations of model states.
- measurementStdDevs -- Standard deviations of measurements.
- nominalDtSeconds -- Nominal discretization timestep.
- Inputs -- the number of inputs.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- Error -- If TRUE, an error occurred.

UnscentedKalmanFilter_New_FuncGroup



Creates a new packed function group cluster .

Inputs:

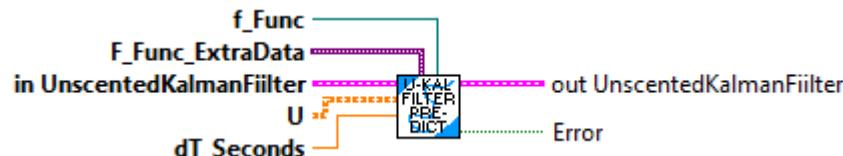
- meanFuncX -- A strict function reference that computes the mean of 2States + 1 state vectors using a given set of weights.
- meanFuncXExtra -- A variant containing extra data for the meanX function
- meanFuncY -- A strict function reference that computes the mean of 2 States + 1 measurement vectors using a given set of weights.

- meanFuncYExtra -- A variant containing extra data for the meanY function
- residualFuncX -- A strict function reference that computes the residual of two state vectors (i.e. it subtracts them.)
- residualFuncXExtra -- A variant containing extra data for the residualX function
- residualFuncY -- A strict function reference that computes the residual of two measurement vectors (i.e. it subtracts them.)
- residualFuncYExtra -- A variant containing extra data for the residualY function
- addFuncX -- A strict function reference that adds two state vectors.
- addFuncXExtra -- A variant containing extra data for the addX function

Outputs:

- UnscentedKalmanFilterFuncGroup -- Cluster containing the functions and extra data.
- Error -- If TRUE, an error occurred.

UnscentedKalmanFilter_Predict



Project the model into the future with a new control input u.

Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- u -- New control input from controller.
- dtSeconds -- Timestep for prediction.
- F_Func -- strict function reference. A vector-valued function of x and u that returns the derivative of the state vector.
- F_FuncExtra -- extra data, if any, for F_Func

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
 - error -- If TRUE, an error occurred.
-

UnscentedKalmanFilter_Reset



Resets the observer.

Inputs:

- inUnscentedKalmanFilter -- filter data cluster

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
-

UnscentedKalmanFilter_SetP



Sets the entire error covariance matrix P.

Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- newP -- The new value of P to use.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster

- sizeCoerced -- If TRUE, an error occured

UnscentedKalmanFilter_SetXHat



Set initial state estimate x-hat.

Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- xHat -- The state estimate x-hat.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- sizeCoerced -- If TRUE, an error occured

UnscentedKalmanFilter_SetXHat_Single



Set an element of the initial state estimate x-hat.

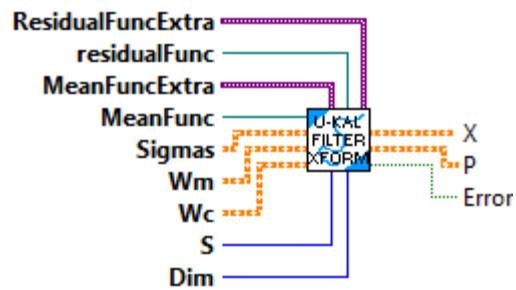
Inputs:

- inUnscentedKalmanFilter -- filter data cluster
- row -- Row of x-hat.
- value -- Value for element of x-hat.

Outputs:

- outUnscentedKalmanFilter -- updated filter data cluster
- error -- If TRUE, an error occurred.

UnscentedKalmanFilter_Transform



Internal routine used by the Unscented Kalman filter routines.

Inputs:

- ResidualFuncExtra -- Extra data passed to the residual function
- Residual Func -- Strict reference to residual function
- MeanFuncExtra -- Extra data passed to the mean function
- Mean Func -- Strict reference to mean function
- Sigmas --
- Wm --
- Wc --
- S --
- Dim --

Outputs:

- X --
- P --
- Error -- If TRUE, an error occurred

Util

Util_ApproxEquals



Determines if two values are approximately equal. It determines if they are within a particular tolerance.

Inputs:

- Value 1 -- First value to be compared.
- Value 2 -- Second value to be compared
- Tolerance -- Tolerance to use when comparing the values (Default - 1.0E-9)

Output:

- ApproxEqual -- Boolean indicating that the values are approximately equal.
-
-

Util_Array_PoseWCurv2d_to_XY



This is a convenience function to convert the X, Y portions of a PoseWithCurvature array into arrays of X,Y for charting.

Util_CalcDist



Utility function to calculate distance (hypotenuse) result = (x ^2 + y ^2) ^ 0.5

Parameters:

- X

- Y

Result

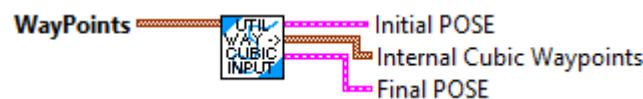
- Distance

Util_DispWaypoint_Eng_To_SI



Utility function to convert utility waypoints from Inches, degrees to meters, radians. All the normal trajectory functions expect the units to be in meters, radians (unless otherwise specified).

Util_DispWaypoint_To_CubicInput



Utility function to convert utility waypoints to the input data structures needed to calculate cubic splines.

Util_DispWaypoint_To_QuinticInput



Utility function to convert utility waypoints to the input data structures needed to calculate cubic splines.

Util_DispWeightedWaypoint_Eng_To_WeightedWaypoint_SI



Utility function to convert utility (displayable) weighted waypoints from Inches, degrees to meters, radians. All the normal trajectory functions expect the units to be in meters, radians (unless otherwise specified).

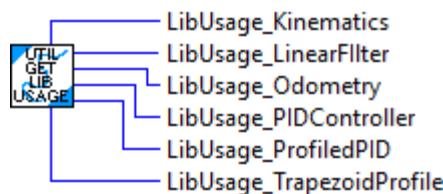
This then converted from the utility weighted waypoints, which are good for putting on front panels, to normal weighted waypoints that are used by the trajectory generation functions.

Util_DispWeightedWaypoint_To_WeightedWayPoint



Utility function to convert UTIL_WeightedWayPoints to WeightedWayPoints

Util_GetLibUsage



Returns how many instances of some of the systems in this library are used. The count increments the first time a "new" routine is called for a particular pre-allocated clone or inline invocation of a system.

Util_GetLibraryVersion



Return the library Trajectory Library version string.

Copyright (c) 2020 James A. Simpson

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the authors nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL AUTHORS OR CONTRIBUTORS BE LIABLE FOR

ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES

(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

=====

=

Copyrights and Licenses for Third Party Software Distributed with FRC LV TRAJECTORY Library

=====

=

The FRC LV Trajectory Library software contains code written by and derived from third parties. The copyrights, license, and restrictions which apply to each piece

of software is included later in this file and/or inside of the individual applicable source files.

The disclaimer of warranty in the FRC LV Trajectory Library license above applies to all code in FRC LV Trajectory Library, and nothing in any of the other licenses gives permission to use the names of FIRST nor the names of the FRC LV Trajectory Library contributors to endorse or promote products derived from this software.

The following pieces of software have additional or alternate copyrights, licenses, and/or restrictions:

Program	Locations
-----	-----
WPILIB	LabVIEW code derived from WPILIB
RoboRIO Libraries	ni-libraries
Pathfinder	Pathfinder Library
JSON for Modern C++	LabVIEW code derived from WPILIB routines
Team 254 Library	LabVIEW code derived from WPILIB SplineParameterizer.java, TrajectoryParameterizer.java

WPILIB License

Copyright (c) 2009-2018 FIRST
All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- * Neither the name of the FIRST nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY FIRST AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY NONINFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL FIRST OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

=====

=

PATHFINDER License

=====

=

The MIT License (MIT)

Copyright (c) 2016 Jaci R

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

=====

=
JSON for Modern C++ License

=====

|||_||_|_||| JSON for Modern C++
||_|_|||_|||_||| version 2.1.1
_____|_____|_____|_||_||| https://github.com/nlohmann/json

Licensed under the MIT License .

Copyright (c) 2013-2017 Niels Lohmann .

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

=====

Team 254 Library

=====

MIT License

Copyright (c) 2018 Team 254

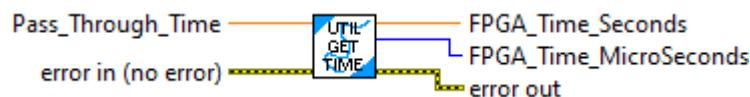
Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal

in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Util_GetTime



This subVI returns the continuously running elapsed time in seconds.

On the RoboRIO this reads the high resolution FPGA time. The resolution, not accuracy, of the FPGA timer is 1 microsecond. The normal FPGA clock register (32 bits) rolls over approximately every 71 minutes. To significantly extend the rollover time,, this routine reads both the high and low FPGA clock registers. High/low register turnover read sequencing is dealt with by reading the high register twice, once before and once after the low register. If the values are different, the read is repeated to assure consistency between the high and low values. On a PC the standard operating system timer with 1 millisecond resolution is used.

Notes:

-
- On the RoboRIO this routine takes approximately 2.1 microseconds to execute. When both words of the FPGA clock have to be read the routine takes approximately 4.2 microseconds. This occurs once every 71 minutes when the routine determines that the low word has rolled over. The WPI read FPGA clock routine takes approximately 2.1 microseconds. The difference can be attributed to this routine reading to extra words from the FPGA. Each FPGA read takes approximately 1 microsecond.
 - Set a conditional compilation variable TRAJLIB_USEOSTIME and set it to a value of 1 to force this routine to read the RoboRIO operating system time instead of the FPGA time. This will have 1 millisecond resolution. This routine executes faster, approx 1 microsecond.
 - On the RoboRIO there is some drift between the FPGA and operating system clocks. The drift isn't consistent, but appears to be approximately 20 milliseconds / hour. This isn't enough to be of concern.

Input parameters:

- PassThroughTime -- double float -- If wired and the value is > 0 then this time is passed through and the FPGA time is not read. Allows calling functions to use a single time instant if desired.
- Input Error -- Input error cluster

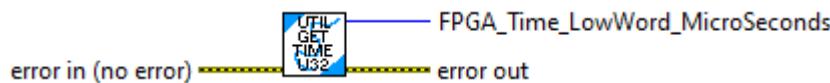
Returns:

- FPGA_Time_Seconds -- double float -- Elapsed time in seconds.
- FPGA_Time_Microseconds -- Uint64 -- Elapsed time in microseconds (This can be used as an input to the Network Table routines to synchronize times.)
- Output Error -- Output error cluster. An error will be returned if there are problems reading the FPGA or of the FPGA clock high register changes twice while reading the value.

Custom Errors:

- 8020 -- The upper FPGA time register changed during both attempts to read the FPGA time values. The returned time may not be valid.

Util_GetTime_U32



This subVI returns the low word of the FPGA continuously running elapsed time in microseconds.

This is an internal routine and should NOT be called directly by the end user.

On the RoboRIO this reads the high resolution FPGA time. The resolution, not accuracy, of the FPGA timer is 1 microsecond. The normal FPGA clock register (32 bits) rolls over approximately every 71 minutes. To significantly extend the rollover time,, this routine reads both the high and low FPGA clock registers. High/low register turnover read sequencing is dealt with by reading the high register twice, once before and once after the low register. If the values are different, the read is repeated to assure consistency between the high and low values. On a PC the standard operating system timer with 1 millisecond resolution is used.

NOTES:

- Set a conditional compilation variable TRAJLIB_USEOSTIME and set it to a value of 1 to force this routine to read the RoboRIO operating system time instead of the FPGA time. This will have 1 millisecond resolution. This routine executes faster, approx 1 microsecond.

- On the RoboRIO there is some drift between the FPGA and operating system clocks. The drift isn't consistent, but appears to be approximately 20 milliseconds / hour. This isn't enough to be of concern.

Input parameters:

- Error In -- Error Data Cluster --

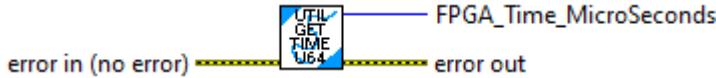
Returns:

- FPGA_Time_LowWord_Microseconds -- UInt32 -- Low word of continuously running FPGA clock.
- Output Error -- Output error cluster. An error will be returned if there are problems reading the FPGA or of the FPGA clock high register changes twice while reading the value.

Custom Errors:

- 8020 -- The upper FPGA time register changed during both attempts to read the FPGA time values. The returned time may not be valid.

Util_GetTime_U64



This subVI returns both words of the FPGA continuously running elapsed time in microseconds.

This is an internal routine and should NOT be called directly by the end user.

On the RoboRIO this reads the high resolution FPGA time. The resolution, not accuracy, of the FPGA timer is 1 microsecond. The normal FPGA clock register (32 bits) rolls over approximately every 71 minutes. To significantly extend the rollover time,, this routine reads both the high and low FPGA clock registers. High/low register turnover read sequencing is dealt with by reading the high register twice, once before and once after the low register. If the values are different, the read is repeated to assure consistency between the high and low values. On a PC the standard operating system timer with 1 millisecond resolution is used.

NOTES:

- Set a conditional compilation variable TRAJLIB_USEOSTIME and set it to a value of 1 to force this routine to read the RoboRIO operating system time instead of the FPGA time. This will have 1 millisecond resolution. This routine executes faster, approx 1 microsecond.
- On the RoboRIO there is some drift between the FPGA and operating system clocks. The drift isn't consistent, but appears to be approximately 20 milliseconds / hour. This isn't enough to be of concern.

Input parameters:

- Error In -- Error Data Cluster --

Returns:

- FPGA_Time_Microseconds -- UInt64 -- Full 64 bit value of the continuously counting FPGA clock.
- Output Error -- Output error cluster. An error will be returned if there are problems reading the FPGA or of the FPGA clock high register changes twice while reading the value.

Custom Errors:

- 8020 -- The upper FPGA time register changed during both attempts to read the FPGA time values. The returned time may not be valid.

Util_TrajState_to_DiffDrive_WheelPos



Utility function to create the position of individual differential drive wheels from a trajectory state. This is used to plot the wheel positions. Inputs are Meters and Radians

Util_TrajectoryState_Meters_To_Inches



Utility function to convert a trajectory state from meters, radians to inches and degrees. This is for DISPLAY ONLY. Do not SAMPLE or perform other operations on this data!!!

Util_Trajectory_Absolute_To_Relative



Utility function to convert an absolute trajectory to a robot relative trajectory. A robot relative trajectory's initial pose is at X,Y,Ang = 0,0,0

Util_Trajectory_ReadFile



Create a trajectory from a CSV file. This can be used on a PC or the RoboRIO. Normally the CSV file is created as output from one of the trajectory utility programs. The file could also be created manually or by a custom written program.

Parameters:

- FileName -- Name of the CSV file to read. See file name notes for additional information.

- Error In -- Input error cluster (optional)

Returns:

- outTrajectory - Trajectory data structure cluster
- Error out - returned error cluster

Notes on use:

- This routine writes informational messages to the console and to the driver station log.

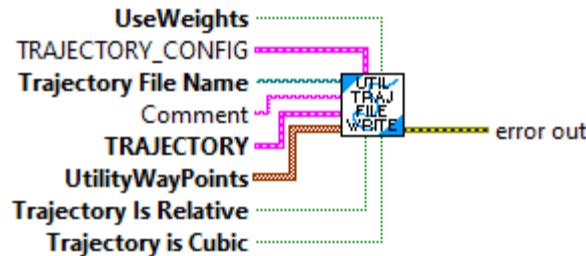
Notes on file naming:

- The file name must include the extension. ".csv" is not automatically appended to the name.
- The file name can be a simple file or an absolute path. If a simple file name is used the default path on the RoboRIO is: "home:\lvuser\natinst\LabVIEW Data". On a Windows PC the default path is the LabVIEW default directory. Normally this is: %HOMEDRIVE%&%HOMEPATH%\Documents\LabView Data".
- Filenames on the RoboRIO, which runs Linux, are case sensitive.

Notes on file contents:

- Blank lines are ignored.
- Lines that begin with either #, !, or ' in the first character are considered comments and are ignored.
- Other lines are interpreted as comma separated data as follows:
 - Trajectory time -- seconds
 - Expected velocity -- meters/sec
 - Expected acceleration -- meters/sec²
 - Expected X position -- meters
 - Expected Y position -- meters
 - Expected heading (not necessarily robot orientation) -- radians
 - Expected curvature (rotational portion of velocity) -- radians/meter
 - Optional comment string. This is ignored when reading file

Util_Trajectory_WriteFile



Create a CSV file from a Trajectory data structure. This can be used on a PC or the RoboRIO.

Parameters:

- Trajectory - Data structure containing trajectory
- TrajectoryConfig - Trajectory configuration
- Trajectory File Name - File name to write to. Existing files are overwritten.
- Comment - Comment string to include in file comments.
- Waypoints - waypoints to include in file comments
- TrajectoryIsRelative - Relative if TRUE otherwise absolute
- TrajectoryIsCubic - Cubic if TRUE otherwise Quintic

Returns:

- Error out - returned error cluster

Returns:

- outTrajectory - Trajectory data structure
- Error out - returned error cluster

Util_Trajectory_WriteFile_Config



Internal routine to write the Trajectory Configuration data to a file.

Parameters:

- ByteStream In - File stream
- TrajectoryConfig - Data structure containing trajectory configuration

Returns:

- ByteStream Out - File stream
- Error out - returned error cluster

Returns:

- outTrajectory - Trajectory data structure
 - Error out - returned error cluster
-
-

Util_Trajectory_WriteFile_OneState



Internal subVI used by Util_Trajectory_WriteFile (and others). This writes one trajectory state to a file.

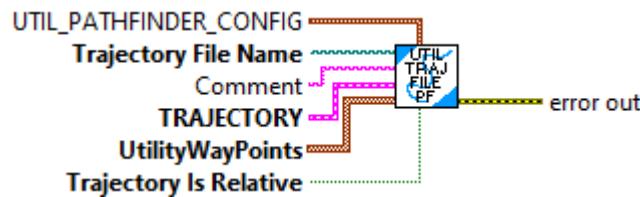
Parameters:

- Byte stream in - file stream
- comment - comment for this line
- TrajectoryState - The state to write

Returns:

- Byte Stream Out - file stream
-
-

Util_Trajectory_WriteFile_PathFinder



Create a CSV file from a Trajectory data structure. This can be used on a PC or the RoboRIO. This is a customized version for Pathfinder created trajectories.

Parameters:

- Trajectory - Data structure containing trajectory
- PathfinderConfig - Data structure with pathfinder configuration.
- Trajectory File Name - File name to write to. Existing files are overwritten.
- Comment - Comment string to include in file comments.
- Waypoints - waypoints to include in file comments
- TrajectoryIsRelative - Relative if TRUE otherwise absolute.

Returns:

- Error out - returned error cluster

Util_Trajectory_WriteFile_PathFinderConfig



Internal routine to write the Pathfinder configuration to a file. This can be used on a PC or the RoboRIO.

Parameters:

- ByteStreamIn - file stream
- PathfinderConfig - cluster with configuration information.

Returns:

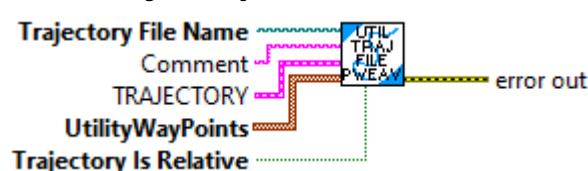
- ByteStreamOut - File stream.

- Error out - returned error cluster

Returns:

- outTrajectory - Trajectory data structure
- Error out - returned error cluster

Util_Trajectory_WriteFile_Pathweaver



Create a CSV file from a Trajectory data structure. This is a special version for Pathweaver created trajectories. This can be used on a PC or the RoboRIO.

Parameters:

- Trajectory - Data structure containing trajectory
- Trajectory File Name - File name to write to. Existing files are overwritten.
- Comment - Comment string to include in file comments.
- Waypoints - waypoints to include in file comments (future)
- TrajectoryIsRelative - Relative if TRUE otherwise absolute.

Returns:

- Error out - returned error cluster

Util_Trajectory_WriteFile_States



Write trajectory states to a file. This is an internal routine

Parameters:

- ByteStreamIn - File stream
- Trajectory - Data structure containing trajectory

Returns:

- ByteStreamOut - File stream
 - Error out - returned error cluster
-

Util_Trajectory_WriteFile_WayPoints



Internal routine to write trajectory waypoints to a file.

Parameters:

- ByteStreamIn - File stream
- Waypoints - waypoints to include in file comments (future)

Returns:

- ByteStreamOut - File stream
 - Error out - returned error cluster
-

Util_Trajectory_to_XY



This is a convenience function to convert the X, Y portions of trajectory states into arrays of X,Y for charting.

Inputs:

- trajectory -- trajectory data structure

Outputs:

- outArrayXy -- Cluster of X and Y arrays useful for charting.

VecBuilder

VecBuilder_1x1Fill



Build a 1×1 Matrix (vector) from an array of input values.

Inputs:

-- R0_C0 -- Input value

Outputs:

-- Output Matrix -- Resulting 1×1 matrix (vector)

VecBuilder_2x1Fill



Build a 2×1 Matrix (vector) from an array of input values.

Inputs:

-- R0 -- Input value 1

-- R1 -- Input value 2

Outputs:

-- Output Matrix -- Resulting 2×1 matrix (vector)

VecBuilder_3x1Fill



Build a 3 x 1 Matrix (vector) from an array of input values.

Inputs:

- R0 -- Input value 1
- R1 -- Input value 2
- R2 -- Input value 3

Outputs:

- Output Matrix -- Resulting 3 x 1 matrix (vector)
-

VecBuilder_4x1Fill



Build a 4 x 1 Matrix (vector) from an array of input values.

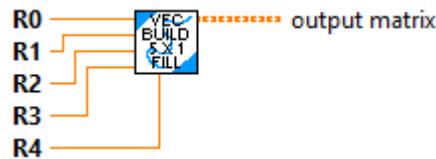
Inputs:

- R0 -- Input value 1
- R1 -- Input value 2
- R2 -- Input value 3
- R3 -- Input value 4

Outputs:

- Output Matrix -- Resulting 4 x 1 matrix (vector)
-

VecBuilder_5x1Fill



Build a 5 x 1 Matrix (vector) from an array of input values.

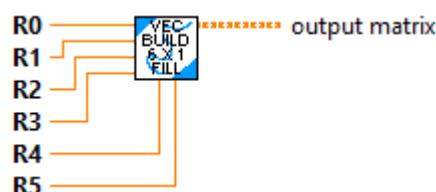
Inputs:

- R0 -- Input value 1
- R1 -- Input value 2
- R2 -- Input value 3
- R3 -- Input value 4
- R4 -- Input value 5

Outputs:

- Output Matrix -- Resulting 5 x 1 matrix (vector)
-

VecBuilder_6x1Fill



Build a 6 x 1 Matrix (vector) from an array of input values.

Inputs:

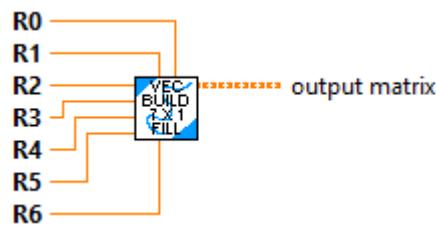
- R0 -- Input value 1
- R1 -- Input value 2
- R2 -- Input value 3
- R3 -- Input value 4

- R4 -- Input value 5
- R5 -- Input value 6

Outputs:

- Output Matrix -- Resulting 6 x 1 matrix (vector)
-
-

VecBuilder_7x1Fill



Build a 7 x 1 Matrix (vector) from an array of input values.

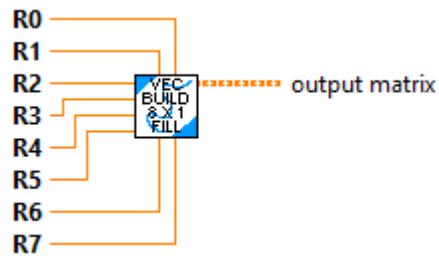
Inputs:

- R0 -- Input value 1
- R1 -- Input value 2
- R2 -- Input value 3
- R3 -- Input value 4
- R4 -- Input value 5
- R5 -- Input value 6
- R6 -- Input value 7

Outputs:

- Output Matrix -- Resulting 7 x 1 matrix (vector)
-
-

VecBuilder_8x1Fill



Build a 8 x 1 Matrix (vector) from an array of input values.

Inputs:

- R0 -- Input value 1
- R1 -- Input value 2
- R2 -- Input value 3
- R3 -- Input value 4
- R4 -- Input value 5
- R5 -- Input value 6
- R6 -- Input value 7
- R7 -- Input value 8

Outputs:

- Output Matrix -- Resulting 8 x 1 matrix (vector)

VecBuilder_ArrayBy1Fill



Build an "n" x 1 matrix (vector) from an array of provided values.

Inputs:

- R Array -- Array of values used to build the vector

Outputs:

- Output_Vector -- "n" x 1 matrix (vector) created from the input values. The size of the this matrix is based on the number of items in the input array.

Vector

Vector_Dot



Calculate the dot product of two vectors

Inputs:

- InputVector 1 -- Input vector 1
- InputVector 2 -- Input vector 2. This vector must have at least as many rows as vector 1.

Outputs:

- Dot Product -- Calculated vector dot product.

Vector_Norm



Calculate the norm of the vector (square root of the sum of the squares of all values).

Inputs:

- InputVector -- Input vector

Outputs:

- Norm -- Calculated norm

Type Definitions

TypeDef

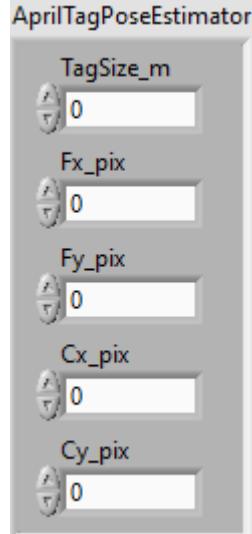
TypeDef-APRIL_TAG_POSE_ESTIMATOR



Pose estimators for AprilTag tags.

Contains:

```
-- tagSize -- double -- tag size, in meters  
-- fx -- double -- camera horizontal focal length, in pixels  
-- fy -- double -- camera vertical focal length, in pixels  
-- cx -- double -- camera horizontal focal center, in pixels  
-- cy -- double -- camera vertical focal center, in pixels
```



TypeDef-ARM_FF

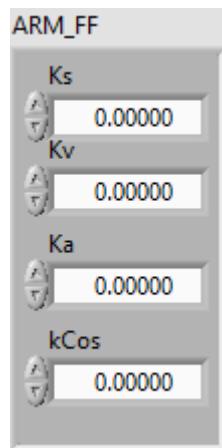


Data cluster for a set of functions. that compute feedforward outputs for a simple arm (modeled as a motor acting against the force of gravity on a beam suspended at an angle.

Units of the gain values will dictate units of the computed feedforward.

Elements"

- Ks - The static gain.
- Kv - The velocity gain.
- Ka - The acceleration gain.
- KCos - The gravity gain.

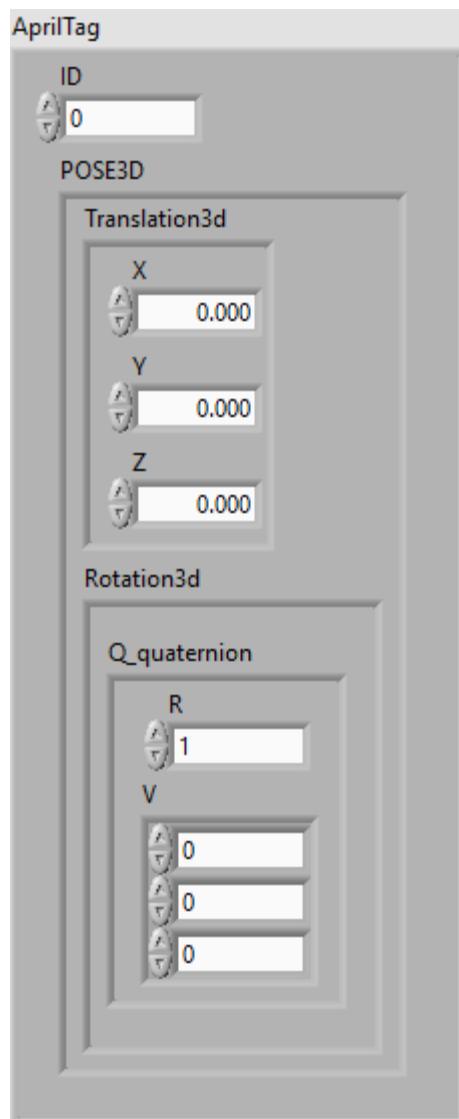


TypeDef-AprilTag



Data cluster to store information for an April Tag vision target. The stored data includes:

- ID -- Integer -- The April Tag ID value.
- Pose3d -- Pose3d -- The position of this April Tag.



TypeDef-AprilTagFieldLayout



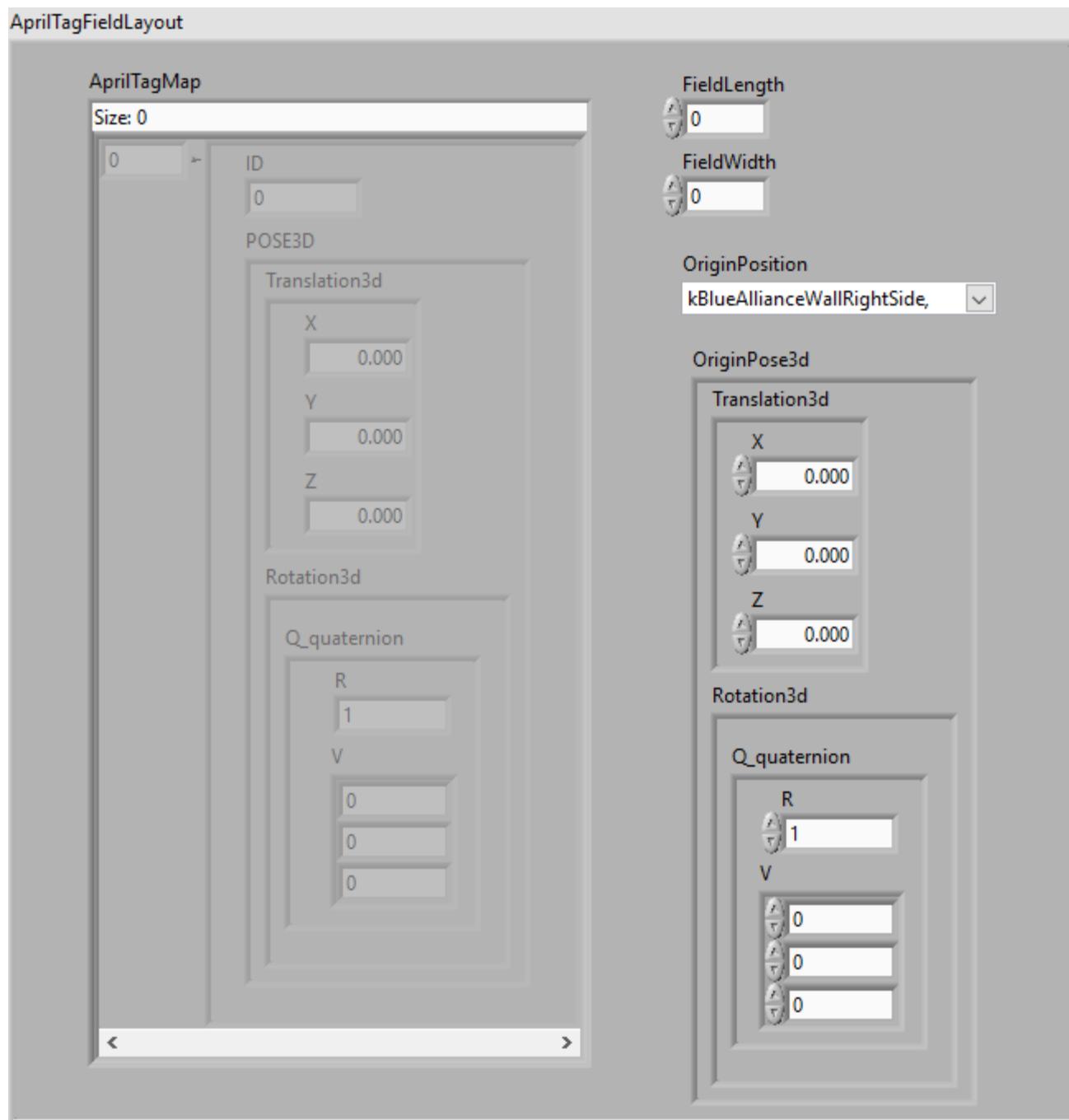
Data cluster and set of functions for representing a layout of AprilTags on a field. (Currently reading the field definition from a file is not yet supported.)

The JSON format contains two top-level objects, "tags" and "field". The "tags" object is a list of all AprilTags contained within a layout. Each AprilTag serializes to a JSON object containing an ID and a Pose3d. The "field" object is a descriptor of the size of the field in meters with "width" and "length" values. This is to account for arbitrary field sizes when transforming the poses.

Pose3ds in the JSON are measured using the normal FRC coordinate system, NWU with the origin at the bottom-right corner of the blue alliance wall. {@link #setOrigin(OriginPosition)} can be used to change the poses returned from AprilTagFieldLayout_getTagPose(int) to be from the perspective of a specific alliance.

The data cluster contains:

- AprilTagMap -- Map -- A map indexable by April Tag ID to obtain specific AprilTag data clusters.
- Field Length -- double -- Field length (meters)
- Field Width -- double -- Field width (meters)
- OriginPosition -- enum -- Origin is red or blue alliance. (normal origin is blue alliance).
- OriginPose3d -- Pose3d -- Position of the origin.



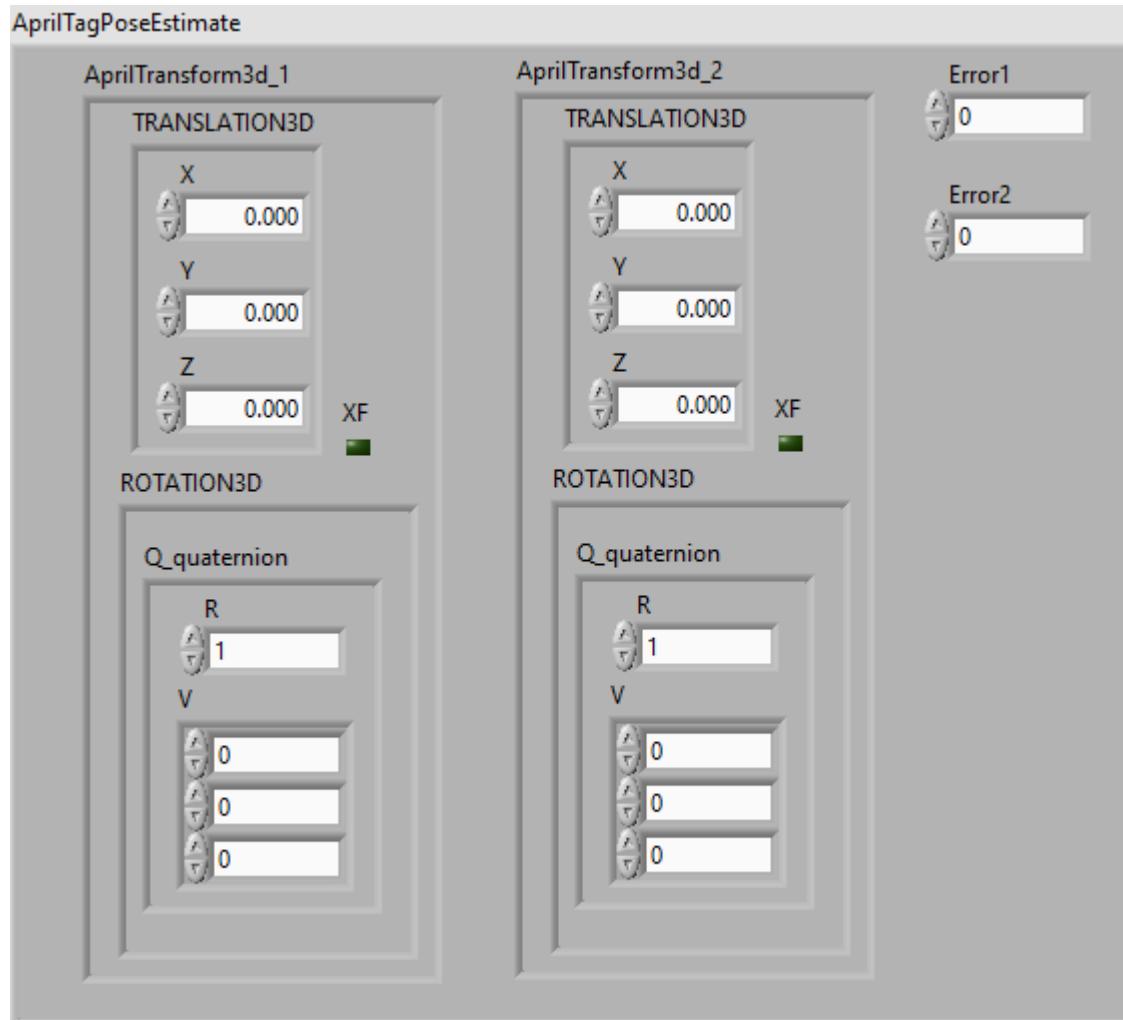
TypeDef-AprilTagPoseEstimate



pair of AprilTag pose estimates.

Contains:

- AprilTransform3d_1 -- Transform3d -- First Transform to AprilTag
- AprilTransform3d_2 -- Transform3d -- Second Transform to AprilTag
- Error1 -- Double -- error of first pose
- Error2 -- Double -- error of second pose



TypeDef-BANG_BANG

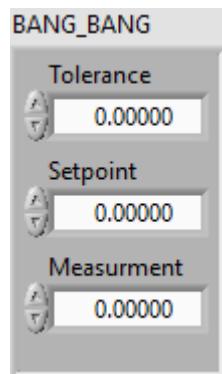


Implements a bang-bang controller, which outputs either 0 or 1 depending on whether the measurement is less than the setpoint. This maximally-aggressive control approach works very well for velocity control of high-inertia mechanisms, and poorly on most other things.

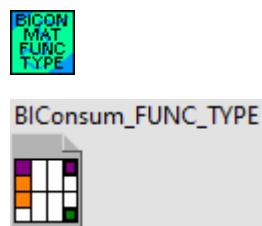
Note that this is an **asymmetric** bang-bang controller - it will not exert any control effort in the reverse direction (e.g. it won't try to slow down an over-speeding shooter wheel). This asymmetry is **extremely important.** Bang-bang control is extremely simple, but also potentially hazardous. Always ensure that your motor controllers are set to "coast" before attempting to control them with a bang-bang controller.

Data stored in this cluster

- Tolerance
- Setpoint
- Measurement



TypeDef-BiCon_Matrix_FUNC_TYPE



TypeDef-CALLBACK_FUNC_TYPE



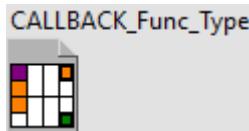
This strictly typed function reference it used by various functions to declare the call-backs needed for operation. It has the following parameters:

Inputs:

- ExtraData -- Variant containing extra data used by this function
- Mat X -- Input Matrix
- Mat U -- Input Matrix

Outputs:

- Mat Output -- Output Matrix
- Error -- Boolean, If TRUE indicates an error has occurred.

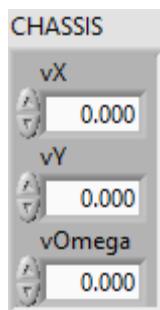


TypeDef-CHASSIS_SPEEDS



Represents the speed of a robot chassis. Although this struct contains similar members compared to a Twist2d, they do NOT represent the same thing. Whereas a Twist2d represents a change in pose w.r.t to the robot frame of reference, this ChassisSpeeds struct represents a velocity w.r.t to the robot frame of reference.

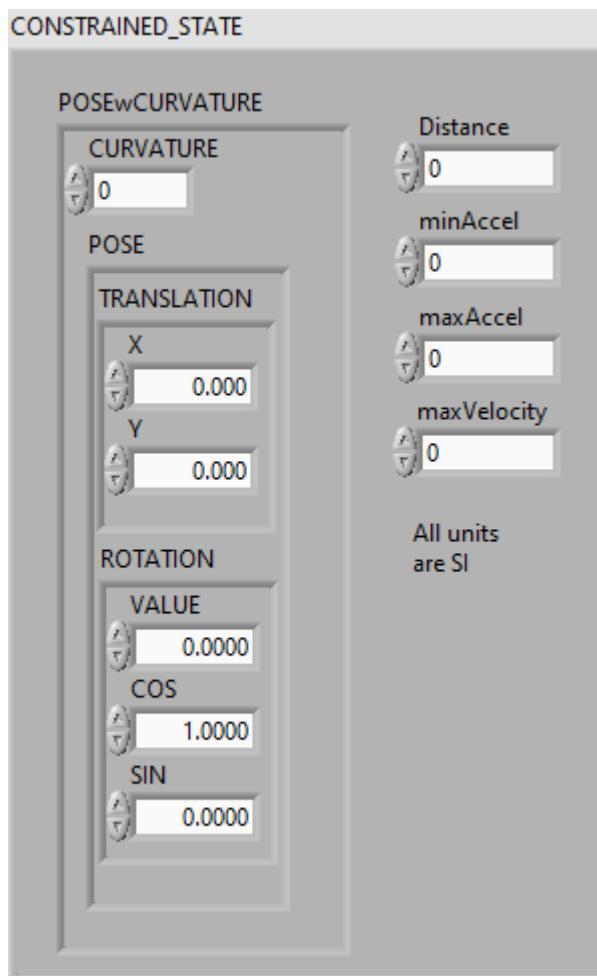
A strictly non-holonomic drivetrain, such as a differential drive, should never have a dy component because it can never move sideways. Holonomic drivetrains such as swerve and mecanum will often have all three components.



TypeDef-CONSTRAINED_STATE



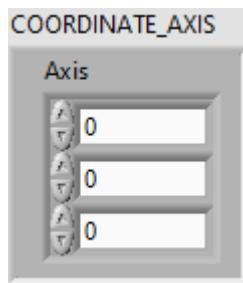
Internal data structure used by TrajectoryParam_timeParam.



TypeDef-COORDINATE_AXIS



A data cluster representing a coordinate system axis within the NWU coordinate system.



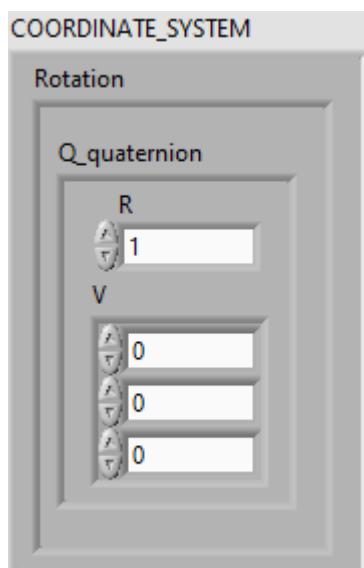
TypeDef-COORDINATE_SYSTEM



A helper data cluster that converts Pose3d objects between different standard coordinate frames.

Contains:

-- Rotation 3d -- Rotation from this coordinate system to NWU coordinate system



TypeDef-DCMOTOR



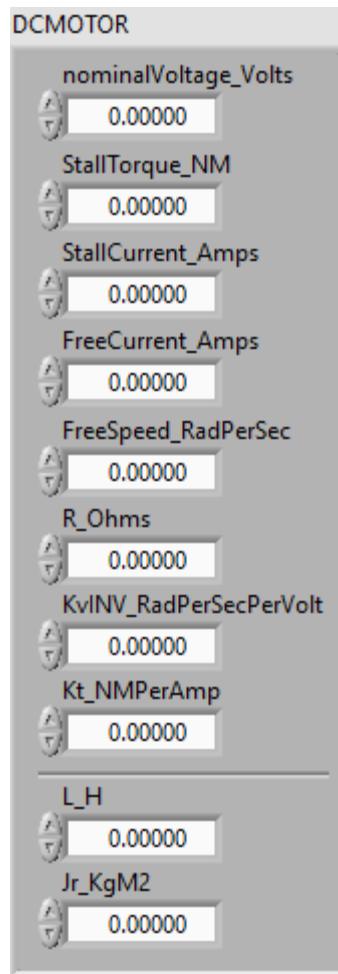
Holds the constants for a DC motor.

Data

- NominalVoltage_Volts -- Voltage at which the motor constants were measured. (Volts)
- StallTorque_NewtonMeters --
- StallCurrent_Amps -- Current draw when stalled.
- FreeCurrent_Amps -- Current draw under no load.
- FreeSpeed_RadPerSec -- Angular velocity under no load.
- R_Ohms -- Winding resistance (Ohms)
- Kv_RadPerSecPerVolt -- Motor speed constant. Note that some documentation inverts this value.
- Kt_NewtonMeterPerAmp -- Motor torque constant.

The following pieces of data may not exist for all motors.

- L_H -- Winding inductance (Henrys) --
- Jr_KgM2 -- Motor moment of inertia --



TypeDef-DCMOTOR_SIM



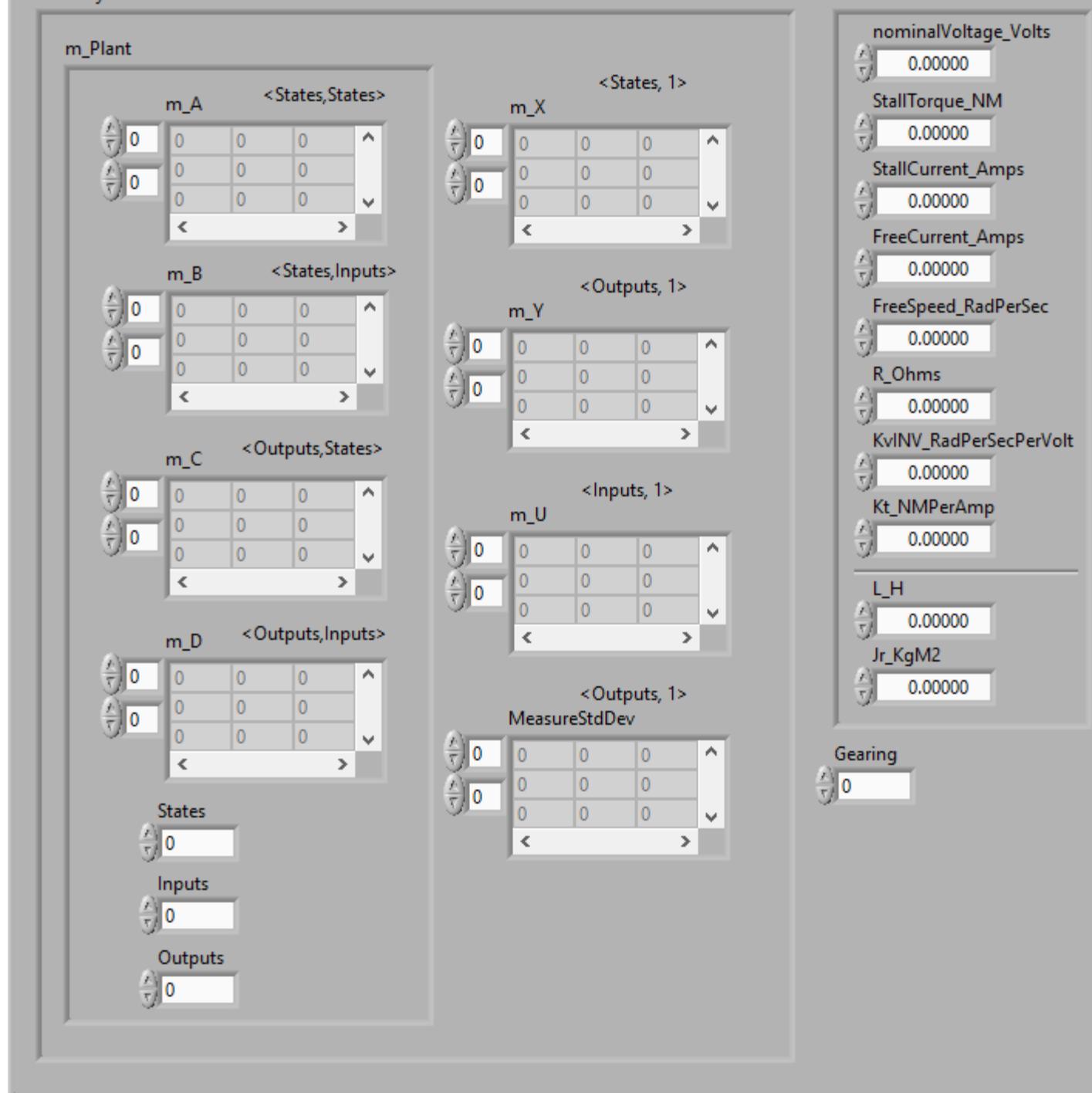
Represents a simulated DC Motor mechanism.

Cluster contains:

- Gearbox -- DCMotor cluster for the flywheel.
- Gearing -- Double, gearing between the motors and the output.
- LinearSystemSim -- Cluster for the simulation.

DCMOTOR_SIM

LinearSystemSim



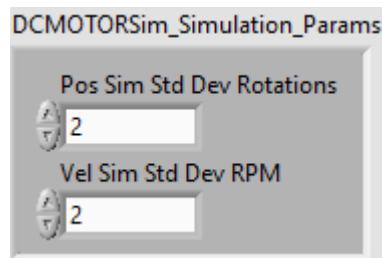
TypeDef-DCMOTOR_SIM_SIMULATION_PARAMS



Cluster containing packed simulation parameters used by the DC Motor Simulation Execute routine.

Contains:

- Pos Sim Std Dev -- double -- Standard deviation for the motor position (Rotations Default: 2)
- Vel Sim Std Dev -- double -- Standard deviation for the motor velocity (RPM, Default: 2)



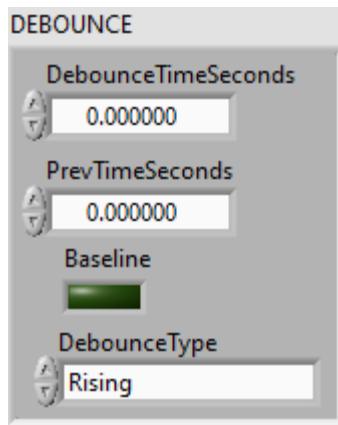
TypeDef-DEBOUNCER



A simple debounce filter for boolean streams. Requires that the boolean change value from baseline for a specified period of time before the filtered value changes.

This cluster contains:

- DebounceTimeSeconds -- Double
- PrevTimeSeconds -- Double
- DebounceType -- Debounce Type enum
- Baseline -- Boolean



TypeDef-DIFF_DRIVE_ACCEL_LIMIT

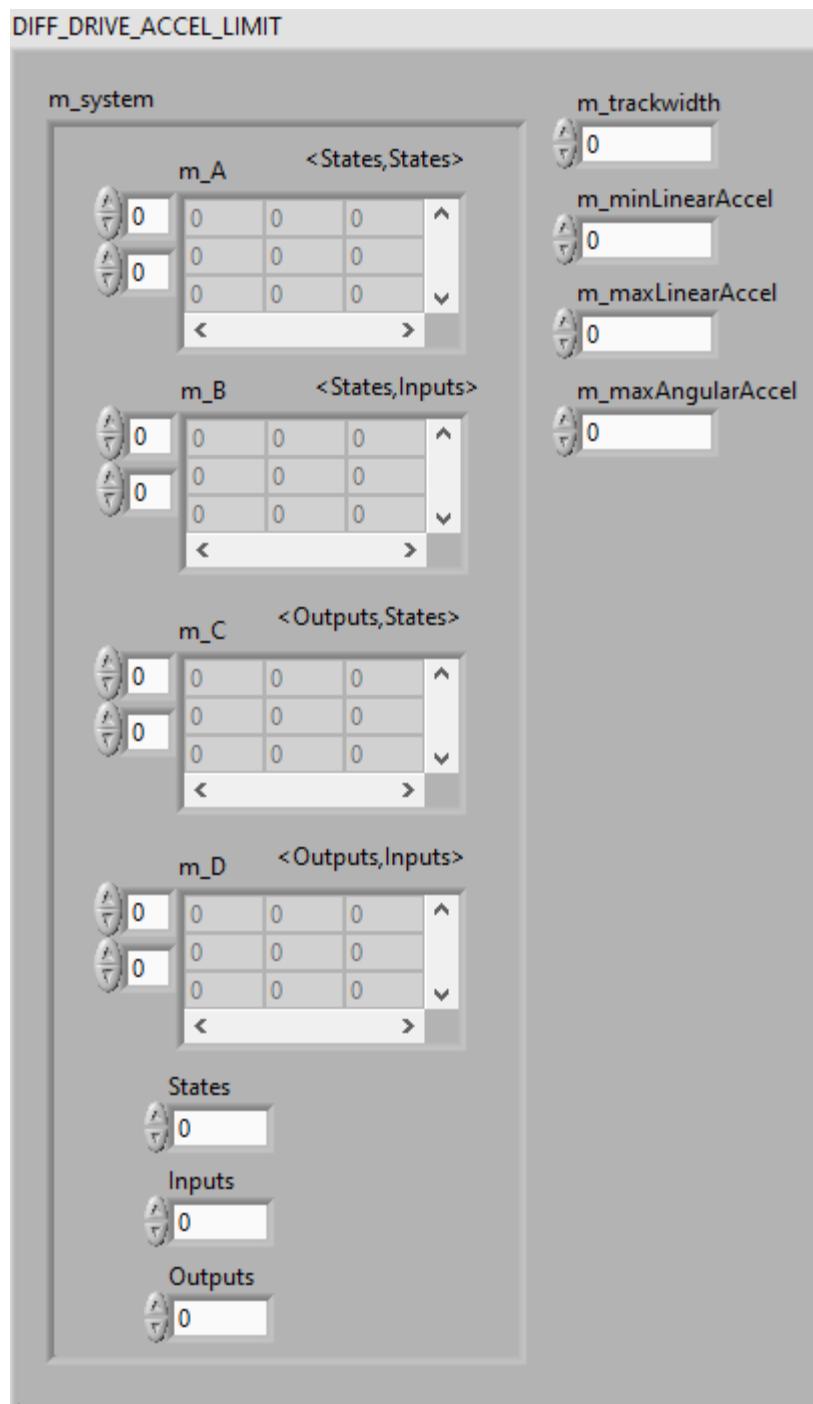


Filters the provided voltages to limit a differential drive's linear and angular acceleration.

The differential drive model can be created via the functions in LinearSystemId.

The data values are:

- System -- Linear System
- trackWidth -- trackwidth (meters)
- minLinearAccel -- (meters/sec²)
- maxLinearAccel -- (meters/sec²)
- maxAngularAccel -- (radians/sec²)

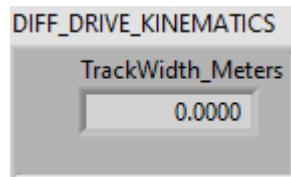


TypeDef-DIFF_DRIVE_KINEMATICS



Helper class that converts a chassis velocity (dx and dtheta components) to left and right wheel velocities for a differential drive.

Inverse kinematics converts a desired chassis speed into left and right velocity components whereas forward kinematics converts left and right component velocities into a linear and angular chassis speed.



TypeDef-DIFF_DRIVE_ODOM2



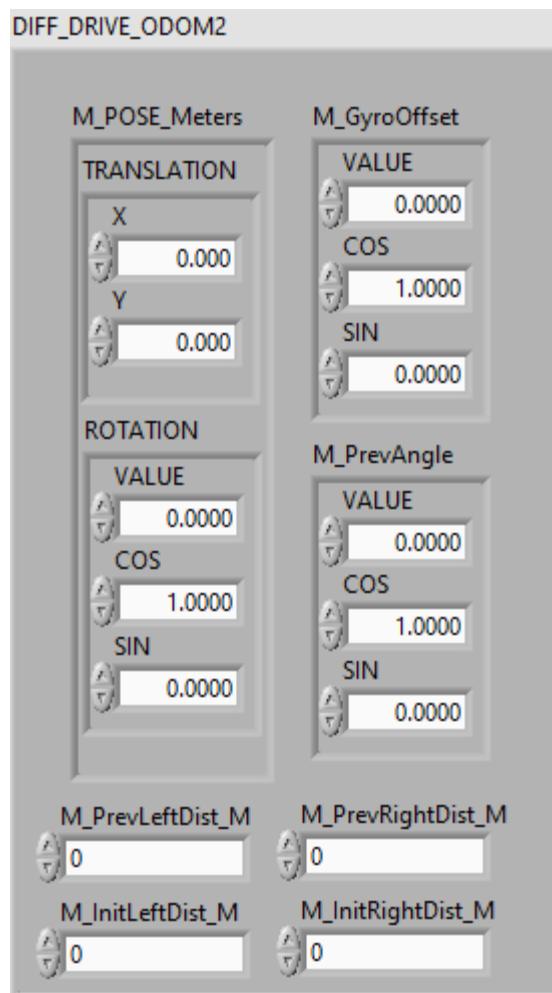
Cluster and subVI for differential drive odometry. Odometry allows you to track the robot's position on the field over the course of a match using readings from 2 encoders and a gyroscope.

Teams can use odometry during the autonomous period for complex tasks like path following. Furthermore, odometry can be used for latency compensation when using computer-vision systems.

The gyro and encoders do NOT have to be reset before using this.

This cluster contains:

- M_Pose_Meters -- Pose2d -- Current robot position (meters)
- M_GyroOffset -- Rotation2d -- Initial gyro value
- M_PrevLeftDist_M -- Double -- Previous reading of left wheel encoder distance (meters)
- M_PrevRightDist_M -- Double -- Previous reading of right wheel encoder distance (meters)
- M_InitLeftDist_M -- Double -- Initial reading of left wheel encoder distance (meters)
- M_InitRightDist_M -- Double -- Initial reading of right wheel encoder distance (meters)



TypeDef-DIFF_DRIVE_POSE_EST



This set of functions is deprecated. Suggest using the new Diff Drive Pose Est 2 instead.

This set of subVI wraps an UnscentedKalmanFilter to fuse latency-compensated vision measurements with differential drive encoder measurements. It will correct for noisy vision measurements and encoder drift. It is intended to be an easy drop-in for DifferentialDriveOdometry; in fact, if you never call DifferentialDrivePoseEstimator_addVisionMeasurement and only call DifferentialDrivePoseEstimator_update then this will behave exactly the same as DifferentialDriveOdometry.

DifferentialDrivePoseEstimator_update should be called every robot loop (if your robot loops are faster than the default then you should change specify the update time when creating this data cluster).

DifferentialDrivePoseEstimator_addVisionMeasurement can be called as infrequently as you want; if you never call it then this class will behave exactly like regular encoder odometry.

To promote stability of the pose estimate and make it robust to bad vision data, we recommend only adding vision measurements that are already within one meter or so of the current pose estimate.

Our state-space system is:

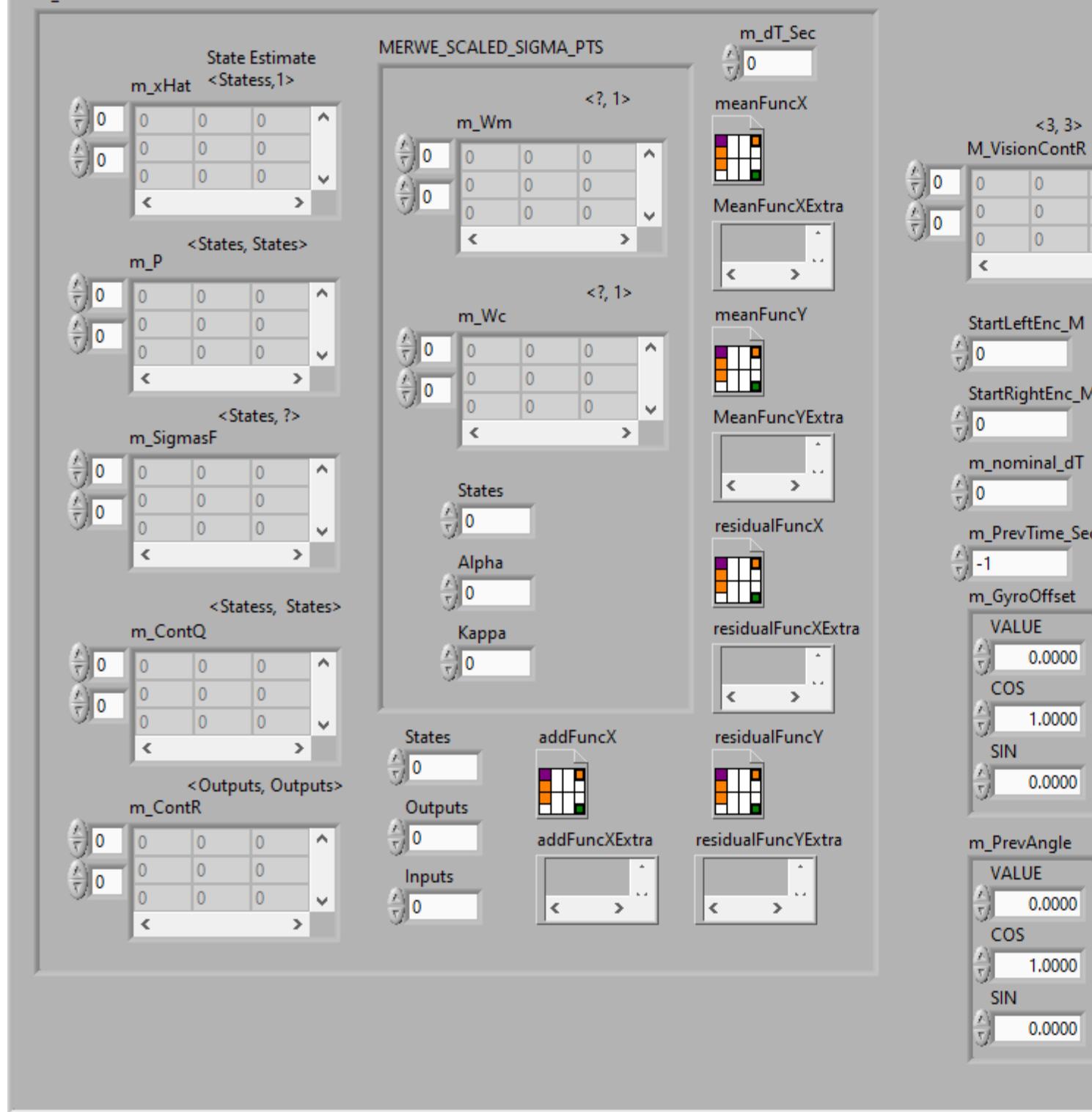
$x = [[x, y, \theta, dist_l, dist_r]]?$ in the field coordinate system ($dist_*$ are wheel distances.)

$u = [[vx, vy, \omega]]?$ (robot-relative velocities) -- NB: using velocities make things considerably easier, because it means that teams don't have to worry about getting an accurate model. Basically, we suspect that it's easier for teams to get good encoder data than it is for them to perform system identification well enough to get a good model.

$y = [[x, y, \theta]]?$ from vision, or $y = [[dist_l, dist_r, \theta]]$ from encoders and gyro.

DIFF_DRIVE_POSE_EST

m_Observer



TypeDef-DIFF_DRIVE_POSE_EST2



This set of functions wraps Differential Drive Odometry to fuse latency-compensated vision measurements with differential drive encoder measurements. It is intended to be a drop-in replacement for DiffDrvOdom2; in fact, if you never call DiffDrvPoseEst2_AddVisionMeasurement and only call DiffDrvPoseEst2_Update then this will behave exactly the same as DiffDrvOdom2.

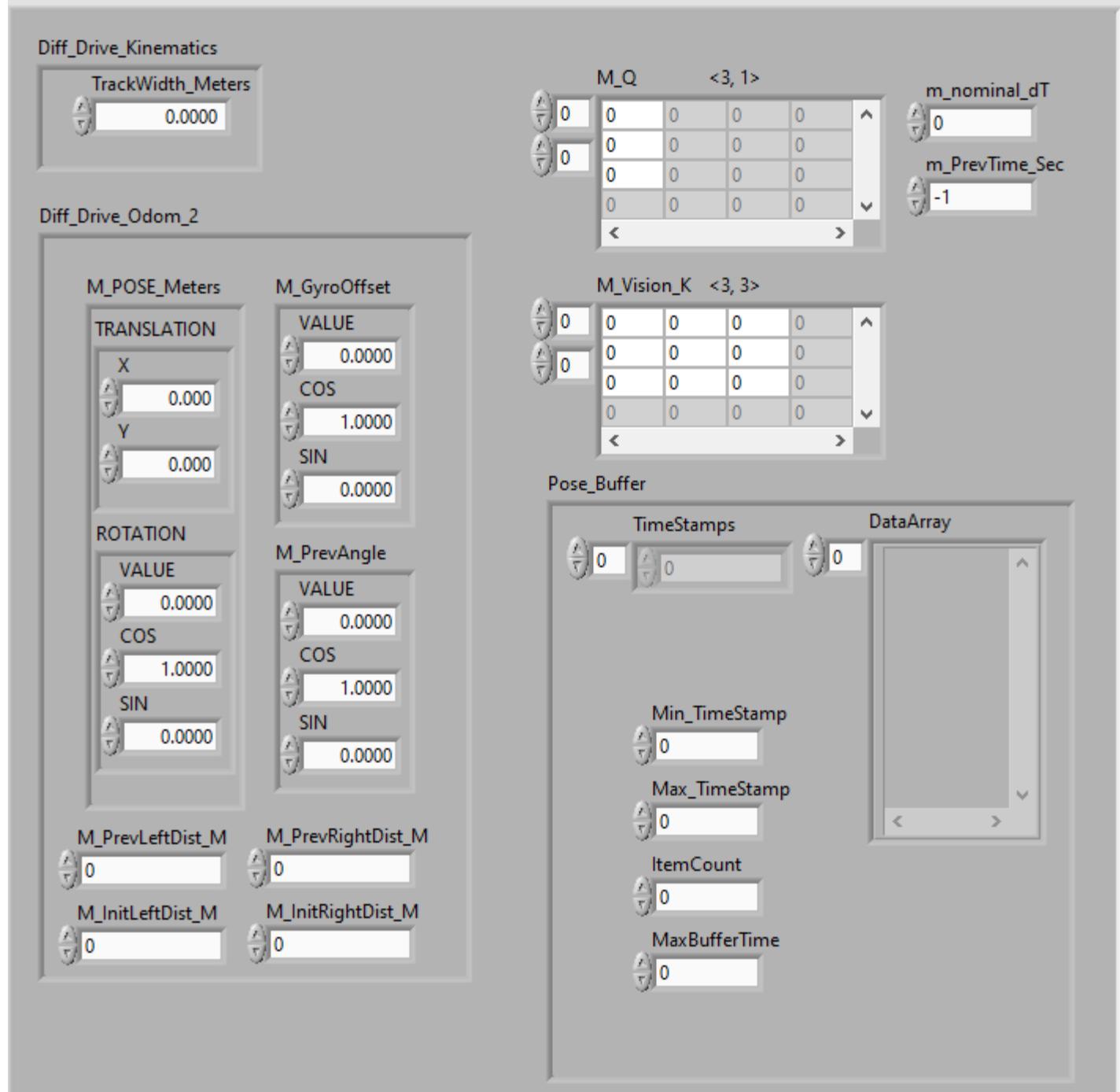
DiffDrvPoseEst2_Update should be called every robot loop.

DiffDrvPoseEst2_AddVisionMeasurement can be called as infrequently as you want. If you never call it then this set of VI will behave exactly like regular encoder odometry.

This data cluster contains:

- Diff_Drive_Kinematics -- Diff_Drive_Kinematics -- Kinematics data cluster that defines this drive.
- Diff_Pose_Odom2 -- Diff_Drive_Odom2 -- Odometry data cluster for this drive.
- M_Q -- <3,1> Matrix -- Matrix that holds standard deviations for the robot pose X, Y, theta (meters, meters, radians)
- M_Vision_K -- <3,3> Matrix -- Matrix that holds the gain matrix for closed loop continuous Kalman filter.
- Pose_Buffer -- TimeInterpBufferVariant -- Holds last 1.5 seconds of position measurements.

DIFF_DRIVE_POSE_EST2



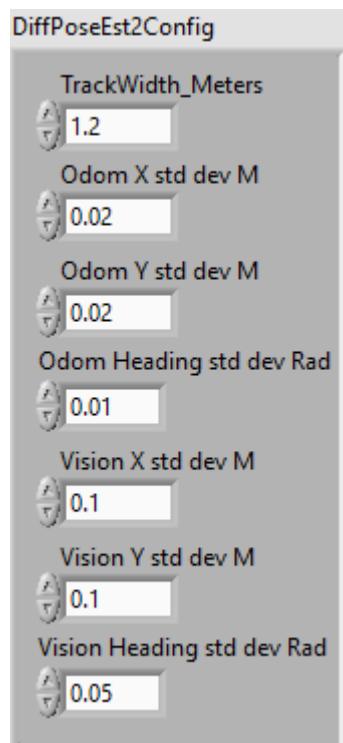
TypeDef-DIFF_DRIVE_POSE_EST2_CONFIG



Clustere containing configuration data required by the DiffDrivePoseEst2_Execute function.

Contains:

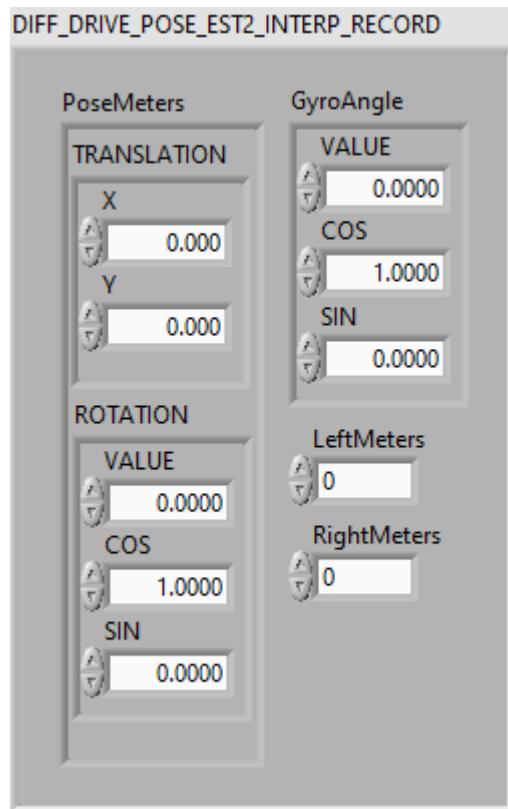
- TrackWidth_Meters -- double -- Effective track width (meters)
- Odom X Std Dev M -- double -- Odometry X position standard deviation (Default: 0.02) (meters)
- Odom Y Std Dev M -- double -- Odometry Y position standard deviation (Default: 0.02) (meters)
- Odom Heading Std Dev M -- double -- Odometry Heading (gyro) position standard deviation (Default: 0.01) (radians)
- Vision X Std Dev M -- double -- Vision X position standard deviation (Default: 0.1) (meters)
- Vision Y Std Dev M -- double -- Vision Y position standard deviation (Default: 0.1) (meters)
- Vision Heading Std Dev M -- double -- Vision Heading (gyro) position standard deviation (Default: 0.05) (radians)



TypeDef-DIFF_DRIVE_POSE_EST2_INTERP_RECORD

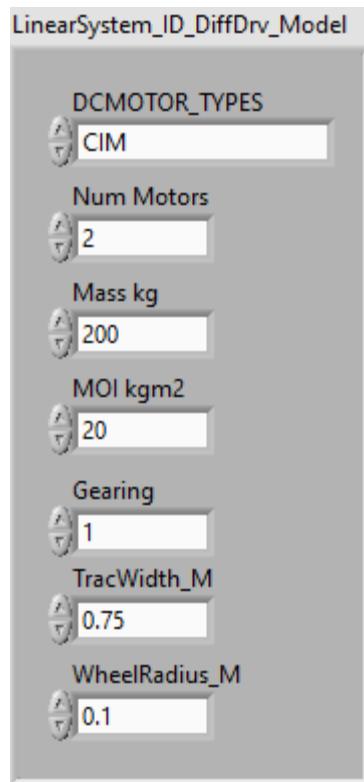


Differential Drive Pose Estimate 2 -- Interporatable Record.

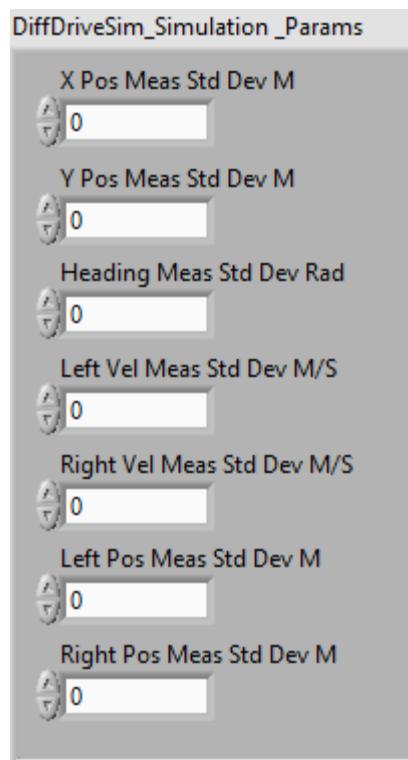


TypeDef-DIFF_DRIVE_SIM_MODEL_PARAMS





TypeDef-DIFF_DRIVE_SIM_SIMULATION_PARAMS



TypeDef-DIFF_DRIVE_TRAIN_SIM



This data cluster holds the values for the Differential Drive Train functions that simulate the state of the drivetrain. In simulationPeriodic, users should first set inputs from motors with setInputs(double, double)}, call update(double)} to update the simulation, and set estimated encoder and gyro positions, as well as estimated odometry pose.

Our state-space system is:

$$x = [[x, y, \theta, v_l, v_r, d_l, d_r]]$$

in the field coordinate system (dist_* are wheel distances.)

$$u = [[voltage_l, voltage_r]]?$$

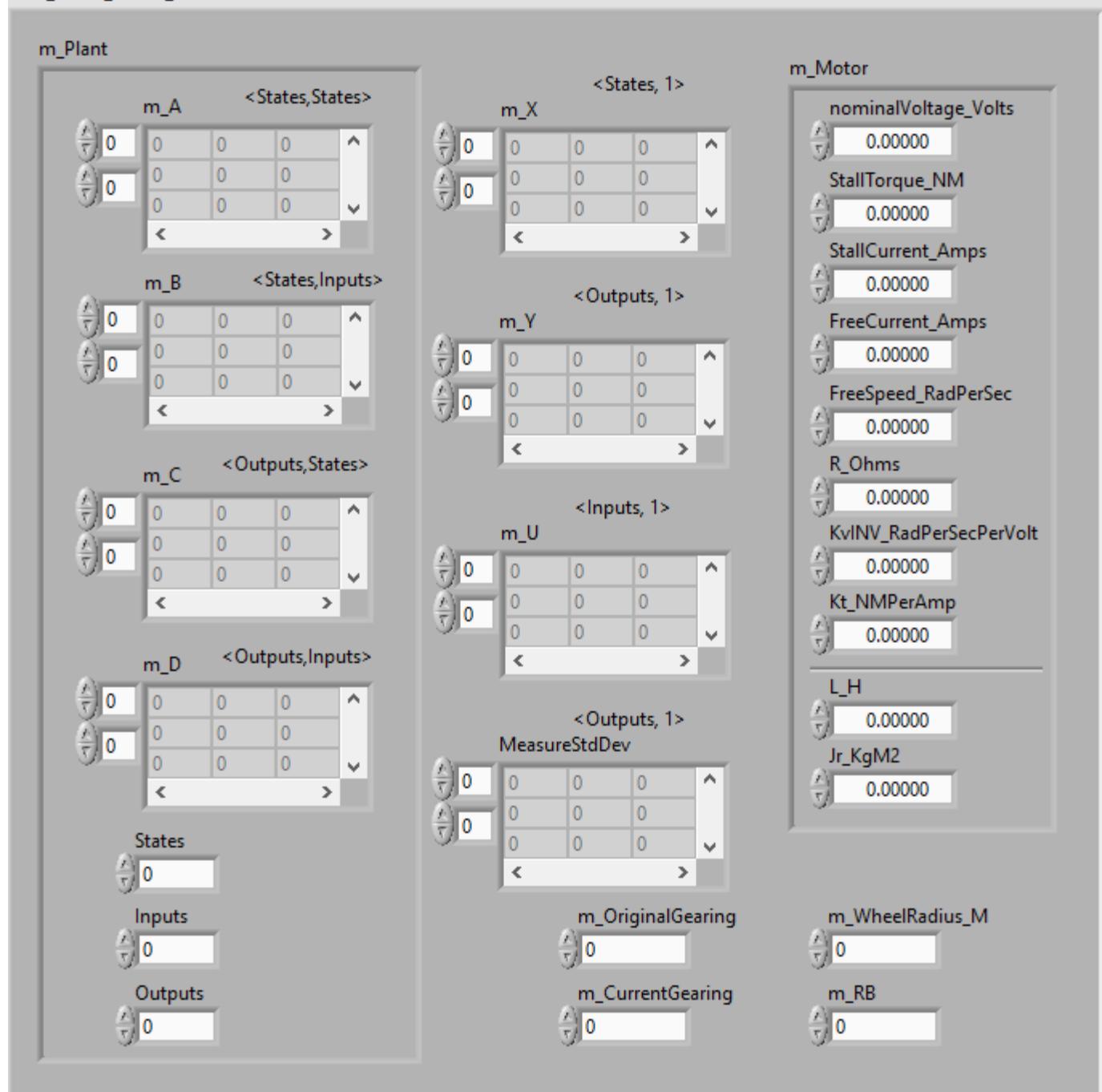
This is typically the control input of the last timestep from a LTVDiffDriveController.

$$y = x$$

The data values are:

- Plant -- Linear System
- X --
- Y --
- U --
- MeasureStdDev --
- Motor -- DC motor coefficients.
- OriginalGearing --
- CurrentGearing --
- WheelRadius -- (Meters)
- RB -- Radius of the robot base (TrackWidth / 2) (Meters)

DIFF_DRIVE_TRAIN_SIM

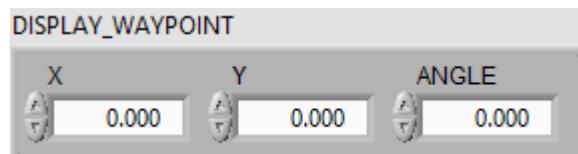


TypeDef-DISPLAY_WAYPOINT



Defines a trajectory waypoint. This is a convenience TypeDef for use with utility programs. The actual trajectory creation subVI take a number of different data types, which can all be derived from this way point.

In some cases, such as cubic spline creation, the angle data for the interior waypoints is ignored. The cubic routine calculates optimum angles.



TypeDef-DISPLAY_WEIGHTED_WAYPOINT



Defines a trajectory waypoint. This is a convenience TypeDef for use to display weighted waypoint. The actual trajectory creation subVI take a number of different data types, which can all be derived from this way point. In some cases, such as cubic spline creation, the angle data for the interior waypoints is ignored. The cubic routine calculates optimum angles.



TypeDef-ELEVATOR_SIM



Represents a simulated elevator mechanism.

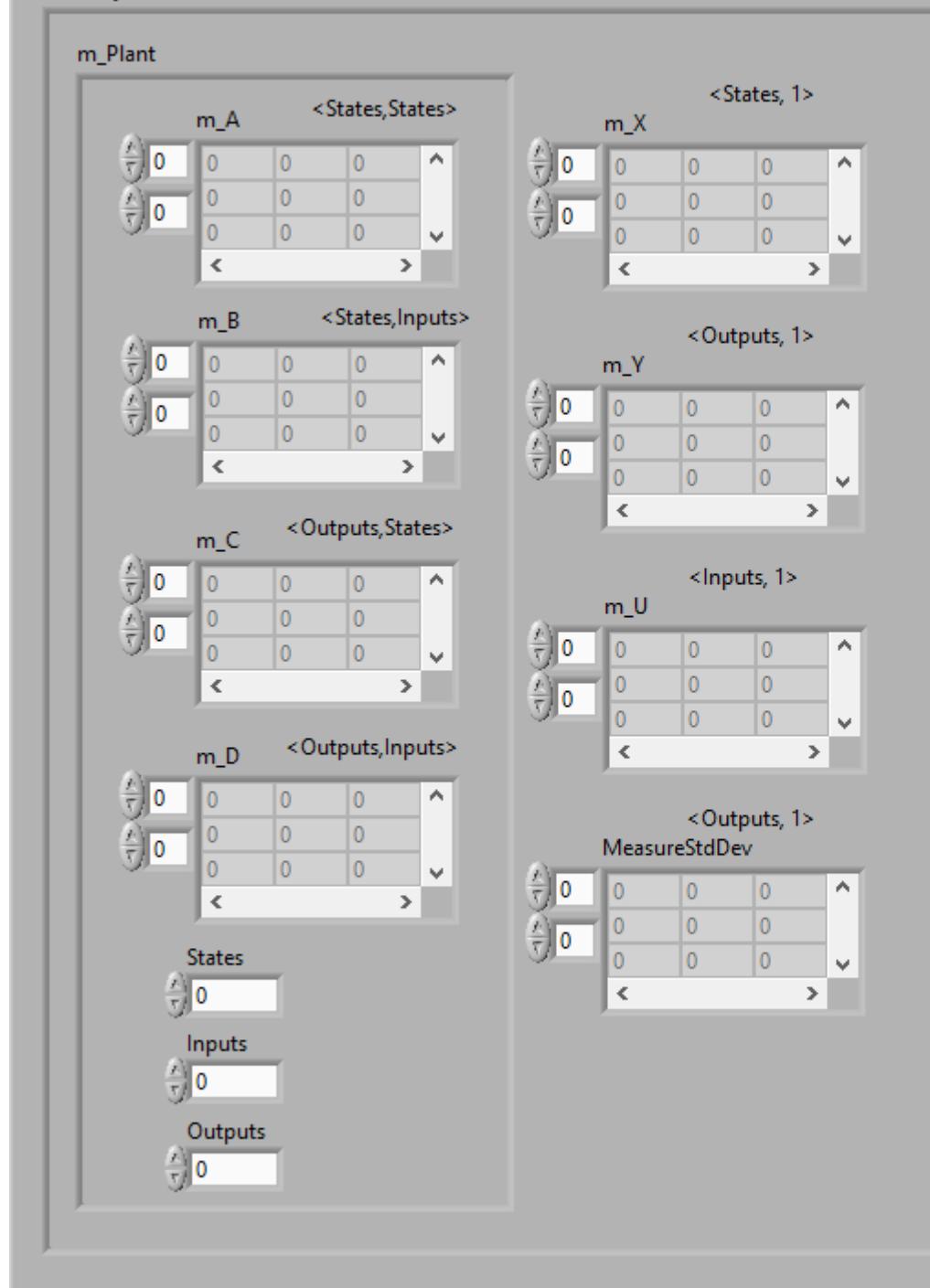
Cluster contains:

- Gearbox -- DCMotor cluster for the elevator.
- Gearing -- Double, gearing between the motors and the output.
- DrumRadius -- Double, the radius of the drum that the elevator spool is wrapped around. (Meters)
- MinHeight -- Double, the min allowable height for the elevator. (Meters)
- MaxHeight -- Double, the max allowable height for the elevator. (Meters)
- CarriageMass -- Double, the mass of the elevator carriage (Kilograms)

- LinearSystemSim -- Cluster for the simulated system.

ELEVATOR_SIM

LinearSystemSim



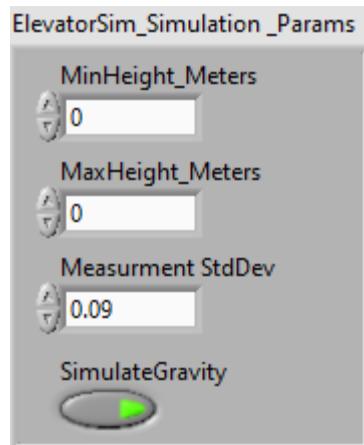
TypeDef-ELEVATOR_SIM_SIMULATION_PARAMS



Cluster contains simulation parameters used by the Elevator Simulation Execute routine.

Inputs:

- Min Height -- double -- Minimum physical elevator height (meters)
- Max Height -- double -- Maximum physical elevator height (meters)
- Pos Sim Std Dev -- double -- Standard deviation for the motor position (meters Default: 0.09)
- SimulateGravity -- boolean -- Simulate gravity (Default: True)



TypeDef-ELEV_FF

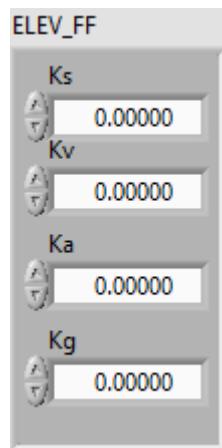


A helper set of functions that computes feedforward outputs for a simple elevator (modeled as a motor acting against the force of gravity).

Data cluster contents:

- Ks -- The static gain
- Kv -- The velocity gain
- Ka -- The acceleration gain

- Kg -- The gravity gain



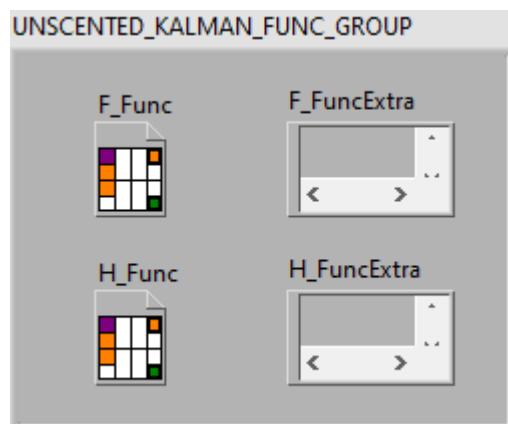
TypeDef-EXTENDED_KALMAN_CORRECT_FUNC_GROUP



This data cluster holds data for the call back functions used by the Extended Kalman Filter.

The cluster contains:

- F_Func -- Strictly typed function referenced.
- F_FuncExtra -- Variant containing extra data, if any, used by the callback function.
- H_Func -- Strictly typed function referenced.
- H_FuncExtra -- Variant containing extra data, if any, used by the callback function.



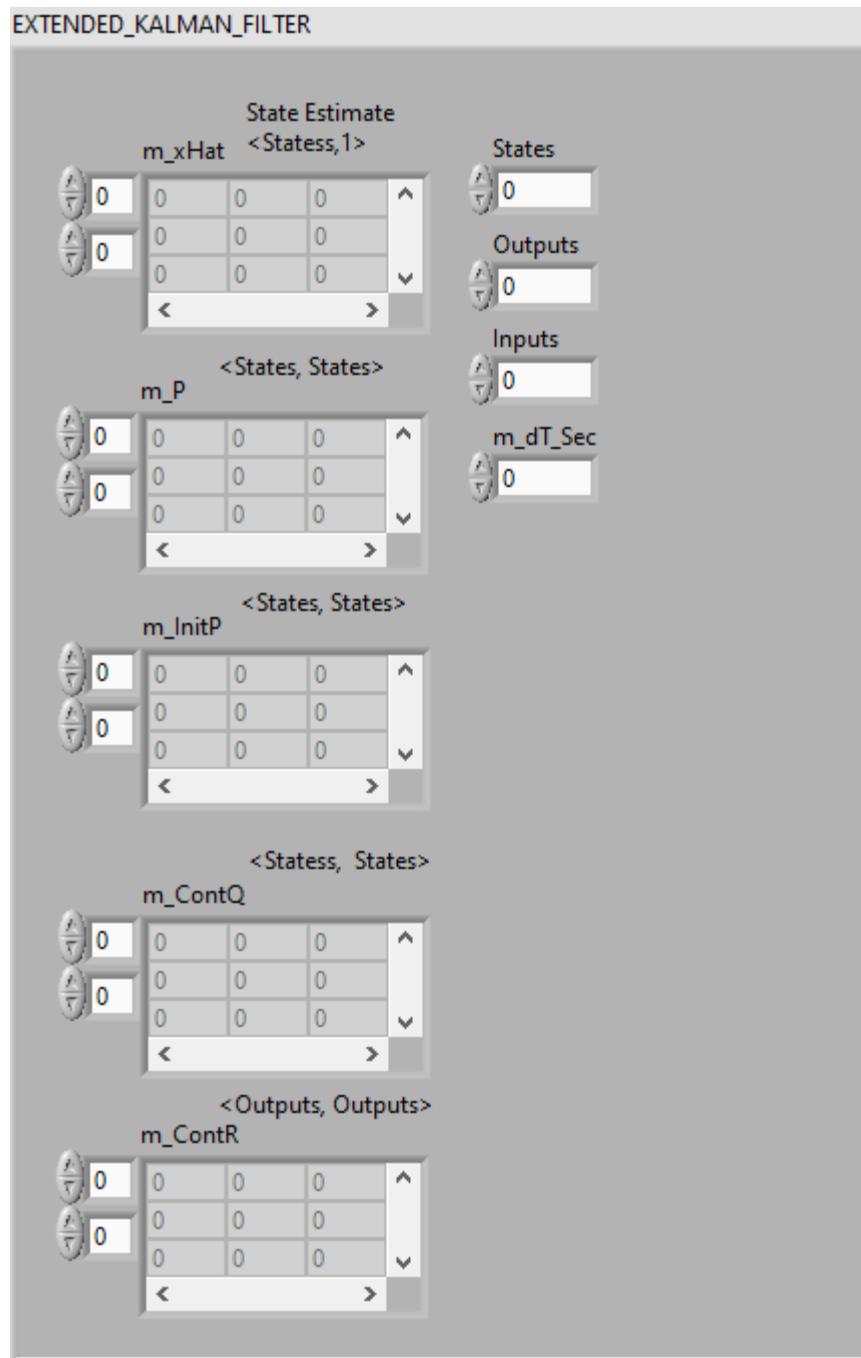
TypeDef-EXTENDED_KALMAN_FILTER



THIS IS A WORK IN PROGRESS. PROBABLY NOT FIT FOR USE YET...

Kalman filters combine predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

The Extended Kalman filter is just like the KalmanFilter Kalman filter, but we make a linear approximation of nonlinear dynamics and/or nonlinear measurement models. This means that the EKF works with nonlinear systems.



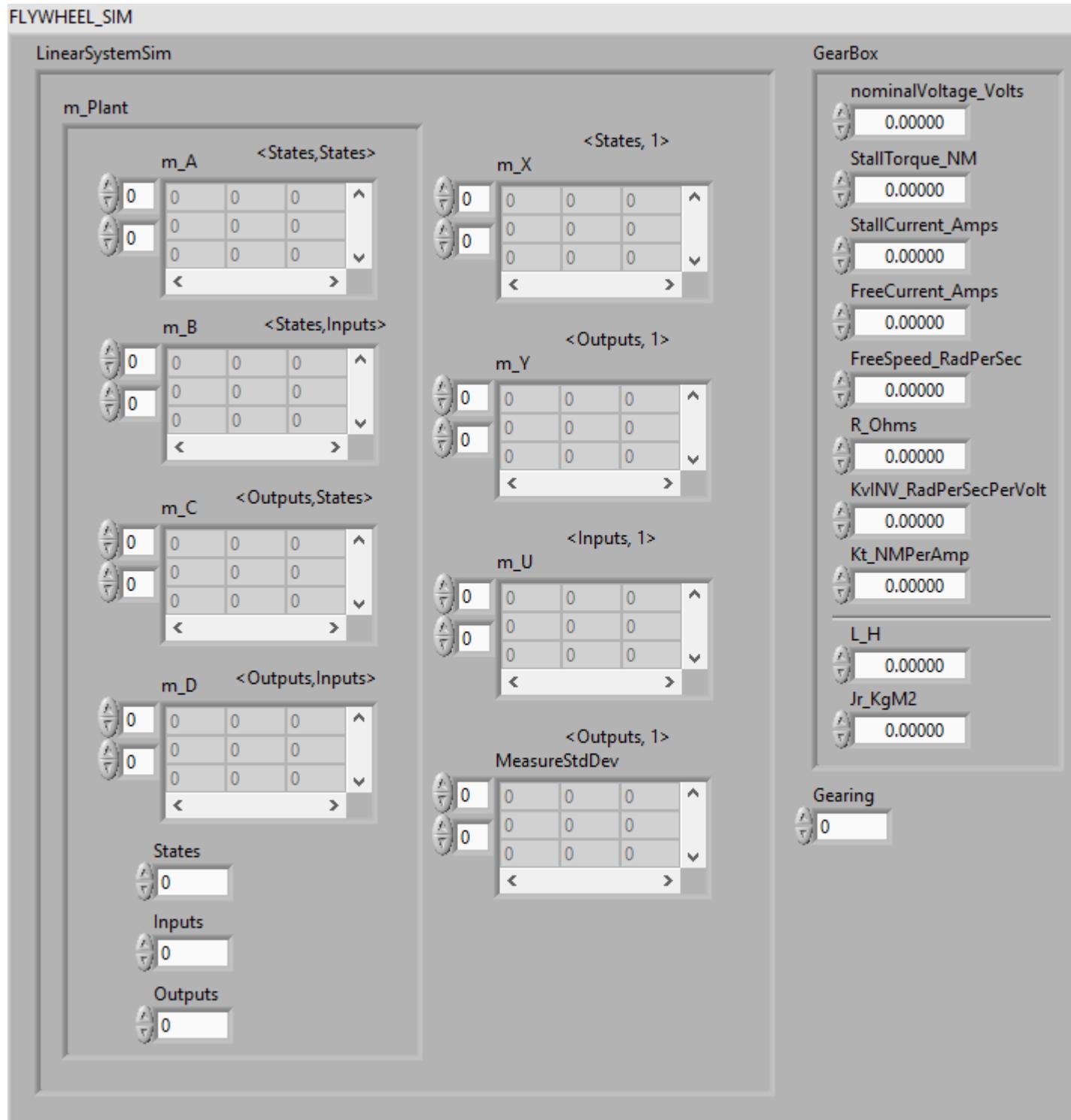
TypeDef-FLYWHEEL_SIM



Represents a simulated flywheel mechanism.

Cluster contains:

- Gearbox -- DCMotor cluster for the flywheel.
- Gearing -- Double, gearing between the motors and the output.
- LinearSystemSim -- Cluster for the simulation.



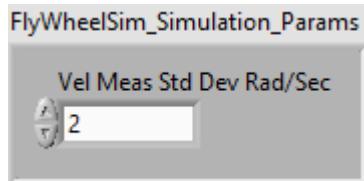
TypeDef-FLYWHEEL_SIM_SIMULATION_PARAMS



Cluster containing simulation parameters used by the Flywheel Simulation Execute routine.

Contains:

- Vel Sim Std Dev -- double -- Standard deviation for the flywheel velocity (rad/sec Default: 1)



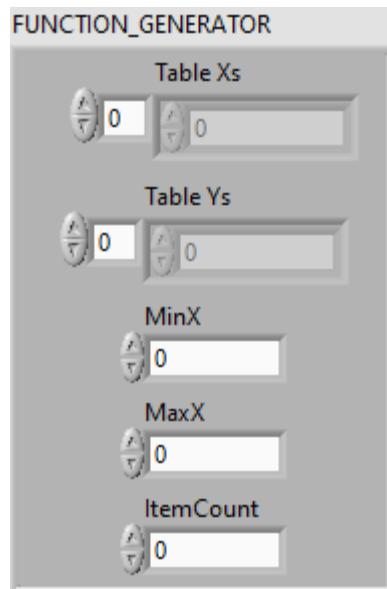
TypeDef-FUNCTION_GENERATOR



Function Generators (Interpolating Tree Maps) are used to get values at points that are not defined by making a guess from points that are defined. This uses linear interpolation.

The data cluster contains:

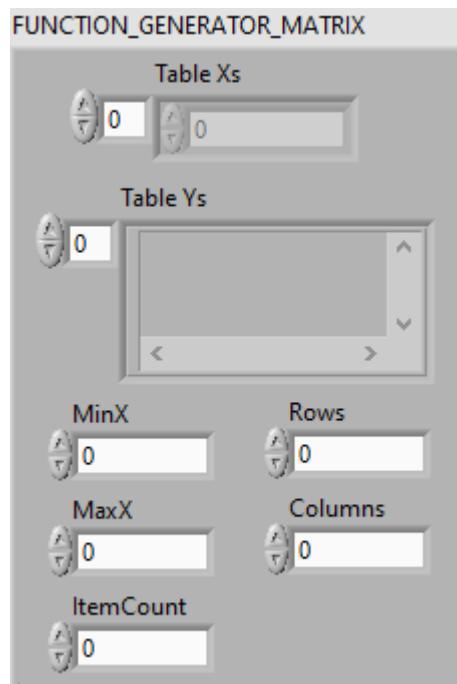
- Table Xs -- Array of X values
- Table Ys -- Array of Y values cooresponding the the X values.
- MinX -- Smallest X value
- MaxX -- Largest X value.
- ItemCount -- Number of items in the X and Y arrays.

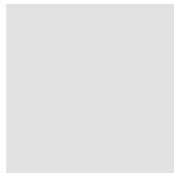


TypeDef-FUNCTION_GENERATOR_MATRIX

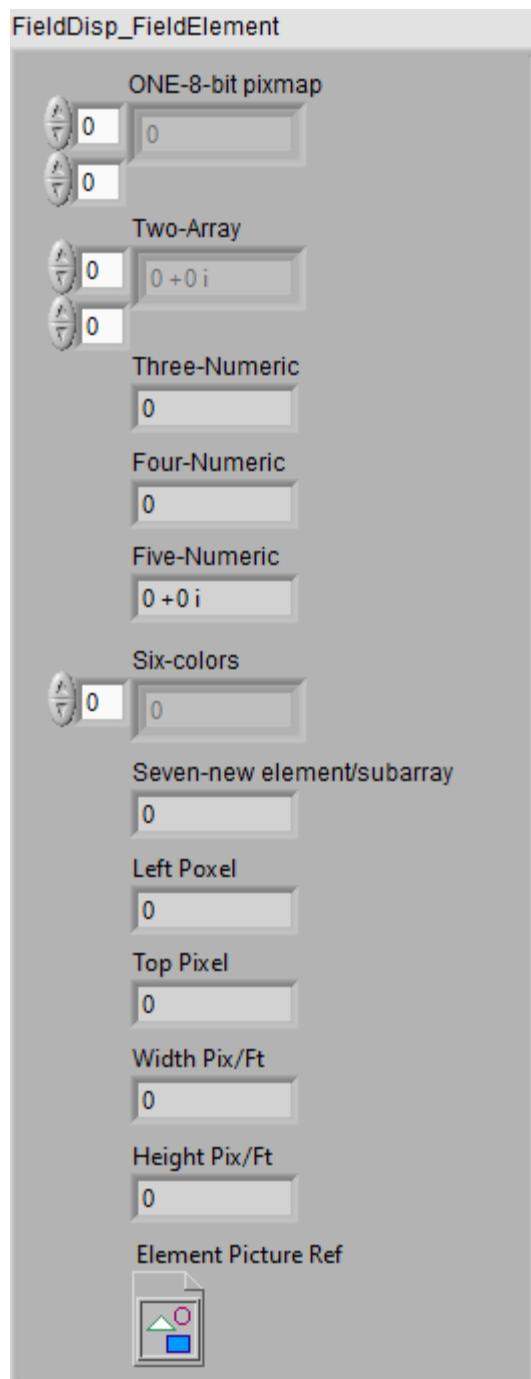


Function Generator data cluster



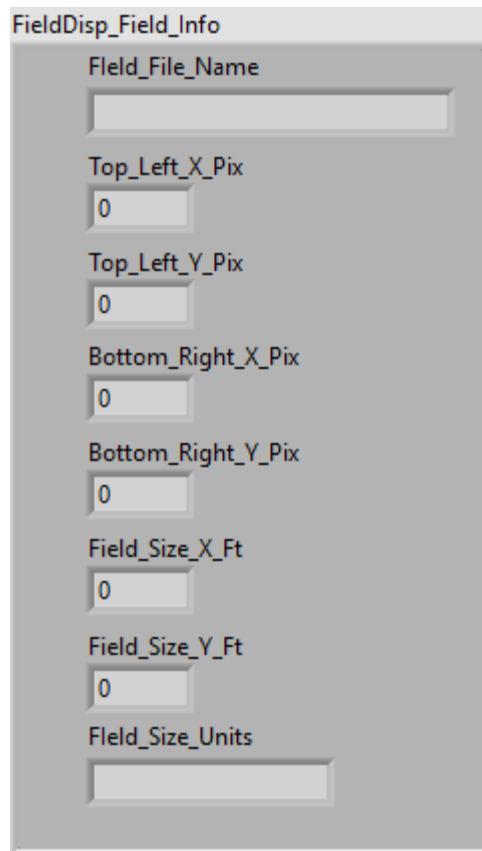
TypeDef-FieldDisp_ElementPicture

TypeDef-FieldDisp_FieldElement



TypeDef-FieldDisp_Field_Info





TypeDef-HOLONOMIC_DRV_CTRL



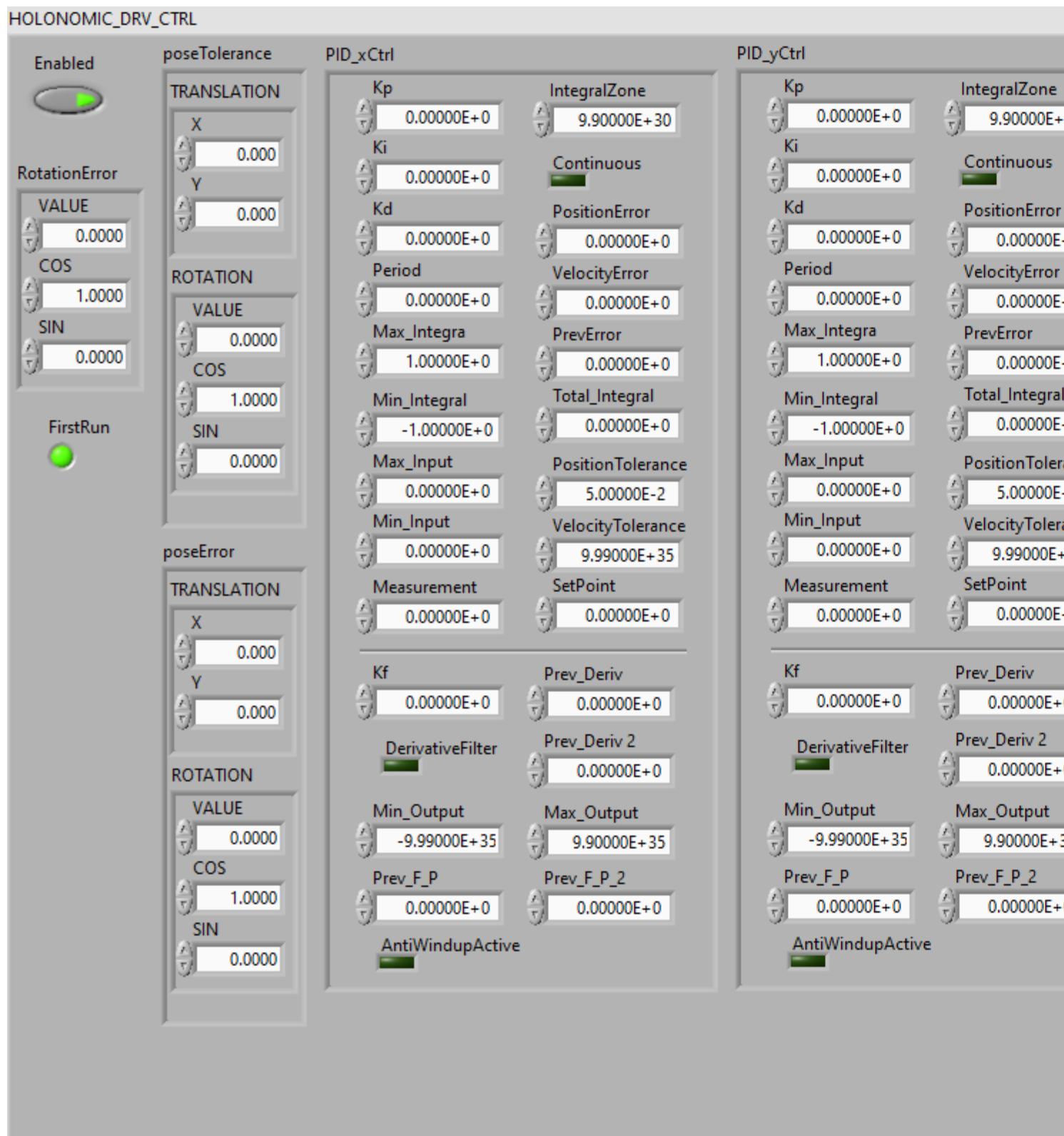
This holonomic drive controller can be used to follow trajectories using a holonomic drivetrain (i.e. swerve or mecanum). Holonomic trajectory following is a much simpler problem to solve compared to skid-steer style drivetrains because it is possible to individually control forward, sideways, and angular velocity.

The holonomic drive controller takes in one PID controller for each direction, forward and sideways, and one profiled PID controller for the angular direction. Because the heading dynamics are decoupled from translations, users can specify a custom heading that the drivetrain should point toward. This heading reference is profiled for smoothness.

The cluster contains:

- Enabled -- Boolean, indicating the controller is enabled.
- FirstRun -- Boolean, indicating the controller is running its first loop
- PoseTolerance -- Pose indicating allowable error for calculating At Reference.

- RotationError -- Rotation containing the current rotation error
- PoseError -- Pose containing the current position error
- PID_xCtrl -- PID to control the X position
- PID_yCtrl -- PID to control the Y position
- PROF_PID_ThetaCtrl -- Profiled PID to control the rotation.



TypeDef-HOLONOMIC_PACK_TUNING



This holonomic drive controller can be used to follow trajectories using a holonomic drivetrain (i.e. swerve or mecanum). Holonomic trajectory following is a much simpler problem to solve compared to skid-steer style drivetrains because it is possible to individually control forward, sideways, and angular velocity.

The holonomic drive controller takes in one PID controller for each direction, forward and sideways, and one profiled PID controller for the angular direction. Because the heading dynamics are decoupled from translations, users can specify a custom heading that the drivetrain should point toward. This heading reference is profiled for smoothness.

The cluster contains:

- Enabled -- Boolean, indicating the controller is enabled.
- FirstRun -- Boolean, indicating the controller is running its first loop
- PoseTolerance -- Pose indicating allowable error for calculating At Reference.
- RotationError -- Rotation containing the current rotation error
- PoseError -- Pose containing the current position error
- PID_xCtrl -- PID to control the X position
- PID_yCtrl -- PID to control the Y position
- PROF_PID_ThetaCtrl -- Profiled PID to control the rotation.

HOLONOMIC_CTRL_PACK_TUNING

X PID Tuning

Kp	0
Ki	0
Kd	0
MaximumIntegral	9.9E+30
MinimumIntegral	-9.9E+30
IntegralZone	9.9E+30

Y PID Tuning

Kp	0
Ki	0
Kd	0
MaximumIntegral	9.9E+30
MinimumIntegral	-9.9E+30
IntegralZone	9.9E+30

thetaProfiledPID_CTRL

Controller

Kp	0.00000E+0	IntegralZone	9.90000E+30
Ki	0.00000E+0	Continuous	Continuous
Kd	0.00000E+0	PositionError	0.00000E+0
Period	0.00000E+0	VelocityError	0.00000E+0
Max_Integral	1.00000E+0	PrevError	0.00000E+0
Min_Integral	-1.00000E+0	Total_Integral	0.00000E+0
Max_Input	0.00000E+0	PositionTolerance	5.00000E-2
Min_Input	0.00000E+0	VelocityTolerance	9.99000E+35
Measurement	0.00000E+0	SetPoint	0.00000E+0
<hr/>			
Kf	0.00000E+0	Prev_Deriv	0.00000E+0
DerivativeFilter	DerivativeFilter	Prev_Deriv 2	0.00000E+0
Min_Output	-9.99000E+35	Max_Output	9.90000E+35
Prev_F_P	0.00000E+0	Prev_F_P_2	0.00000E+0
AntiWindupActive			

Goal

Position	0.00000E+0
Velocity	0.00000E+0

Setpoint

Position	0.00000E+0
Velocity	0.00000E+0

Constraint

MaxVelocity	0.00000E+0
MaxAcceleration	0.00000E+0

TypeDef-IMPLICIT_MODEL_FOLLOWER

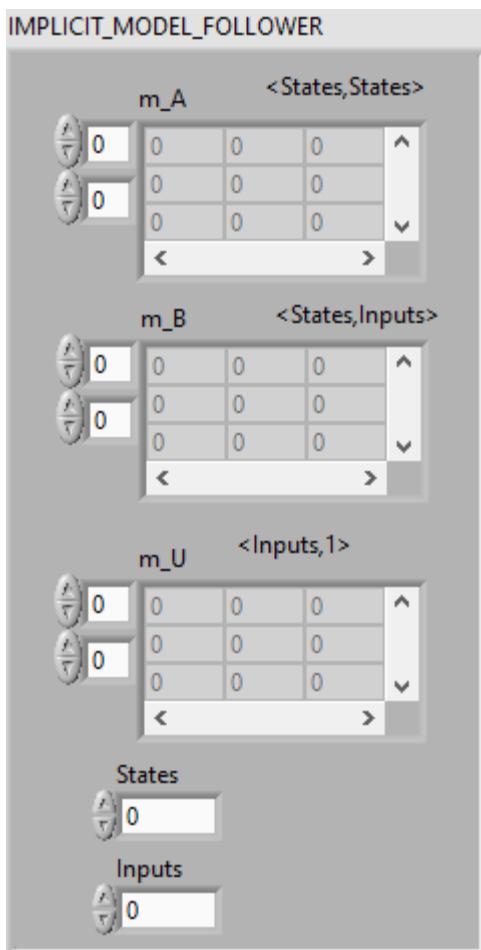


Contains the controller coefficients and logic for an implicit model follower.

Implicit model following lets us design a feedback controller that erases the dynamics of our system and makes it behave like some other system. This can be used to make a drivetrain more controllable during teleop driving by making it behave like a slower or more benign drivetrain.

The data cluster contains:

- m_A -- State space conversion gain
- m_B -- Input space conversion gain
- m_U -- Computed controller output
- states
- inputs



TypeDef-KALMAN_FILTER



A Kalman filter combines predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

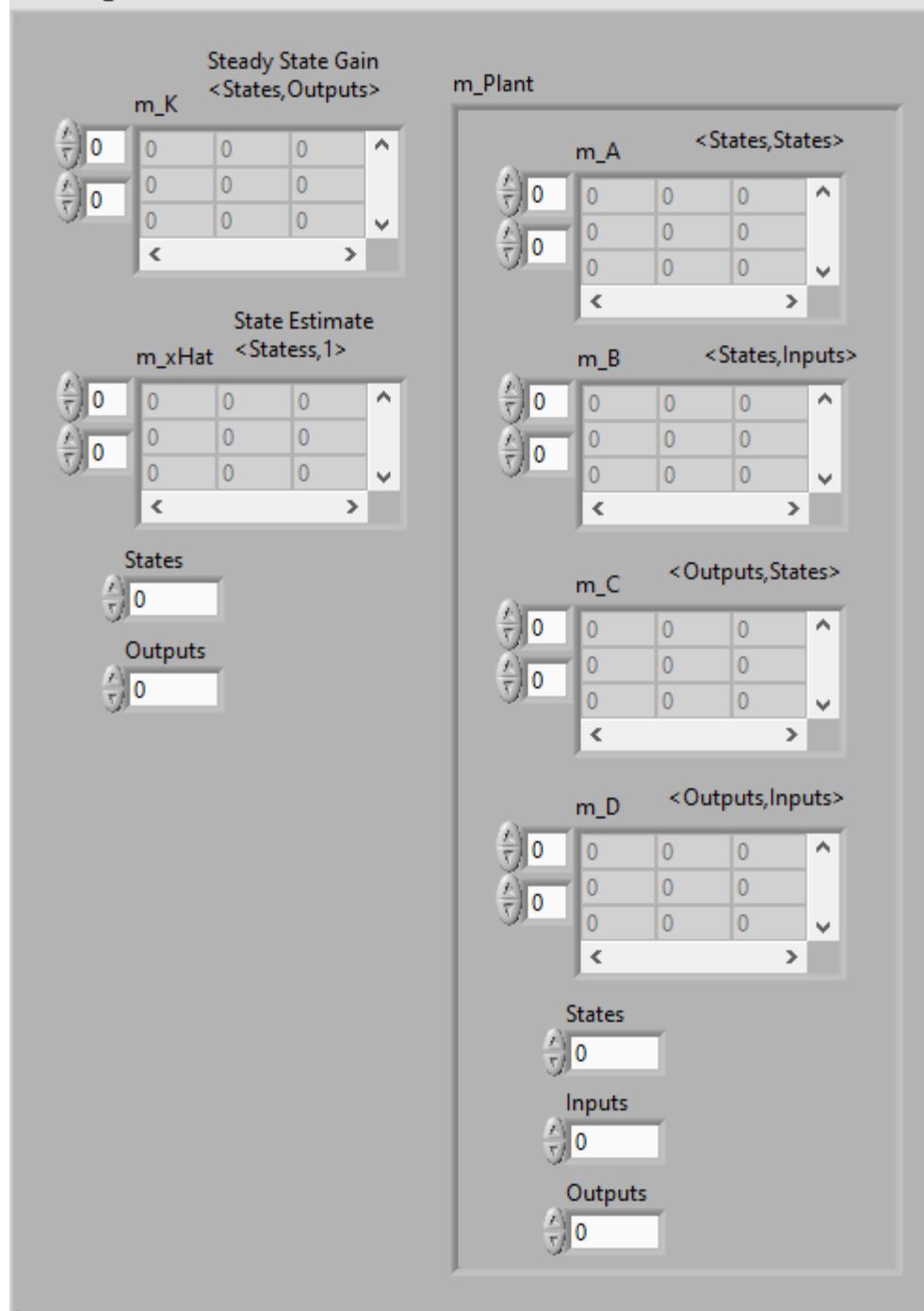
Kalman filters use a K gain matrix to determine whether to trust the model or measurements more. Kalman filter theory uses statistics to compute an optimal K gain which minimizes the sum of squares error in the state estimate. This K gain is used to correct the state estimate by some amount of the difference between the actual measurements and the measurements predicted by the model.

For more on the underlying math, read <https://file.tavsys.net/control/controls-engineering-in-frc.pdf> chapter 9 "Stochastic control theory".

The cluster contains:

- m_Plant -- Linear System data cluster
- m_K -- Steady state kalman gain
- m_xHat -- State estimate
- States -- Number of states
- Outputs -- Number of outputs

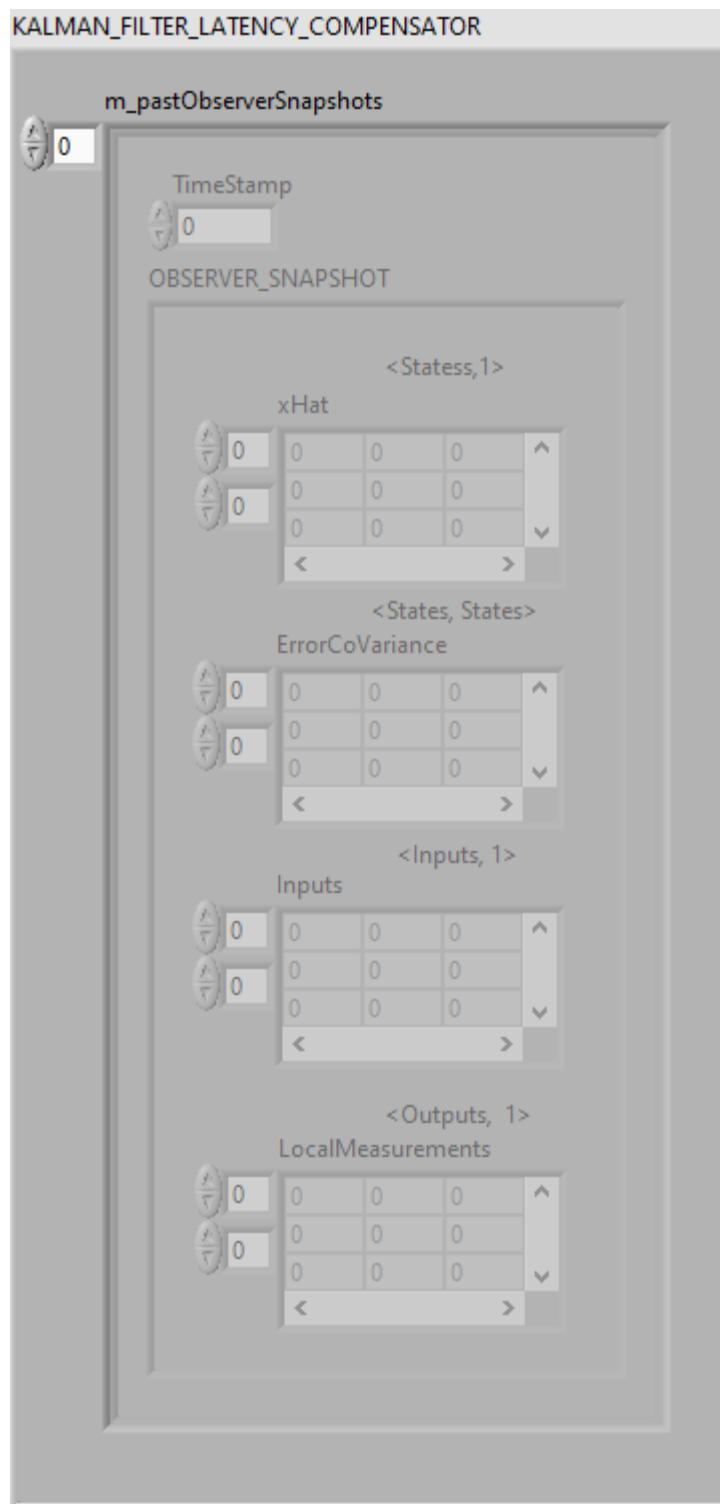
KALMAN_FILTER

**TypeDef-KALMAN_FILTER_LATENCY_COMP**

Data cluster holding past Observer snapshots that can be used to compensate for time latency.

The cluster contains:

- PastObserverSnapshots -- An array of Observer_Snap_List_Item containing:
 - TimeStamp -- Time stamp of snapshot
 - ObserverSnapShot -- System state at time of snapshot.



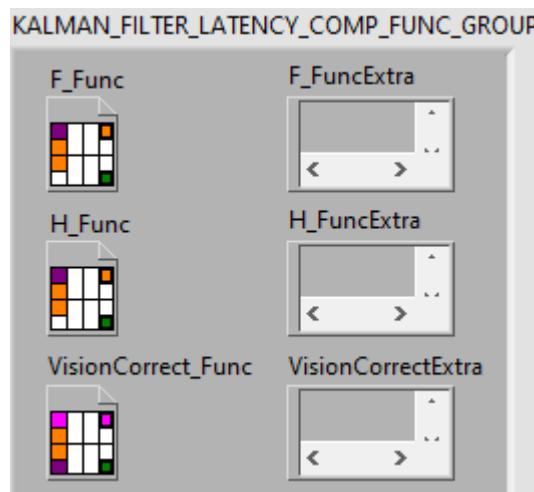
TypeDef-KALMAN_FILTER_LATENCY_COMP_FUNC_GROUP



This data cluster holds data for the call back functions used by the Kalman Filter Latency Compensator.

The cluster contains:

- F_Func -- Strictly typed function referenced.
- F_FuncExtra -- Variant containing extra data, if any, used by the callback function.
- H_Func -- Strictly typed function referenced.
- H_FuncExtra -- Variant containing extra data, if any, used by the callback function.
- VisionCorrect_Func -- Strictly typed function referenced.
- VisionCorrectExtra -- Variant containing extra data, if any, used by the callback function.



TypeDef-LINEAR_FILTER



This cluster stores the data for a set of VIs that implement a linear, digital filter. All types of FIR and IIR filters are supported. A set of VIs are provided to create commonly used types of filters.

Filters are of the form:

$$y[n] = (b_0*x[n] + b_1*x[n-1] + \dots + b_P*x[n-P]) - (a_0*y[n-1] + a_2*y[n-2] + \dots + a_Q*y[n-Q])$$

Where:

- $y[n]$ is the output at time "n"
- $x[n]$ is the input at time "n"
- $y[n-1]$ is the output from the LAST time step ("n-1")
- $x[n-1]$ is the input from the LAST time step ("n-1")
- $b_0 \dots b_P$ are the "feedforward" (FIR) gains
- $a_0 \dots a_Q$ are the "feedback" (IIR) gains

IMPORTANT! Note the "-" sign in front of the feedback term! This is a common convention in signal processing.

What can linear filters do? Basically, they can filter, or diminish, the effects of undesirable input frequencies. High frequencies, or rapid changes, can be indicative of sensor noise or be otherwise undesirable. A "low pass" filter smooths out the signal, reducing the impact of these high frequency components. Likewise, a "high pass" filter gets rid of slow-moving signal components, letting you detect large changes more easily.

Example FRC applications of filters:

- Getting rid of noise from an analog sensor input (note: the roboRIO's FPGA can do this faster in hardware)
- Smoothing out joystick input to prevent the wheels from slipping or the robot from tipping
- Smoothing motor commands so that unnecessary strain isn't put on electrical or mechanical components
- If you use clever gains, you can make a PID controller out of this class! (Use the PID set of VI's instead...)

For more on filters, we highly recommend the following articles:

https://en.wikipedia.org/wiki/Linear_filter

https://en.wikipedia.org/wiki/Iir_filter

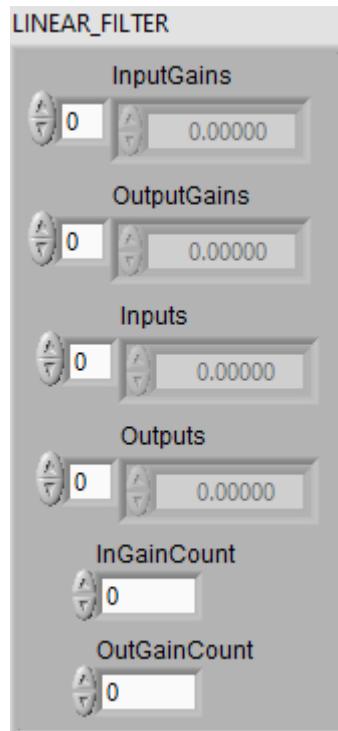
https://en.wikipedia.org/wiki/Fir_filter

Note 1: calculate() should be called by the user on a known, regular period. You can use code in a periodic function.

Note 2: For ALL filters, gains are necessarily a function of frequency. If you make a filter that works well for you at, say, 100Hz, (executing every 10 milliseconds), you will most definitely need to adjust the gains if you then want to run it at 200Hz, (executing every 5 milliseconds)! Combining this with Note 1, the impetus is on YOU as a developer to make sure calculate() gets called at the desired, constant frequency!

Elements:

- InputGains - Array of feedforward or FIR gain factors (bx)
- OutputGains - Array of feedback or IIR gain factors (ax)
- Inputs - Array of the last n saved inputs
- Outputs - Array of the last n saved outputs
- InGainCount - Number of input gain terms
- OutGainCount - number of output gain terms



TypeDef-LINEAR_PLANT_INV_FF



A plant inversion model-based feedforward from a LinearSystem

The feedforward is calculated as

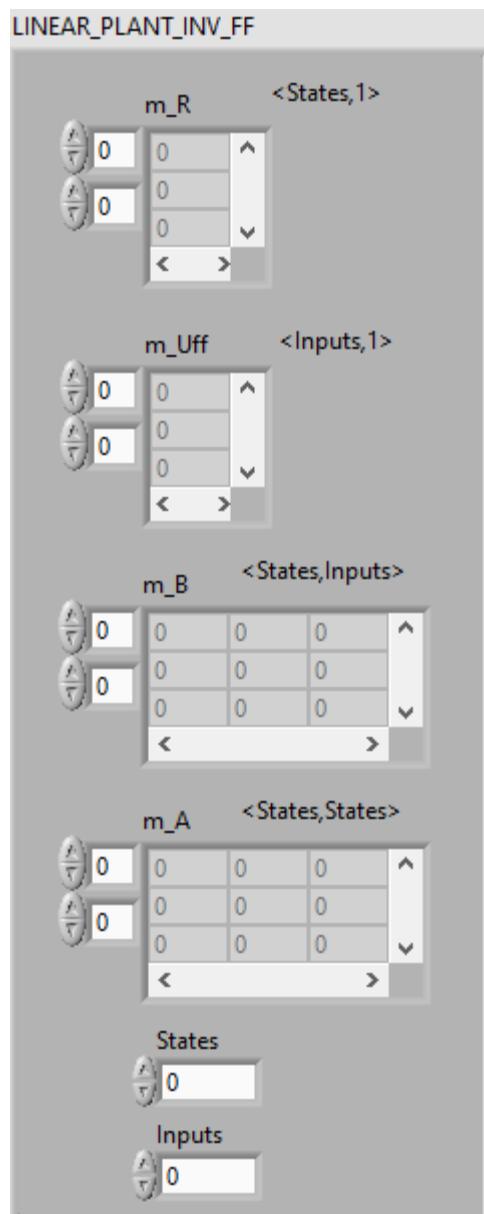
$$u_{ff} = B'' (r_{k+1} - A r_k)$$

,

where B'' is the pseudoinverse of B .

This cluster contains:

- m_r -- The current reference state
- m_{uff} -- The computed feedforward.
- m_B -- Discrete input matrix of the plant being controlled.
- m_A -- Discrete system matrix of the plant being controlled
- States -- Number of states
- Inputs -- Number of inputs



TypeDef-LINEAR_QUADRATIC_REGULATOR

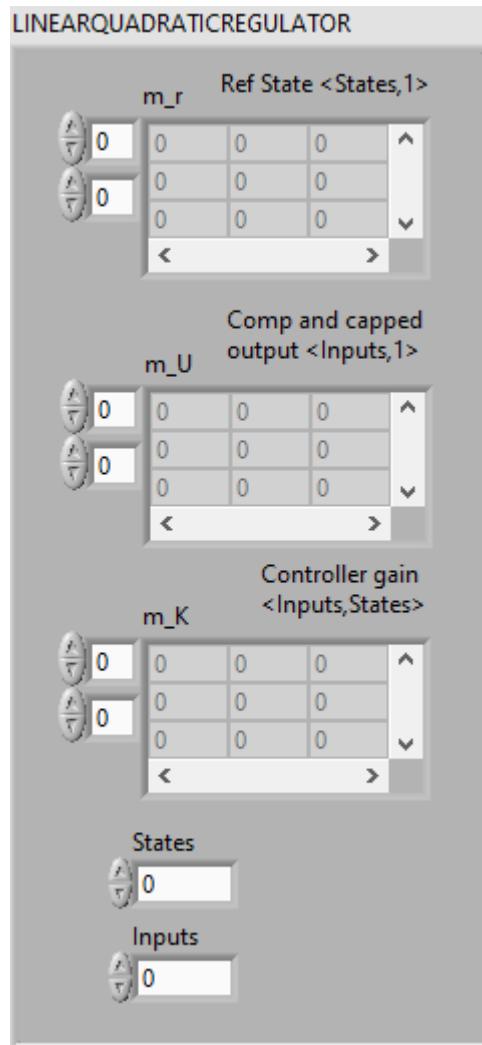


Contains the controller coefficients and logic for a linear-quadratic regulator (LQR). LQRs use the control law $u = K(r - x)$.

This cluster contains:

- **m_r** -- The current reference state.

- m_u -- The computed and capped controller output.
- m_K -- Controller gain.
- States -- Number of states
- Inputs -- Number of inputs.



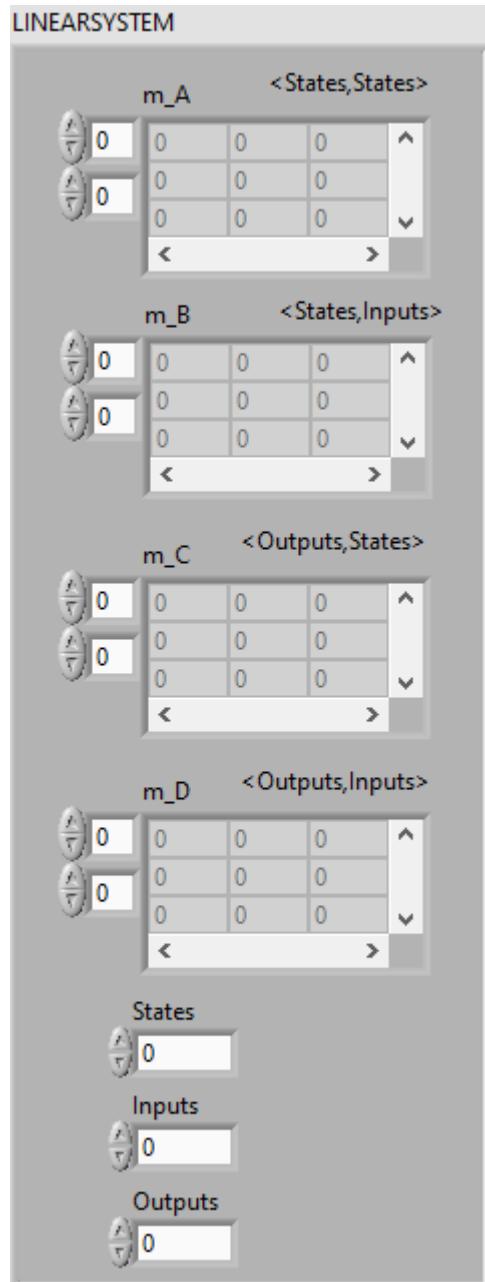
TypeDef-LINEAR_SYSTEM



Holds the data for a linear system. The data cluster contains:

- m_A -- Continuous system matrix.

- m_B -- Continuous input matrix.
- m_C -- Output matrix.
- m_D -- Feedthrough matrix.

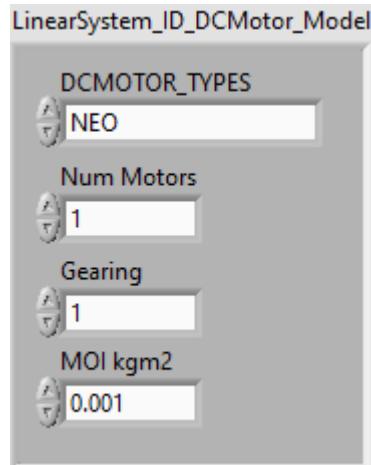


TypeDef-LINEAR_SYSTEM_ID_DCMOTOR_MODEL

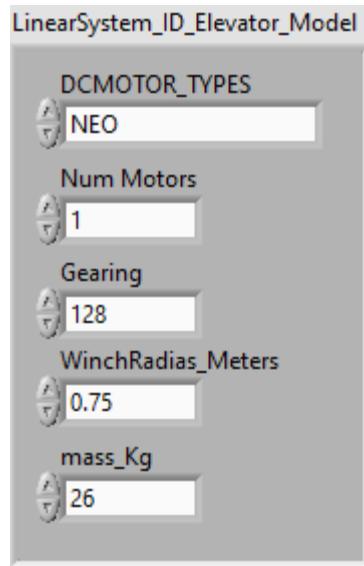
Cluster contains model parameters used to create a DC Motor Linear System.

Contains:

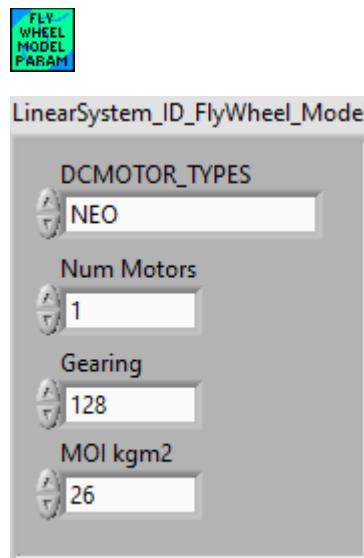
- DCMOTOR_TYPE -- enum -- Model of motor being used.
- Num Motors -- integer -- Number of motors used.
- Gearing -- double -- Gear ratio. (Input speed / Output Speed)
- MOI -- double -- Moment of inertia - kg m²



TypeDef-LINEAR_SYSTEM_ID_ELEVATOR_MODEL

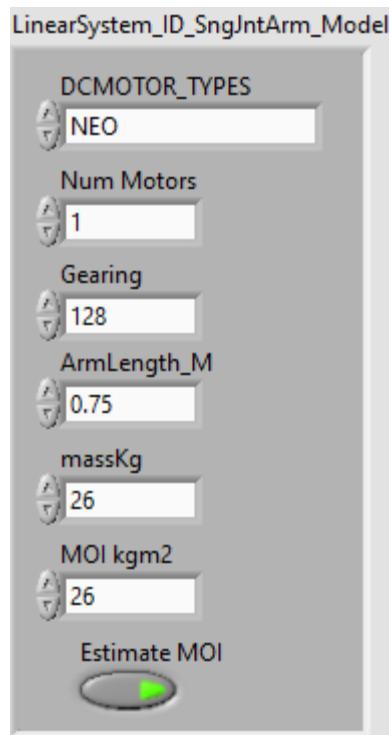


TypeDef-LINEAR_SYSTEM_ID_FLYWHEEL_MODEL



TypeDef-LINEAR_SYSTEM_ID_SINGLE_JOINT_ARM_MODEL





TypeDef-LINEAR_SYSTEM_LOOP



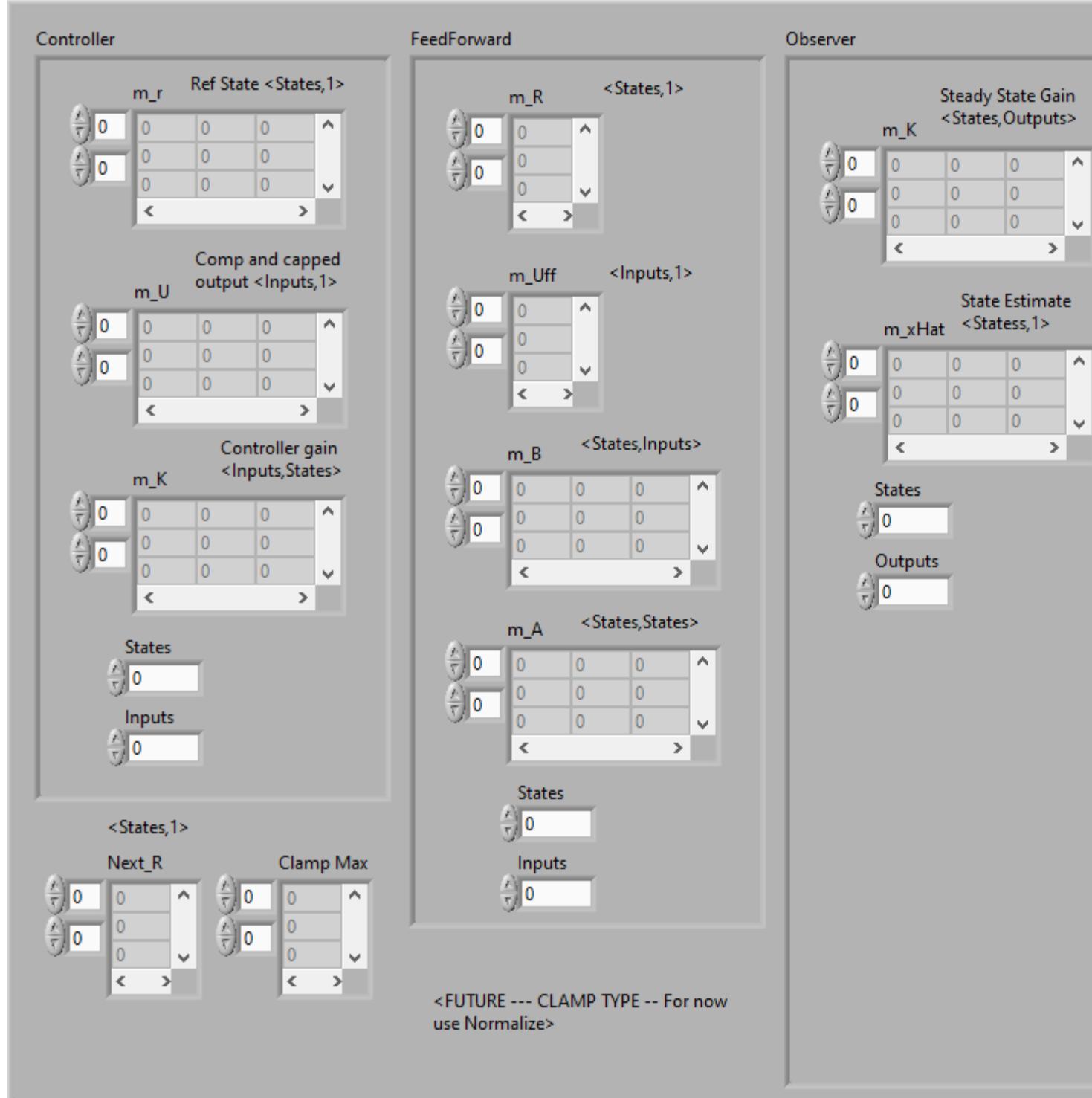
Combines a controller, feedforward, and observer for controlling a mechanism with full state feedback.

For everything in this file, "inputs" and "outputs" are defined from the perspective of the plant. This means U is an input and Y is an output (because you give the plant U (powers) and it gives you back a Y (sensor values)). This is the opposite of what they mean from the perspective of the controller (U is an output because that's what goes to the motors and Y is an input because that's what comes back from the sensors).

This cluster contains:

- Controller -- Linear Quadratic Regulator data cluster
- Feedforward -- Linear Plant Inversion Feedforward data cluster
- Observer -- Kalman Filter data cluster
- Next_R -- Controllers next reference
- Clamp Max -- Maximum clamping values

LINEAR_SYSTEM_LOOP



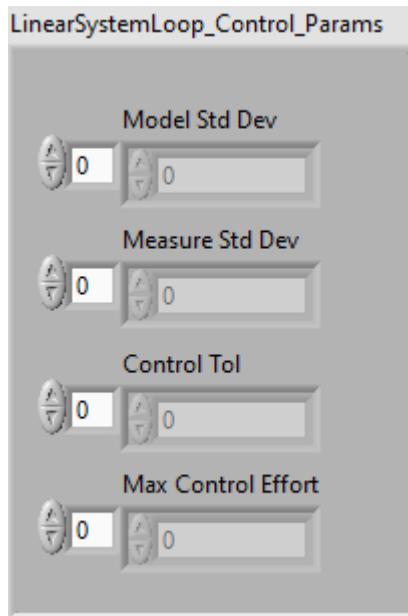
TypeDef-LINEAR_SYSTEM_LOOP_CTRL_PARAMS



Cluster containing control parameters needed for the Linear System Loop Execute function.

Contains:

- Model Std Dev -- double array (states) -- Array of standard deviations for model states.
- Measure Std Dev -- double array(outputs) -- Array of standard deviations for model outputs
- Control Tol -- double array(states) -- Array of standard deviations for controlled model states.
- Max Control Effort -- double array(inputs) -- Array of maximum values for inputs (classical control outputs)



TypeDef-LINEAR_SYSTEM_LOOP_DCMOTOR_CTRL_PARAMS



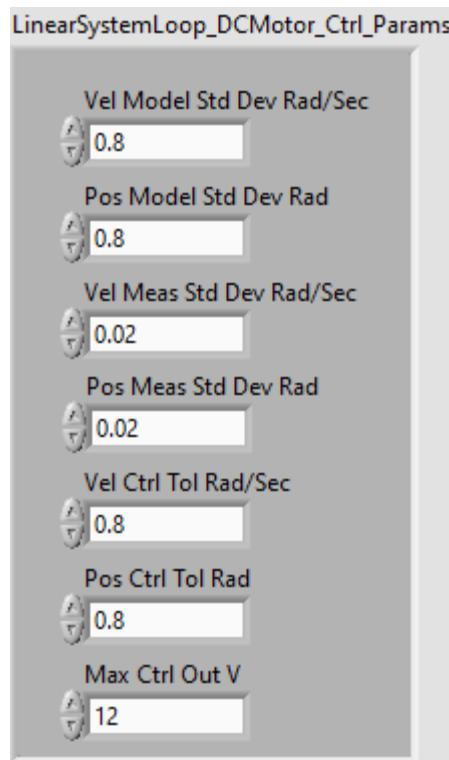
Cluster to contain control parameters for a DC motor linear system.

The DC Motor linear system is defined as:

- states:
 - angular position (rad)
 - angular velocity (rad/sec)
- inputs:
 - motor voltage
- outputs:
 - angular position (rad)
 - angular velocity (rad/sec)

Contains:

- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (rad/sec)
- Pos Model Std Dev -- double -- Standard deviation for modeled position (rad)
- Vel Meas Std Dev -- double -- Standard deviation for measured velocity (rad/sec)
- Pos Meas Std Dev -- double -- Standard deviation for measured position (rad)
- Vel Ctrl Tol Std Dev -- double -- Control tolerance for velocity (rad/sec)
- Pos Ctrl Tol Std Dev -- double -- Control tolerance for position (rad)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)



TypeDef-LINEAR_SYSTEM_LOOP_DIFF_DRV_CTRL_PARAMS



Cluster contains the control parameters for a Diff Drive linear system.

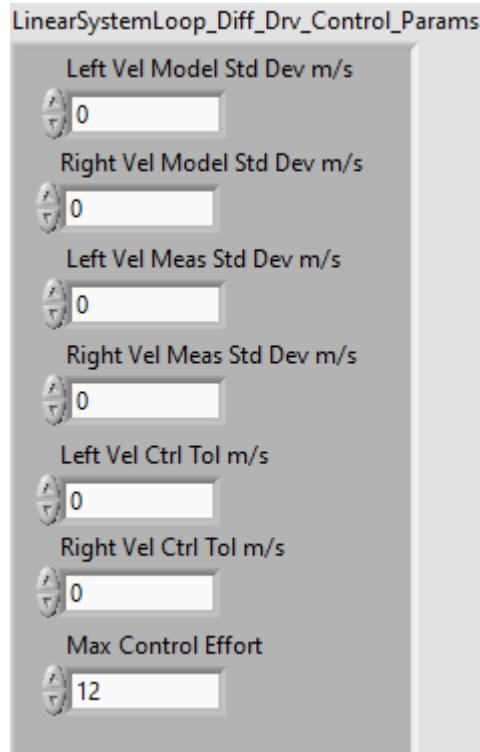
The Diff Drive linear system is defined as:

- states
 - left velocity (m/s)
 - right velocity (m/s)
- inputs
 - left voltage (volts)
 - right voltage(volts)
- outputs
 - left velocity (m/s)

- right velocity (m/s)

Contains:

- Left Vel Model Std Dev -- double -- Standard deviation for modeled left velocity (meters/sec)
- Right Vel Model Std Dev -- double -- Standard deviation for modeled right velocity (meters/sec)
- Left Vel Meas Std Dev -- double -- Standard deviation for measured left velocity (meters/sec)
- Right Vel Meas Std Dev -- double -- Standard deviation for measured right velocity (meters/sec)
- Left Vel Ctrl Tol Std Dev -- double -- Control tolerance for left velocity (meters/sec)
- Right Vel Ctrl Tol Std Dev -- double -- Control tolerance for right velocity (meters/sec)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)



TypeDef-LINEAR_SYSTEM_LOOP_ELEVATOR_CTRL_PARAMS



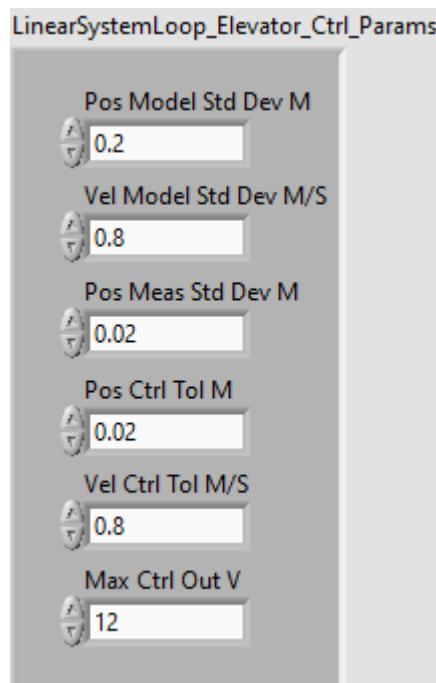
Cluster contains the control parameters for an elevator linear system.

The Elevator linear system definition is:

- States
 - position (meters)
 - velocity (meters/sec)
- Inputs:
 - motor voltage (volts)
- Outputs:
 - position (meters)

Contains:

- Pos Model Std Dev -- double -- Standard deviation for modeled position (meters)
- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (meters/sec)
- Pos Meas Std Dev -- double -- Standard deviation for measured position (meters)
- Pos Ctrl Tol Std Dev -- double -- Control tolerance for position (meters)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)



TypeDef-LINEAR_SYSTEM_LOOP_FLYWHEEL_CTRL_PARAMS



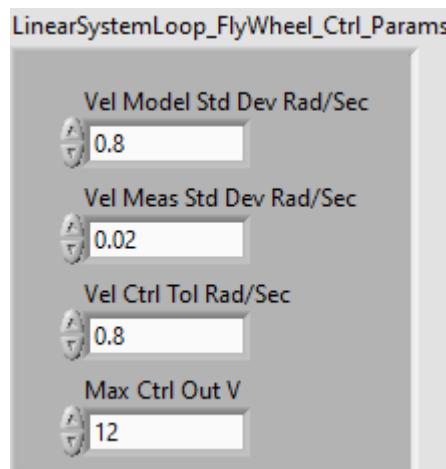
Cluster contains control parameters for a flywheel linear system.

The flywheel linear system definition is:

- States
 - angular velocity (Rad/Sec)
- Inputs:
 - motor voltage
- Outputs:
 - angular velocity (rad/sec)

Contains:

- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (rad/sec)
- Vel Meas Std Dev -- double -- Standard deviation for measured velocity (rad/sec)
- Vel Ctrl Tol Std Dev -- double -- Control tolerance for velocity (rad/sec)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)



TypeDef-LINEAR_SYSTEM_LOOP_SNGJNTARM_CTRL_PARAMS

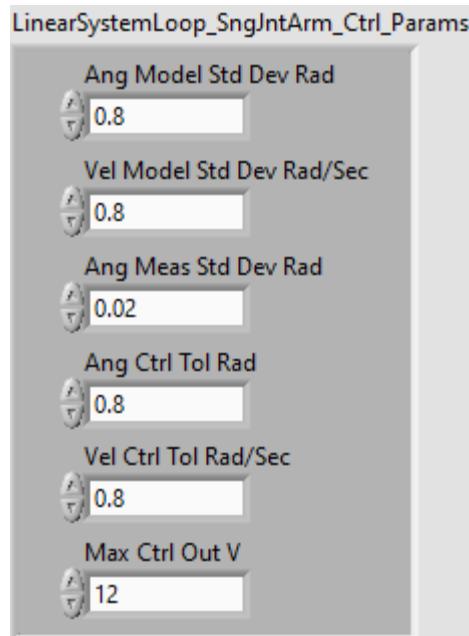
Cluster contains the control parameters for a single jointed arm linear system.

The single jointed arm linear system definition is:

- States
 - angle position (Rad)
 - velocity (Rad/Sec)
- Inputs:
 - motor voltage
- Outputs:
 - angle position (rad)

Contains:

- Ang Model Std Dev -- double -- Standard deviation for modeled angle (Rad)
- Vel Model Std Dev -- double -- Standard deviation for modeled velocity (rad/sec)
- Ang Measl Std Dev -- double -- Standard deviation for measured angle (Rad)
- Ang Ctrl Tol Std Dev -- double -- Control tolerance for angle (rad)
- Max Ctrl Out -- double -- Maximum control output (Default: 12) (volts)



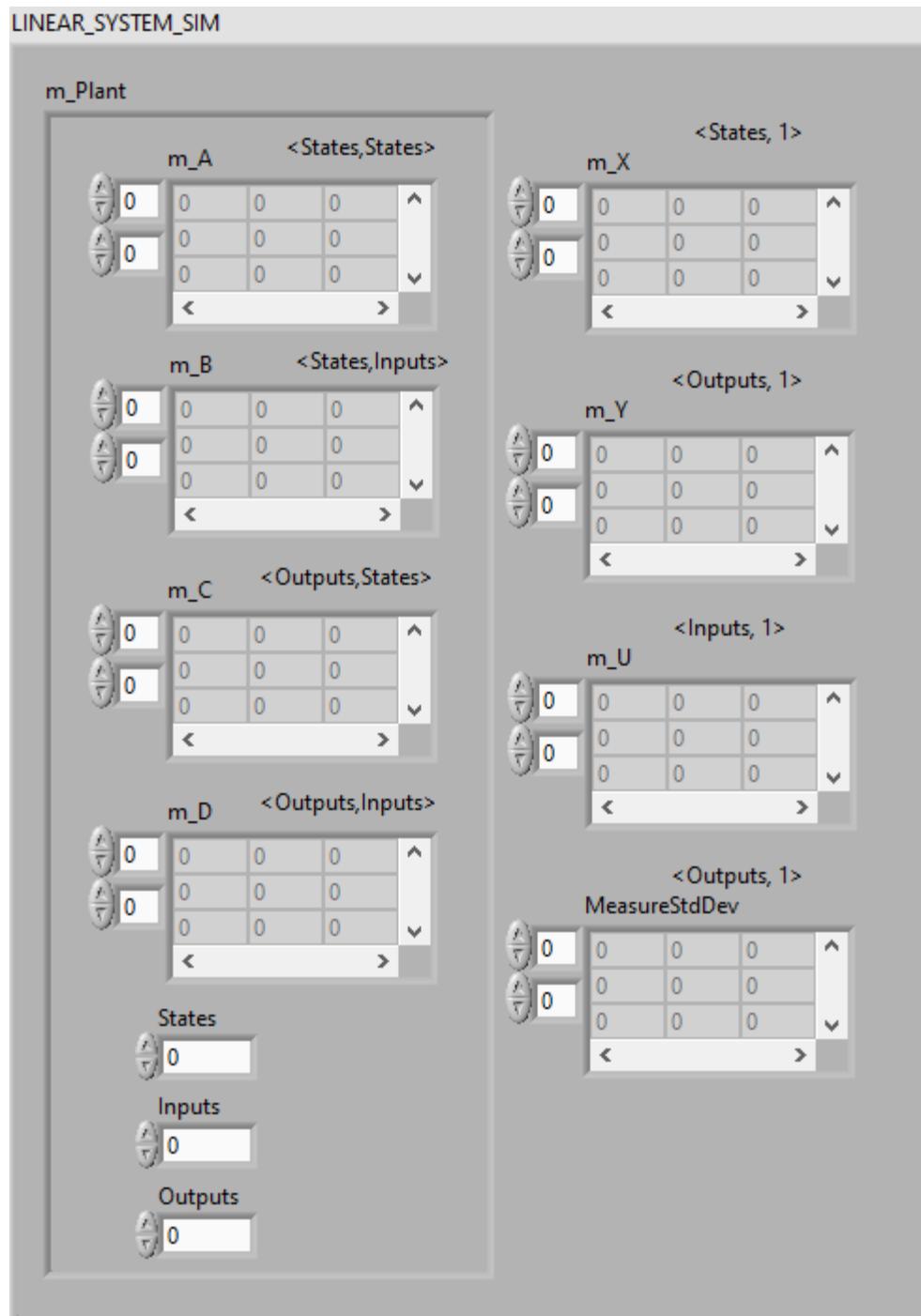
TypeDef-LINEAR_SYSTEM_SIM



This cluster helps simulate linear systems. To use this class, implement a loop, perhaps in Periodic Tasks. Then call "setInput" with the inputs to the system (usually voltage). Then call "update" to update the simulation. Then, set simulated sensor readings with the simulated positions in "getOutput".

The cluster contains:

- m_Plant -- The plant that represents the linear system.
- m_X -- state variable
- m_Y -- output variable
- m_U -- input variable
- MeasureStdDev -- The standard deviations of measurements, used for adding noise to the measurements.



TypeDef-LTV_DIFF_DRIVE_CTRL



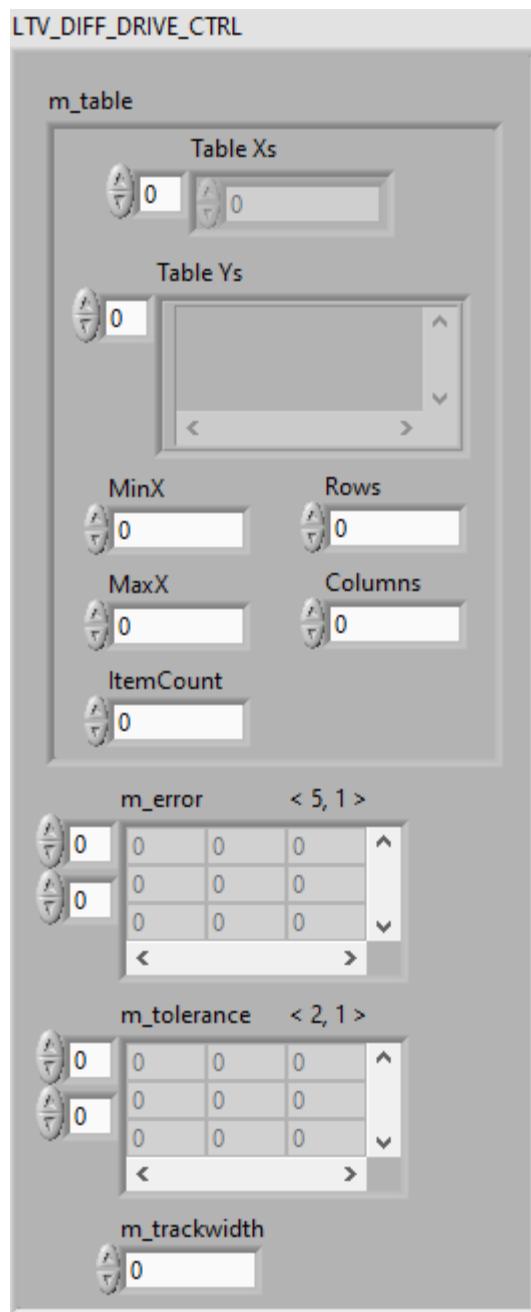
The linear time-varying differential drive controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state. We

precomputed gains for important places in our state-space, then interpolated between them with a LUT to save computational resources. Filters the provided voltages to limit a differential drive's linear and angular acceleration.

The differential drive model can be created via the functions in `LinearSystemId`.

The data values are:

- `table` -- Function Generator Matrix LUT (look up table) from drivetrain linear velocity to LQR gain
- `error`
- `tolerance`
- `trackWidth` -- trackwidth (meters)



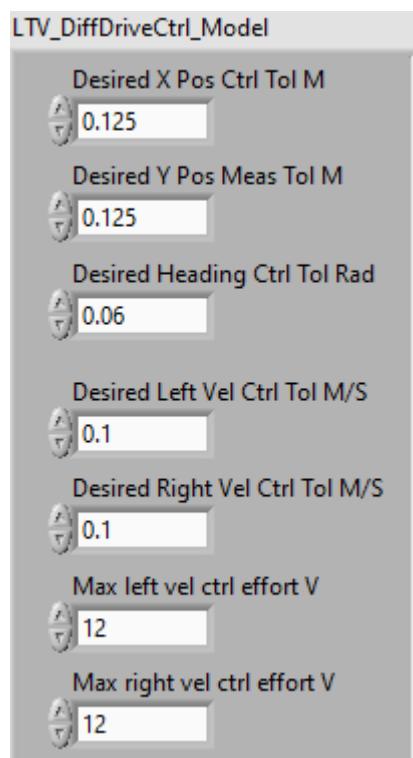
TypeDef-LTV_DIFF_DRIVE_CTRL_CONTROL_PARAMS



Cluster contains packed control values for LTV Diff Drive Ctrl execute function

Contains:

- Desired X Pos Ctrl Tol -- double --- Maximum desired X position control tolerance (meters, Default: 0.125)
- Desired Y Pos Ctrl Tol -- double --- Maximum desired y position control tolerance (meters, Default: 0.125)
- Desired Heading Ctrl Tol -- double --- Maximum desired heading control tolerance (meters, Default: 0.06)
- Desired Left Vel Ctrl Tol -- double --- Maximum desired left velocity control tolerance (meters, Default: 0.1)
- Desired Right Vel Ctrl Tol -- double --- Maximum desired right velocity control tolerance (meters, Default: 0.1)
- Max left vel ctrl effort V -- double -- Maximum left motor dmd control effort (volts, default 12.0)
- Max right vel ctrl effort V -- double -- Maximum right motor dmd control effort (volts, default 12.0)



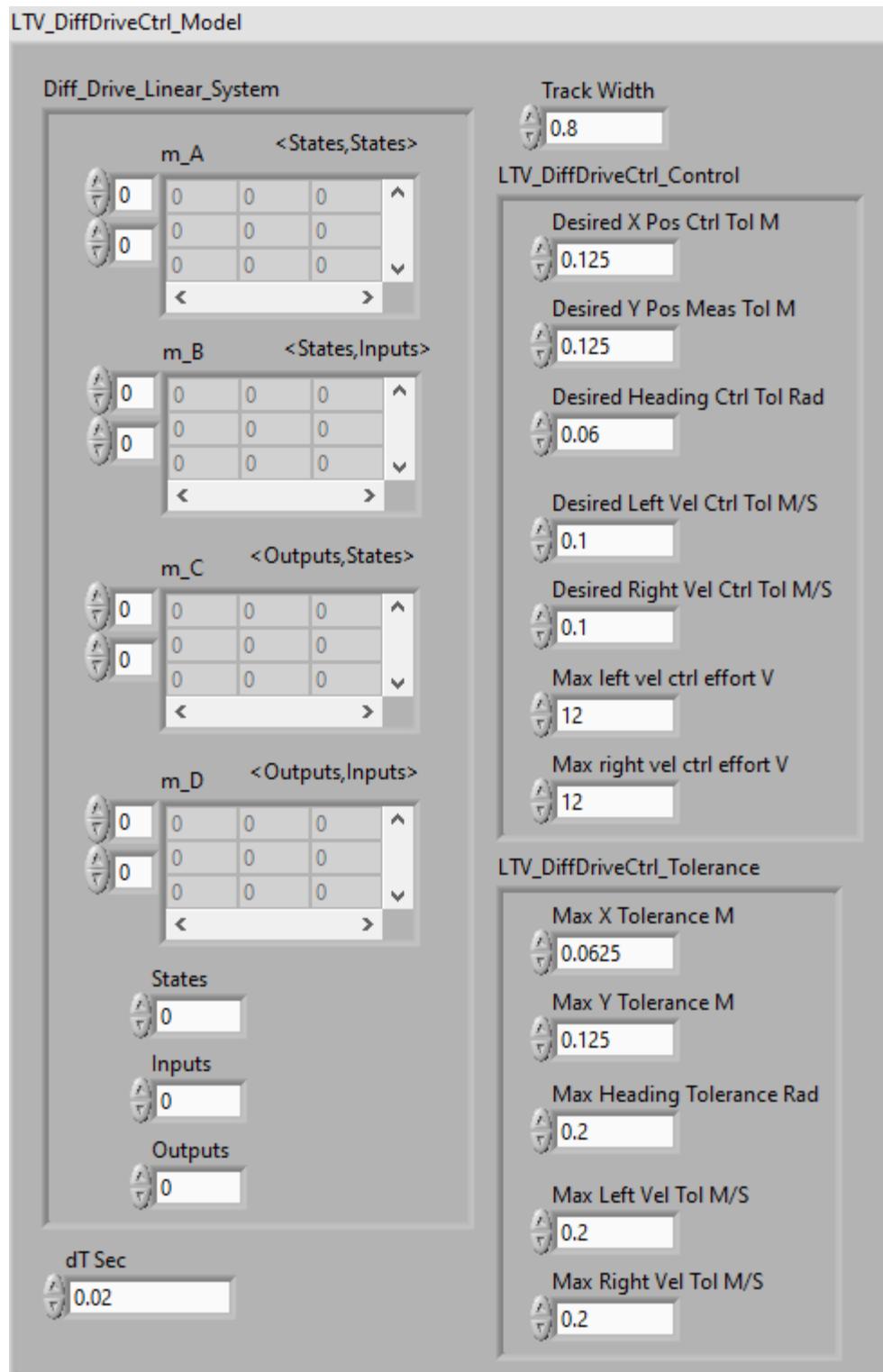
TypeDef-LTV_DIFF_DRIVE_CTRL_MODEL_PARAMS



Cluster contains model, control, and tolerance values for LTV Diff Drive control execute function:

Contains:

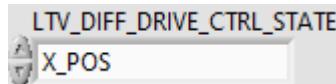
- Diff Drive Linear System -- Linear system -- Diff Drive linear system
- LTV Diff Drve Ctrl -- cluster -- Packed control parameters
- LTV Diff Drive Tol -- cluster -- Packed tolerance parameters
- Track Width -- double -- Track width (meters)
- dT Sec -- double -- Update time (Default 0.02)



TypeDef-LTV_DIFF_DRIVE_CTRL_STATE_ENUM



An enumerated variable containing names for the state indices of the LTV Differential Drive Train controller.



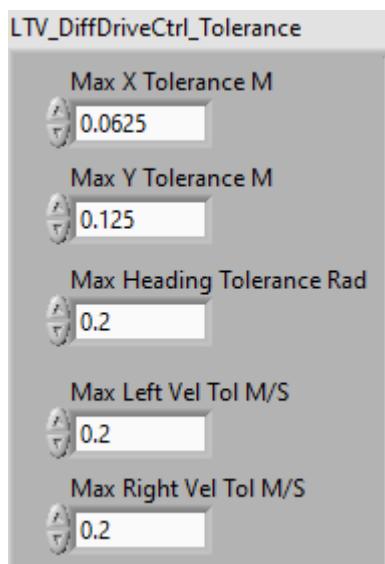
TypeDef-LTV_DIFF_DRIVE_CTRL_TOLERANCE



Contains tolerance values used to determine target for LTV Diff Drive Ctrl execute function

Contains:

- Max X tolerance -- double --- Maximum desired X position tolerance (meters, Default: 0.0625)
- Max Y tolerance -- double --- Maximum desired Y position tolerance (meters, Default 0.125)
- Max heading tolerance -- double --- Maximum desired heading tolerance (radians, Default 0.2)
- Max left vel tolerance -- double --- Maximum desired left velocity tolerance (meters/sec, Default 0.2)
- Max right vel tolerance -- double --- Maximum desired right velocity tolerance (meters/sec, Default: 0.2)



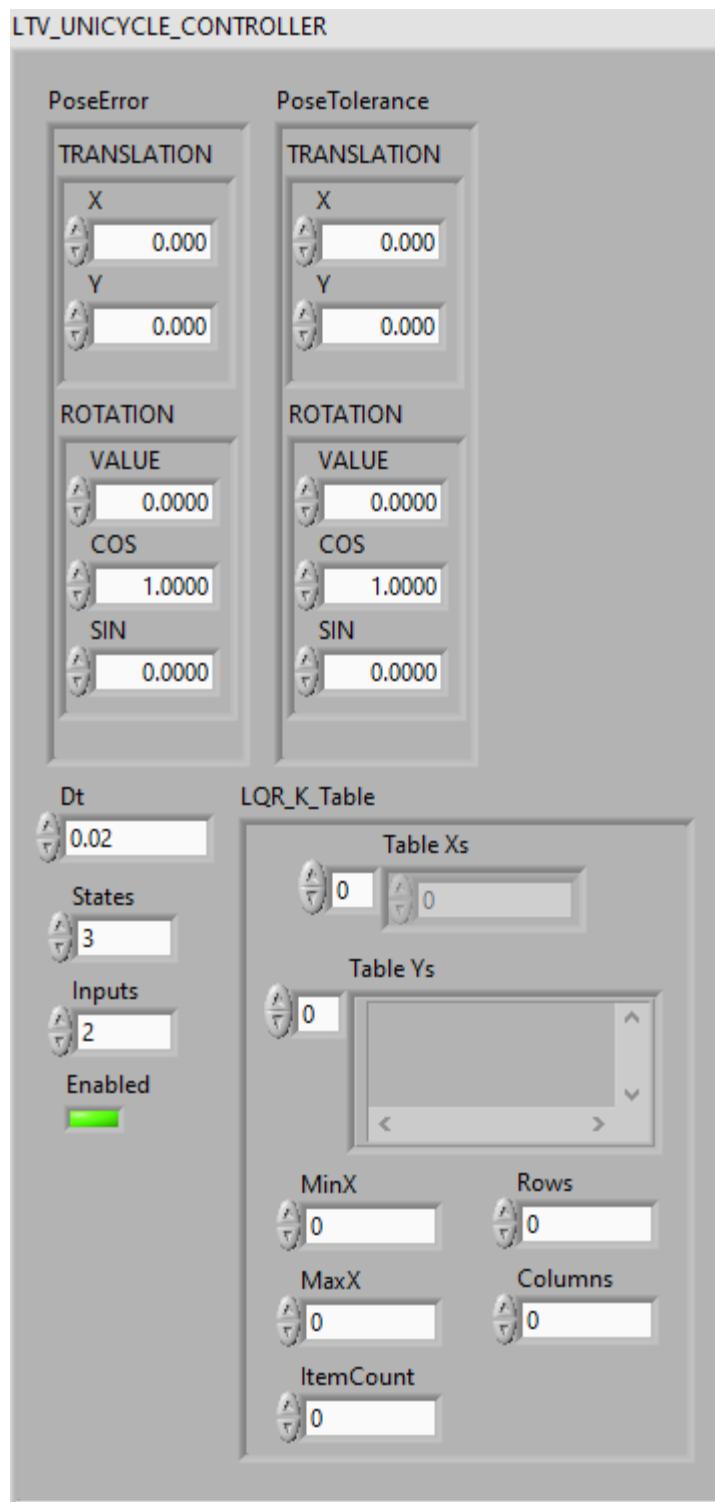
TypeDef-LTV_UNICYCLE_CONTROLLER



The linear time-varying unicycle controller has a similar form to the LQR, but the model used to compute the controller gain is the nonlinear model linearized around the drivetrain's current state.

Holds the data an LTV Unicycle Controller. The data cluster contains:

- PoseError
- PoseTolerance
- states
- inputs
- dT
- Enabled
- LQR K Table

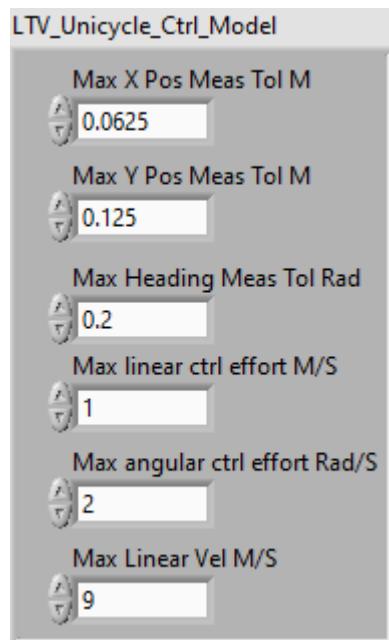


TypeDef-LTV_UNICYCLE_CONTROLLER_MODEL_PARAMS

Contains packed model and control values for LTV Unicycle Ctrl execute function

Inputs:

- Max X Pos Meas Tol -- double --- Maximum desired X position measurement tolerance (meters, Default: 0.0625)
- Max Y Pos Meas Tol -- double --- Maximum desired y position measurement tolerance (meters, Default: 0.125)
- Max Heading Meas Tol -- double --- Maximum desired heading measurement tolerance (meters, Default: 0.2)
- Max angular ctrl effort V -- double -- Maximum angular control effort (rad/sec, default 2.0)
- Max linear vel ctrl effort V -- double -- Maximum linearcontrol effort (meters/sec, default 9.0)



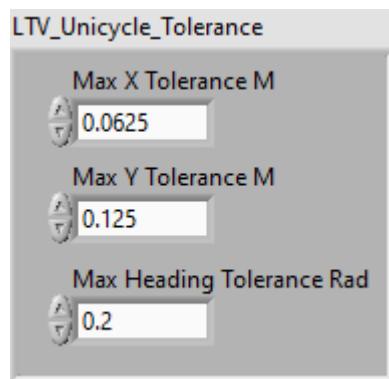
TypeDef-LTV_UNICYCLE_CONTROLLER_TOLERANCE



Cluster contains on target tolerance values for LTV Unicycle Ctrl execute function

Contains:

- Max X tolerance -- double --- Maximum desired X position tolerance (meters, Default: 0.0625)
- Max Y tolerance -- double --- Maximum desired Y position tolerance (meters, Default 0.125)
- Max heading tolerance -- double --- Maximum desired heading tolerance (radians, Default 0.2)



TypeDef-MECA_DRIVE_KINEMATICS



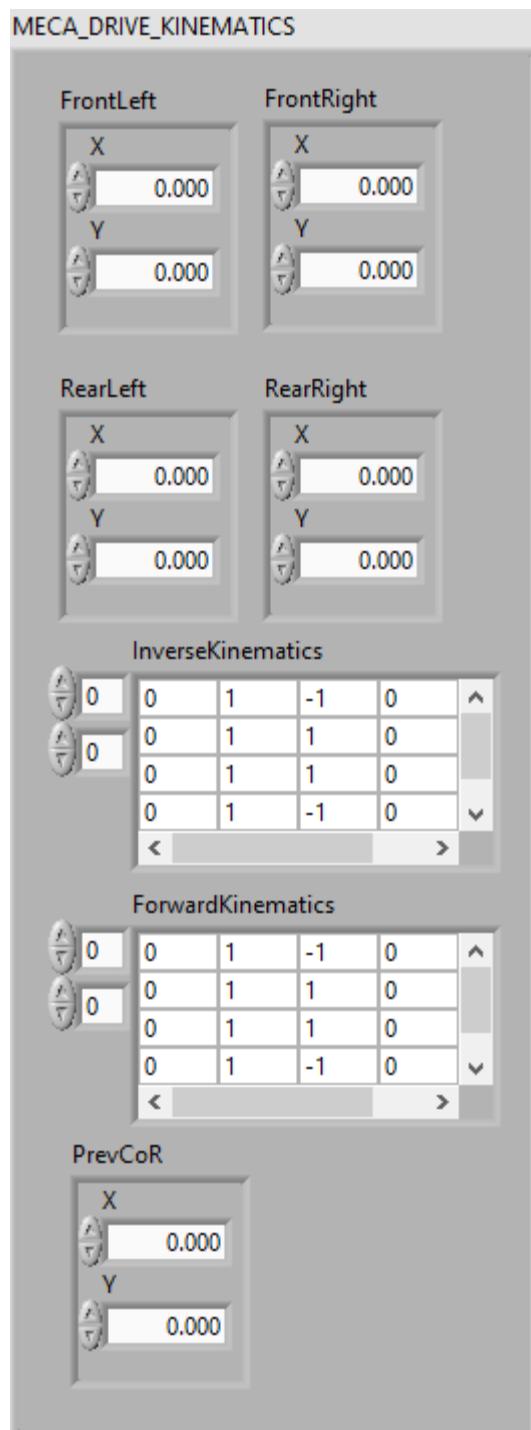
Helper class that converts a chassis velocity (dx , dy , and $d\theta$ components) into individual wheel speeds.

The inverse kinematics (converting from a desired chassis velocity to individual wheel speeds) uses the relative locations of the wheels with respect to the center of rotation. The center of rotation for inverse kinematics is also variable. This means that you can set your center of rotation in a corner of the robot to perform special evasion maneuvers.

Forward kinematics (converting an array of wheel speeds into the overall chassis motion) is performs the exact opposite of what inverse kinematics does. Since this is an overdetermined system (more equations than variables), we use a least-squares approximation.

The inverse kinematics: $[wheelSpeeds] = [wheelLocations][chassisSpeeds]$ We take the Moore-Penrose pseudoinverse of $[wheelLocations]$ and then multiply by $[wheelSpeeds]$ to get our chassis speeds.

Forward kinematics is also used for odometry -- determining the position of the robot on the field using encoders and a gyro.



TypeDef-MECA_DRIVE_ODOMETRY

Class for mecanum drive odometry. Odometry allows you to track the robot's position on the field over a course of a match using readings from your mecanum wheel encoders.

Teams can use odometry during the autonomous period for complex tasks like path following. Furthermore, odometry can be used for latency compensation when using computer-vision systems.

MECA_DRIVE_ODOMETRY

POSE

TRANSLATION

X	0.000
Y	0.000
Z	0.000

GyroOffset

VALUE	0.0000
COS	1.0000
SIN	0.0000

Kinematics

FrontLeft

X	0.000
Y	0.000
Z	0.000

FrontRight

X	0.000
Y	0.000
Z	0.000

RearLeft

X	0.000
Y	0.000
Z	0.000

RearRight

X	0.000
Y	0.000
Z	0.000

PreviousAngle

VALUE	0.0000
COS	1.0000
SIN	0.0000

InverseKinematics

FrontLeft	0	0	1	-1	0
FrontRight	0	0	1	1	0
RearLeft	0	0	1	1	0
RearRight	0	0	1	-1	0

ForwardKinematics

FrontLeft	0	0	1	-1	0
FrontRight	0	0	1	1	0
RearLeft	0	0	1	1	0
RearRight	0	0	1	-1	0

PreviousWheelPositions

FrontLeft

0.000

FrontRight

0.000

RearLeft

0.000

RearRight

0.000

PrevCoR

X	0.000
Y	0.000

TypeDef-MECA_DRIVE_POSE_EST



This data cluster and its associated function blocks wrap an UnscentedKalmanFilter Unscented Kalman Filter to fuse latency-compensated vision measurements with mecanum drive encoder velocity measurements. It will correct for noisy measurements and encoder drift. It is intended to be an easy but more accurate drop-in for MecanumDriveOdometry.

MecanumDrivePoseEstimator_update should be called every robot loop. If your loops are faster or slower than the default of 20 ms, then you should change the nominal delta time using the secondary constructor: MecanumDrivePoseEstimator_MecanumDrivePoseEstimator(Rotation2d, Pose2d, MecanumDriveWheelPositions, MecanumDriveKinematics, Matrix, Matrix, Matrix, double).

MecanumDrivePoseEstimator_addVisionMeasurement can be called as infrequently as you want; if you never call it, then this data cluster will behave mostly like regular encoder odometry.

The state-space system used internally has the following states (x), inputs (u), and outputs (y):

$$x = [x, y, \theta, s_{fl}, s_{fr}, s_{rl}, s_{rr}]^T$$

in the field coordinate system containing x position, y position, and heading, followed by the distance driven by the front left, front right, rear left, and rear right wheels.

$$u = [v_x, v_y, \omega, v_{fl}, v_{fr}, v_{rl}, v_{rr}]^T$$

containing x velocity, y velocity, and angular rate in the field coordinate system, followed by the velocity of the front left, front right, rear left, and rear right wheels.

$$y = [x, y, \theta]^T$$

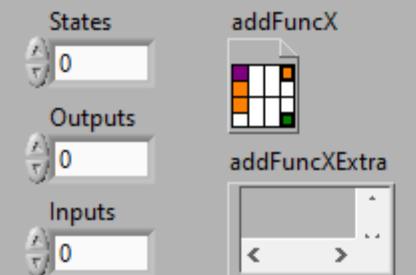
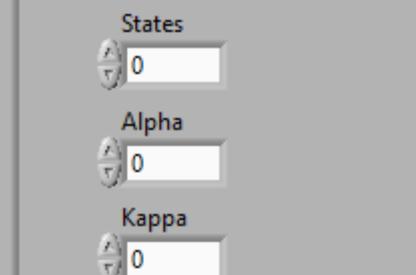
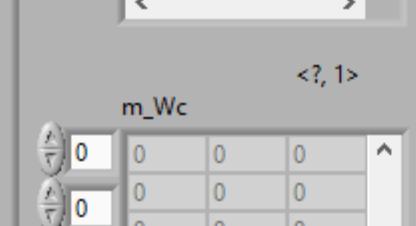
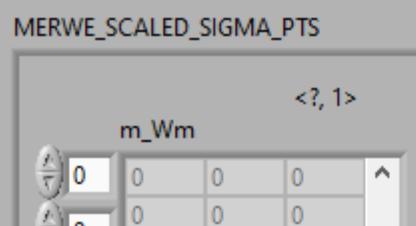
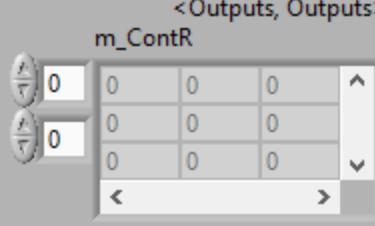
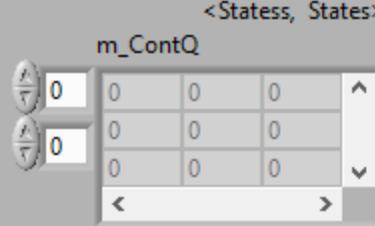
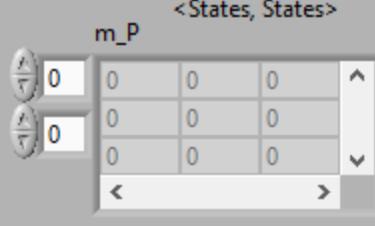
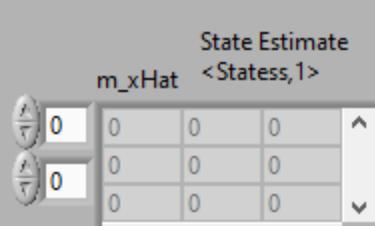
from vision containing x position, y position, and heading; or

$$y = [\theta, s_{fl}, s_{fr}, s_{rl}, s_{rr}]^T$$

containing gyro heading, followed by the distance driven by the front left, front right, rear left, and rear right wheels.

MECA_DRIVE_POSE_EST

m_Observer



m_dT_Sec
 0

meanFuncX

MeanFuncXExtra

meanFuncY

MeanFuncYExtra

residualFuncX

residualFuncXExtra

residualFuncY

residualFuncYExtra

M_VisionContR $\langle 3, 3 \rangle$

m_nominal_dT
 0

m_PrevTime_Sec
 -1

m_GyroOffset
VALUE
 0.0000
COS
 1.0000
SIN
 0.0000

m_PrevAngle
VALUE
 0.0000
COS
 1.0000
SIN
 0.0000

TypeDef-MECA_DRIVE_POSE_EST2



This set of functions wraps MecaDriveOdometry to fuse latency-compensated vision measurements with mecanum drive encoder distance measurements. It will correct for noisy measurements and encoder drift. It is intended to be a drop-in replacement MecaDriveOdometry.

MecaDrivePoseEst2_update should be called every robot loop.

MecaDrivePoseEst2_addVisionMeasurement can be called as infrequently as you want; if you never call it, then this class will behave mostly like regular encoder odometry.

This data cluster contains:

- Meca_Drive_Odom -- Meca_Drive__Odom -- Odometry data cluster for this drive.
- M_Q -- <3,1> Matrix -- Matrix that holds standard deviations for the robot pose X, Y, theta (meters, meters, radians)
- M_Vision_K -- <3,3> Matrix -- Matrix that holds the gain matrix for closed loop continuous Kalman filter.
- Pose_Buffer -- TimeInterpBufferVariant -- Holds last 1.5 seconds of position measurements.

MECA_DRIVE_POSE_EST2

Meca_Drive_Odom

POSE	GyroOffset	Kinematics	
TRANSLATION	VALUE	FrontLeft	FrontRight
X	0.000	X	0.000
Y	0.000	Y	0.000
Z	0.000	Y	0.000
ROTATION	PreviousAngle	RearLeft	RearRight
VALUE	VALUE	X	0.000
COS	0.000	COS	0.000
SIN	1.0000	SIN	1.0000
Z	0.0000	Z	0.0000

PreviousWheelPositions

FrontLeft	ForwardKinematics
0.000	0 1 -1 0
FrontRight	0 1 1 0
0.000	0 1 1 0
RearLeft	0 1 -1 0
0.000	0 1 1 0
RearRight	0 1 -1 0
0.000	0 1 1 0

InverseKinematics

0	0	1	-1	0
0	0	1	1	0
0	0	1	1	0
0	0	1	-1	0

PrevCoR

X	0.000
Y	0.000

M_Q <3, 1>

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

M_Vision_K <3, 3>

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Pose_Buffer

TimeStamps	
0	0

Min_TimeStamp

0

Max_TimeStamp

0

ItemCount

0

MaxBufferTime

0

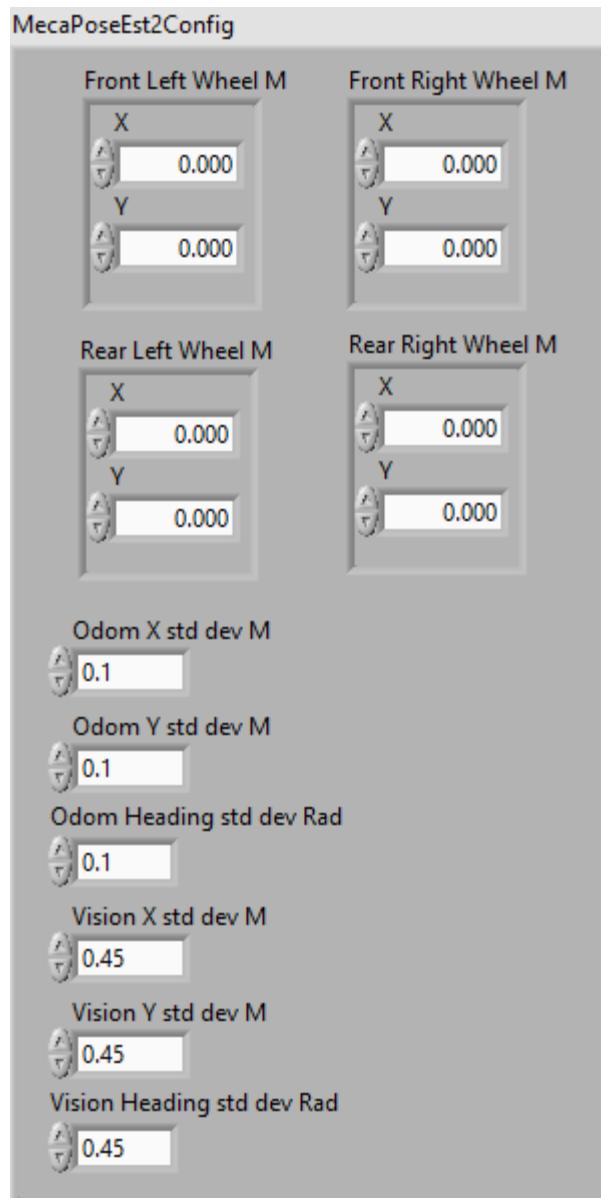
TypeDef-MECA_DRIVE_POSE_EST2_CONFIG



Cluster containing data required by the MecaDrivePoseEst2_Execute function.

Contains:

- Front Left Wheel M -- Translation2d -- Locations front left wheel in relation to center of robot. (meters)
- Front Right Wheel M -- Translation2d -- Locations front right wheel in relation to center of robot. (meters)
- Rear Left Wheel M -- Translation2d -- Locations rear left wheel in relation to center of robot. (meters)
- Rear Right Wheel M -- Translation2d -- Locations rear right wheel in relation to center of robot. (meters)
- Odom X Std Dev M -- double -- Odometry X position standard deviation (Default: 0.02) (meters)
- Odom Y Std Dev M -- double -- Odometry Y position standard deviation (Default: 0.02) (meters)
- Odom Heading Std Dev M -- double -- Odometry Heading (gyro) position standard deviation (Default: 0.01) (radians)
- Vision X Std Dev M -- double -- Vision X position standard deviation (Default: 0.1) (meters)
- Vision Y Std Dev M -- double -- Vision Y position standard deviation (Default: 0.1) (meters)
- Vision Heading Std Dev M -- double -- Vision Heading (gyro) position standard deviation (Default: 0.05) (radians)



TypeDef-MECA_DRIVE_POSE_EST2_INTERP_RECORD

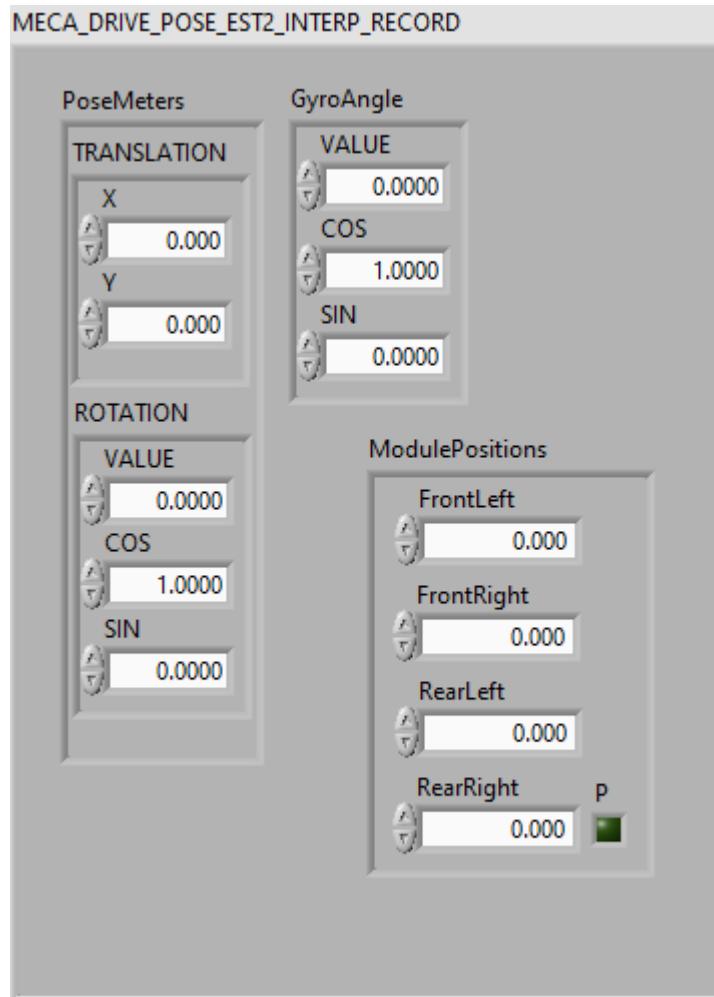


Differential Drive Pose Estimate 2 -- Interporatable Record

Represents an odometry record. The record contains the inputs provided as well as the pose that was observed based on these inputs, as well as the previous record and its inputs.

Contains:

- PoseMeters -- Pose2d -- The pose observed given the current sensor inputs and the previous pose.
- GyroAngle -- Rotation2d -- The current gyro angle.
- WheelPositions -- MecaWheelPositions -- The distances traveled by each wheel encoder.



TypeDef-MECA_WHEEL_POSITIONS



Represents the mecanum drive wheel positions. Cluster contains:

- frontLeft -- double -- Distance measured by the front left wheel. (Meters)

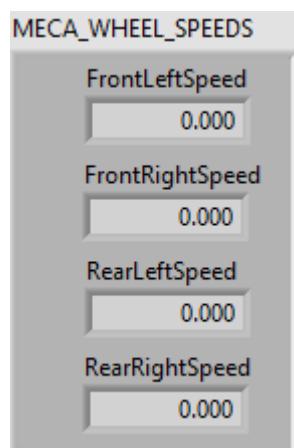
- frontRight -- double -- Distance measured by the front right wheel. (Meters)
- rearLeft -- double -- Distance measured by the rear left wheel. (Meters)
- rearRight -- double -- Distance measured by the rear right wheel. (Meters)



TypeDef-MECA_WHEEL_SPEEDS



Represents the mecanum drive wheel speeds



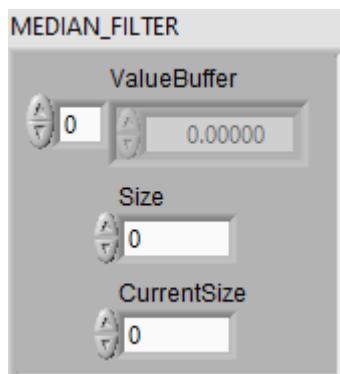
TypeDef-MEDIAN_FILTER



A cluster that implements a moving-window median filter. Useful for reducing measurement noise especially with processes that generate occasional, extreme outliers (such as values from vision processing, LIDAR, or ultrasonic sensors).

This cluster contains:

- m_valueBuffer -- Circular buffer of past values
- Size -- Number of values allowed in buffer. The number of samples in the moving window.
- CurrentSize -- Current number of values in buffer;



TypeDef-MERWE_SCALED_SIGMA PTS



Generates sigma points and weights according to Van der Merwe's 2004 dissertation[1] for the UnscentedKalmanFilter function.

It parametrizes the sigma points using alpha, beta, kappa terms, and is the version seen in most publications. Unless you know better, this should be your default choice.

States is the dimensionality of the state. 2*States+1 weights will be generated.

[1] R. Van der Merwe "Sigma-Point Kalman Filters for Probabilistic Inference in Dynamic State-Space Models" (Doctoral dissertation)

This cluster contains:

- m_Wm --

- m_Wc --

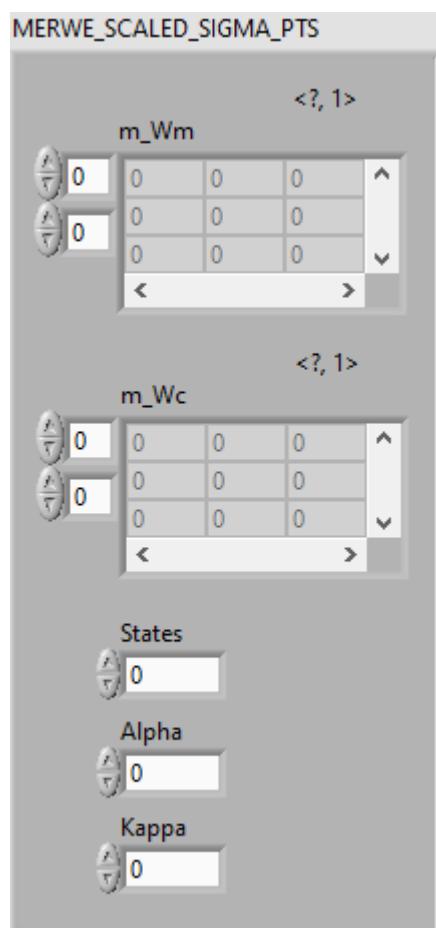
- States --

- Alpha -- Determines the spread of the sigma points around the mean. Usually a small positive value (Default: 1.0e-3).

- Kappa -- Secondary scaling parameter usually set to 0 or 3 - States (Default: 2)

Note:

- Beta is not included in the data, it is calculated or passed as a parameter. (Default: 3 - States)



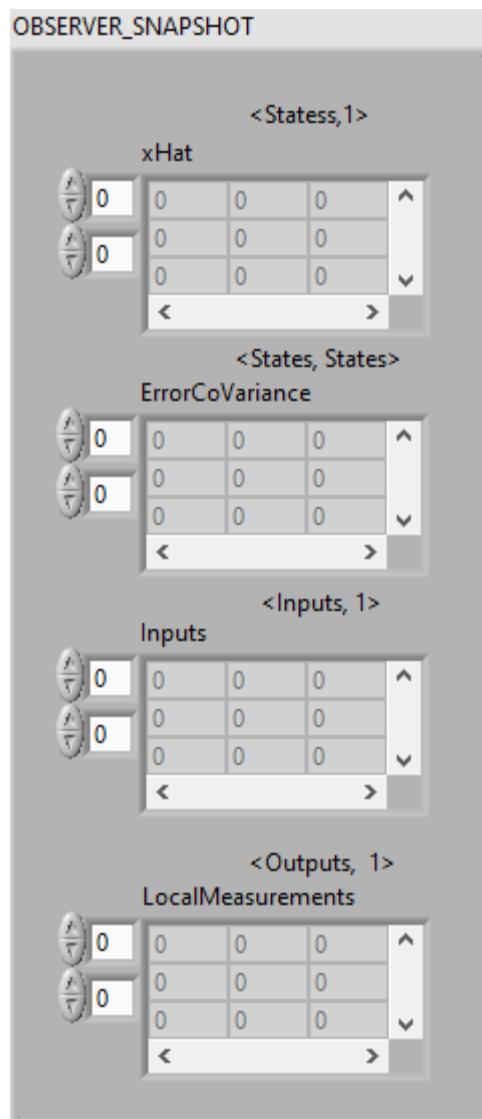
TypeDef-OBSERVER_SNAPSHOT



Cluster containing the data for a single snap shot of the Kalman Filter Latency Compensator.

The cluster contains:

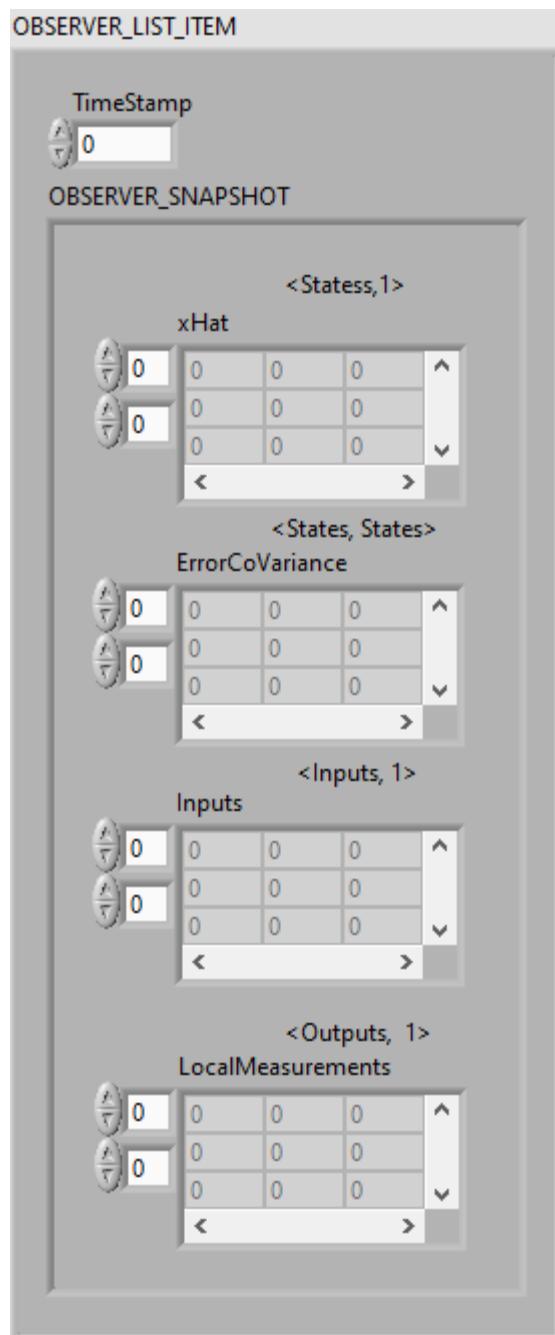
- xHat
- ErrorCoVariance
- Inputs
- LocalMeasurements



TypeDef-OBSERVER_SNAP_LIST_ITEM



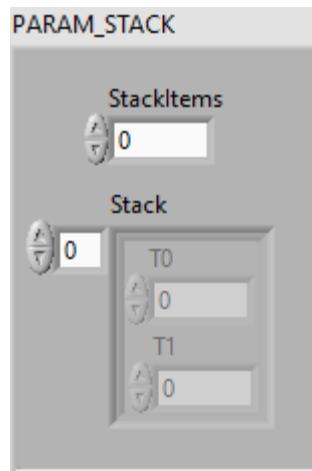
Observer List Item cluster contains a timestamped Observer Snapshot used by the Kalman Filter Latency Compensator.



TypeDef-PARAM_STACK

PARAM
STACK

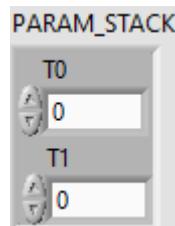
Stack (Array) of data used by SplineParam_Spline_T0_T1.



TypeDef-PARAM_STACK_ITEM



Stack data item used by SplineParam_Spline_T0_T1.



TypeDef-PID_ADV_LIMITS

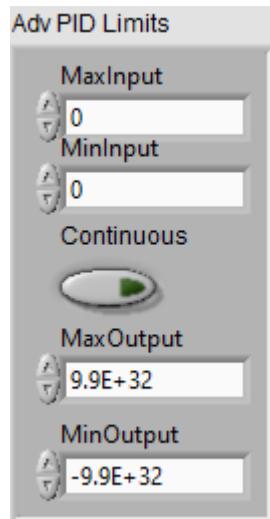


Cluster containing the limits for the Advanced PID controller.

This cluster contains:

- Max_Input -- If Continuous, this defines the maximum input value. (Going above this wraps to the Min_Input.)
- Min_Input -- If Continuous, this defines the minimum input value. (Going below this wraps to the Max_Input.)

- Continuous -- Boolean indicating that the measurement value wraps around (absoulte encoder or gyro)
- Max_Output -- Maximum allowed output (Used when applying integral windup protection.)
- Min_Output -- Minimum allowed output (Used when applying integral windup protection.)



TypeDef-PID_ADV_TUNING

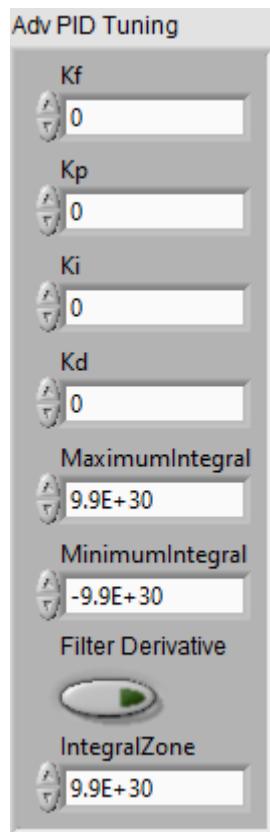


This cluster contains the tuning parameters used by the Advanced PID controller.

This cluster contains:

- Kf -- "Feedforward" control tuning constant
- Kp -- "Proportional" control tuning constant
- Ki -- "Integral" control tuning constant (dt is seconds)
- Kd -- "Derivative" control tuning constant (dt is seconds)
- Max_Integral -- Maximum allowed integral value (Default 1.0)
- Min_Integral -- Minimum allowed integral value (Default -1.0)
- DerivativeFilter -- Boolean indicating that derivative error value filtering is desired.
- IntegralZone -- Integration is allowed when absolute value of error is less than or

equal to this value. (Default: 9.9E30)



TypeDef-PID_CONTROLLER



This implements a non-interacting PID where the Kp, Ki, Kd constants do not interact with each other (as is the case with a classical PID implementation). Tuning schemes applicable to "classical" PIDs will need to consider the independence of the K values when applying new tuning.

This cluster contains:

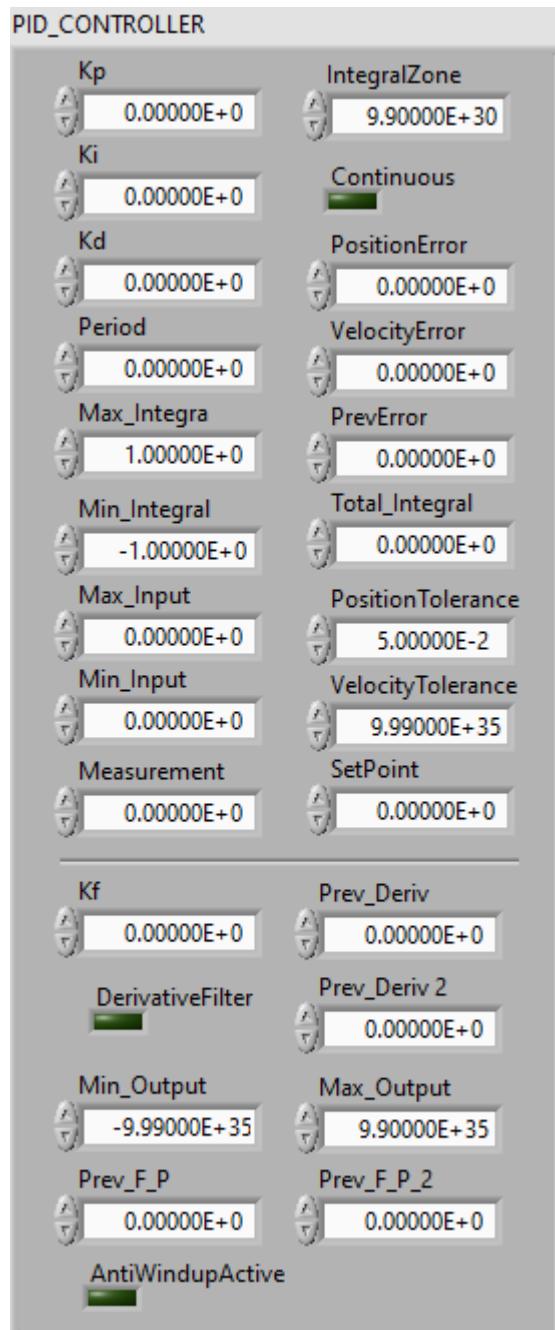
- Kp -- "Proportional" control tuning constant
- Ki -- "Integral" control tuning constant (dt is seconds)
- Kd -- "Derivative" control tuning constant (dt is seconds)
- Period -- The period (in seconds) of the loop that calls the controller. (Default 0.020 seconds)

- Max_Integral -- Maximum allowed integral value (Default 1.0)
- Min_Integral -- Minimum allowed integral value (Default -1.0)
- Continuous -- Boolean indicating that the measurement value wraps around (absoulte encoder or gyro)
- Max_Input -- If Continuous, this defines the maximum input value. (Going above this wraps to the Min_Input.)
- Min_Input -- If Continuous, this defins the minimum input value. (Going below this wraps to the Max_Input.)
- Measurement -- Current measurement (Process Variable or PV)
- Setpoint -- Current setpoint (SP)
- PositionError -- The error at the time of the most recent call to calculate()
- VelocityError -- The current velocity error. Used when calcuating on setpoint.
- PrevError -- The error at the time of the second-most-recent call to calculate() (used to compute velocity)
- TotalError -- The sum of the errors for use in the integral calc. (Note: This sum is prior to applying Ki, causing an discontinuity in the output when performing online tuning.)
- PositionTolerance -- The position error that is considered at setpoint. (Not used in control.) (Default: 0.05)
- VelocityTolerance -- The velocity error that is considered at setpoint. (Not used in control.) (Default: Infinity)

These values are used with the Advanced PID

- Kf -- "Feedforward" control tuning constant
- DerivativeFilter -- Boolean indicating that derivative error value filtering is desired.
- Min_Output -- Minimum allowed output (Used when applying integral windup protection.)
- Max_output -- Maximum allowed output (Used when applying integral windup protection.)
- Prev_Deriv -- Internal value.
- Prev_Deriv2 -- Internal value.
- Prev_F_P -- Internal value
- Prev_F_P2 -- Internal value.

- AntiWindupActive -- Boolean indicating that integral anti-windup protection is active (limiting the integral term)



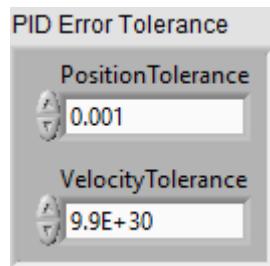
TypeDef-PID_ERROR_TOLERANCE



This cluster contains the velocity and position tolerance for determining AtSetpoint.

This cluster contains:

- PositionTolerance -- The position error that is considered at setpoint. (Not used in control.) (Default: 0.05)
- VelocityTolerance -- The velocity error that is considered at setpoint. (Not used in control.) (Default: Infinity)



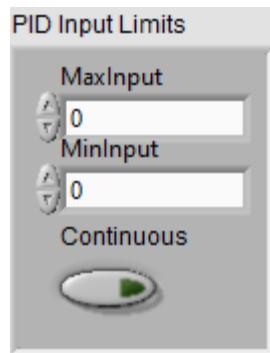
TypeDef-PID_INPUT_LIMITS



This cluster contains the input limit parameters for the "normal" PID controller

This cluster contains:

- Max_Input -- If Continuous, this defines the maximum input value. (Going above this wraps to the Min_Input.)
- Min_Input -- If Continuous, this defines the minimum input value. (Going below this wraps to the Max_Input.)
- Continuous -- Boolean indicating that the measurement value wraps around (absolute encoder or gyro)



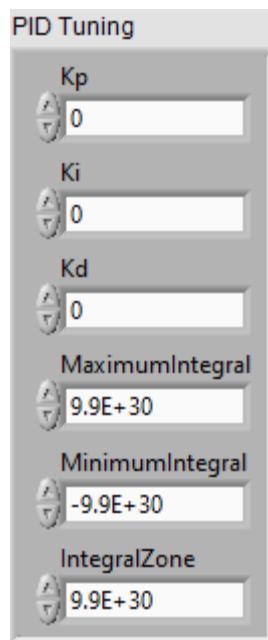
TypeDef-PID_TUNING



This cluster contains the tuning parameters for the "normal" PID controller.

This cluster contains:

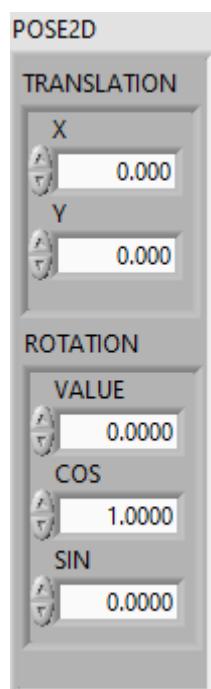
- Kp -- "Proportional" control tuning constant
- Ki -- "Integral" control tuning constant (dt is seconds)
- Kd -- "Derivative" control tuning constant (dt is seconds)
- Max_Integral -- Maximum allowed integral value (Default 1.0)
- Min_Integral -- Minimum allowed integral value (Default -1.0)



TypeDef-POSE2D



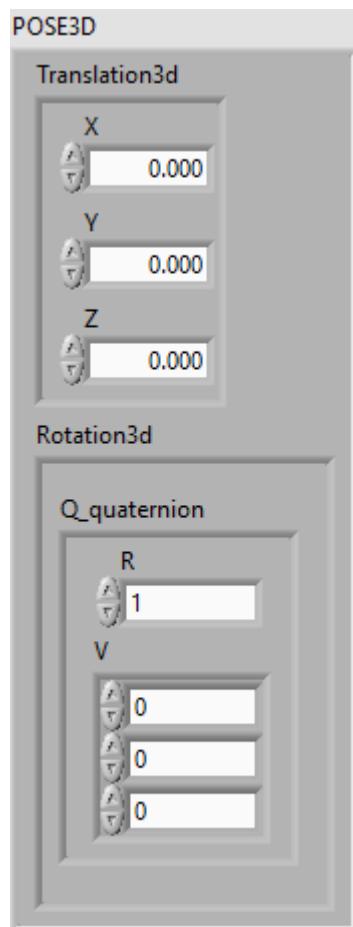
Represents a 2d pose containing translational and rotational elements.



TypeDef-POSE3D



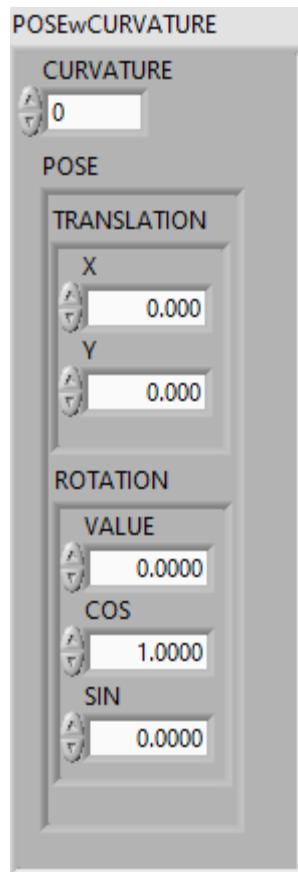
Represents a 3D pose containing translational and rotational elements.



TypeDef-POSEwCURVATURE



Represents a pair of a pose and a curvature.



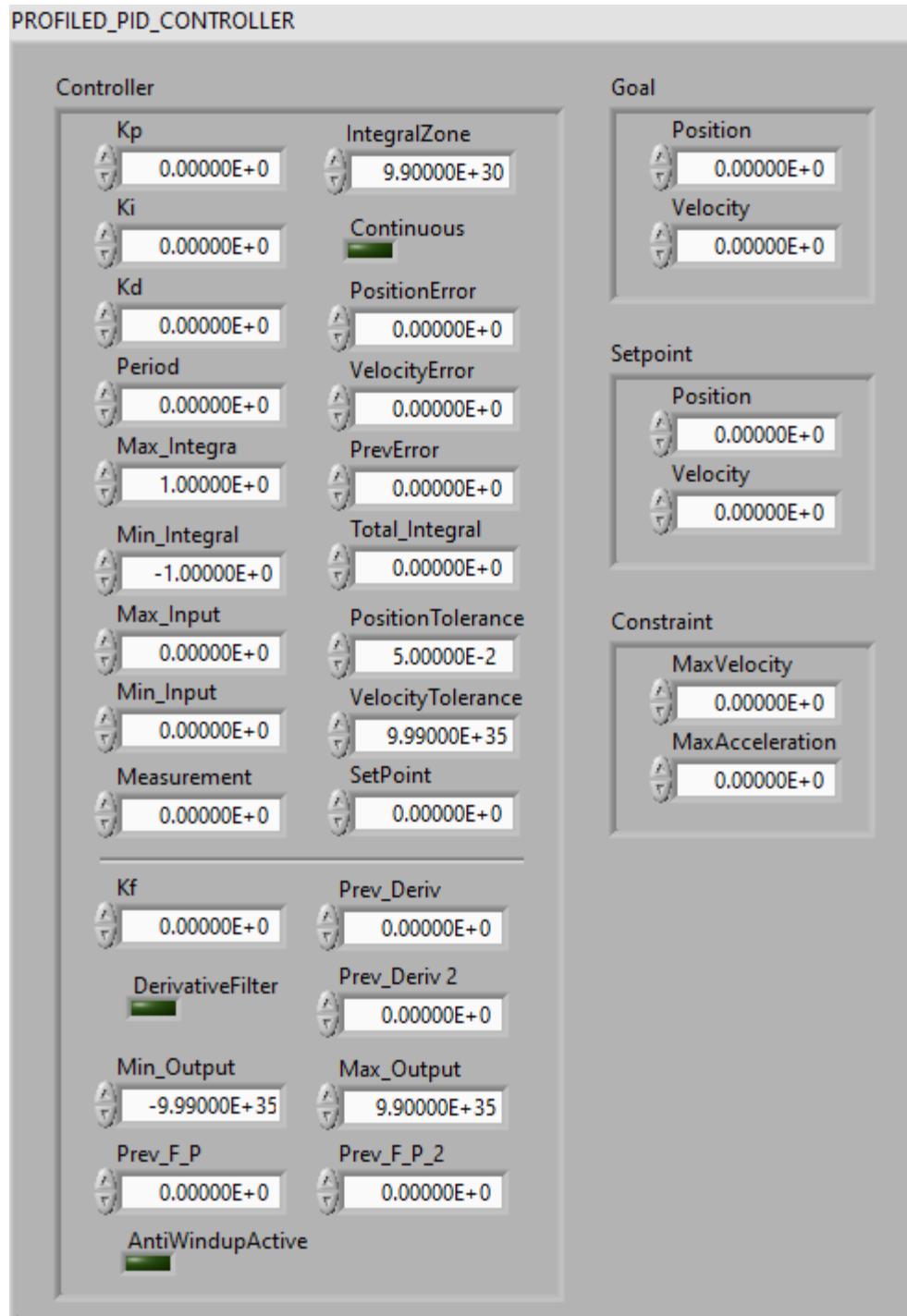
TypeDef-PROFILED_PID_CONTROLLER



Implements a PID control loop whose setpoint is constrained by a trapezoid profile. Users should call reset when they first start running the controller to avoid unwanted behavior.

This cluster contains:

- Controller -- PID controller data cluster
- Goal -- TrapezoidProfile containing the goal. (Velocity and Acceleration when setpoint is reached.)
- Setpoint -- TrapezoidProfile containing the setpoint.
- Constraints -- TrapezoidProfileConstraints containing the velocity and acceleration constraint.



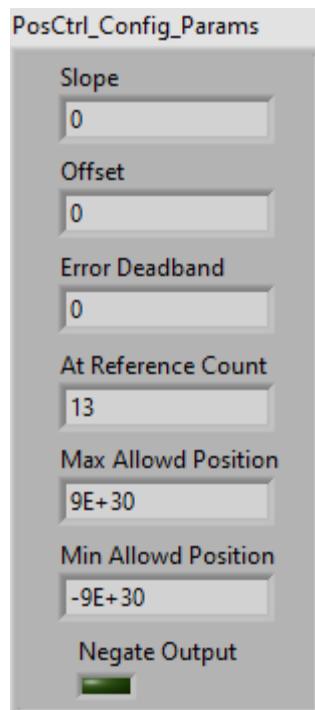
TypeDef-PosCtrl_Tuning_Type



Type definition for the configuration parameters used by the Position Control controller.

This cluster contains:

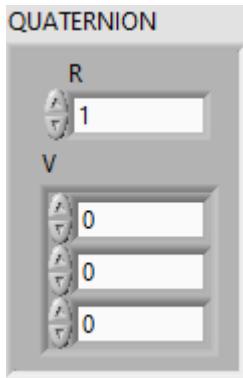
- Slope - double - Straight line slope for calculating the output when error outside the deadband
- Intercept - double - Straight line intercept for calculating the output.
- Error Deadband - double - Absolute value of the deadband. When the error is within the deadband the output is set to zero.
- At Reference Count - integer - Number of consecutive scan cycles the error must be within the deadband before the At Reference output is set TRUE.
- Max Allowed Position - double - Acts as a virtual limit switch. When position is above this value the output isn't allowed to be greater than zero.
- Min Allowed Position - double - Acts as a virtual limit switch. When position is below this value the output isn't allowed to be less than zero.
- Negate Output - boolean - When set the output is multiplied by negative one.



TypeDef-QUATERNION



Quaternion. One use of this is to represent a 3d Rotation.



TypeDef-RAMSETE



Ramsete is a nonlinear time-varying feedback controller for unicycle models that drives the model to a desired pose along a two-dimensional trajectory. Why would we need a nonlinear control law in addition to the linear ones we have used so far like PID? If we use the original approach with PID controllers for left and right position and velocity states, the controllers only deal with the local pose. If the robot deviates from the path, there is no way for the controllers to correct and the robot may not reach the desired global pose. This is due to multiple endpoints existing for the robot which have the same encoder path arc lengths.

Instead of using wheel path arc lengths (which are in the robot's local coordinate frame), nonlinear controllers like pure pursuit and Ramsete use global pose. The controller uses this extra information to guide a linear reference tracker like the PID controllers back in by adjusting the references of the PID controllers.

The paper "Control of Wheeled Mobile Robots: An Experimental Overview" describes a nonlinear controller for a wheeled vehicle with unicycle-like kinematics; a global pose consisting of x, y, and theta; and a desired pose consisting of x_d, y_d, and theta_d. We call it Ramsete because that's the acronym for the title of the book it came from in Italian ("Robotica Articolata e Mobile per i SERVizi e le TEcnologie").

This cluster contains:

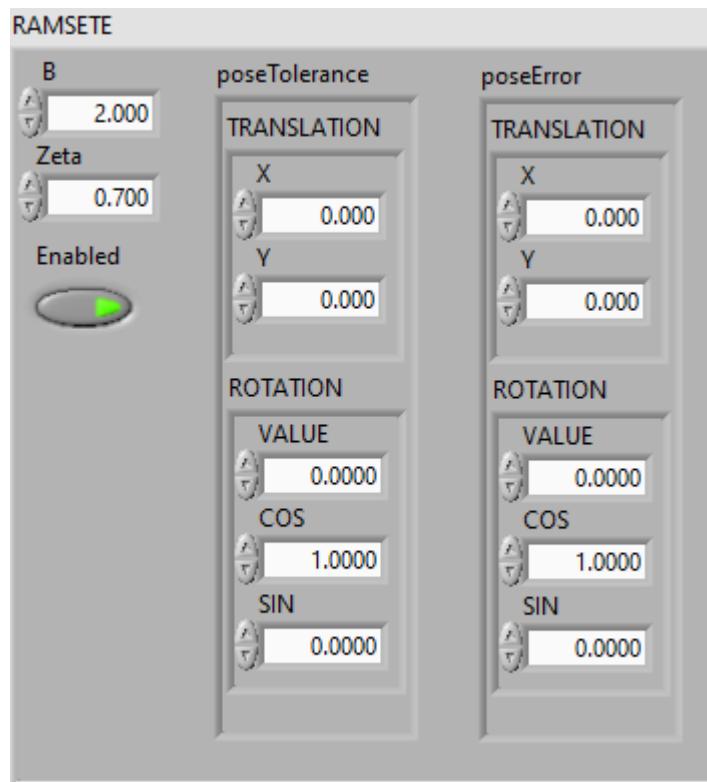
- B -- Tuning parameter ($b > 0 \text{ rad}^2/\text{m}^2$) for which larger values make convergence more aggressive like a proportional term. (Default 2.0)

-- Zeta -- Tuning parameter ($0 \text{ /Rad} < \text{zeta} < 1 \text{ /rad}$) for which larger values provide more damping in response. (Default 0.7)

-- Enabled -- When TRUE the closed loop Ramsete controller is used. When FALSE, input desired speeds are used directly (open loop).

-- Pose Tolerance -- A Pose containing the desired tolerance used to calculate On Target.

-- Pose Error -- Current position and heading error.



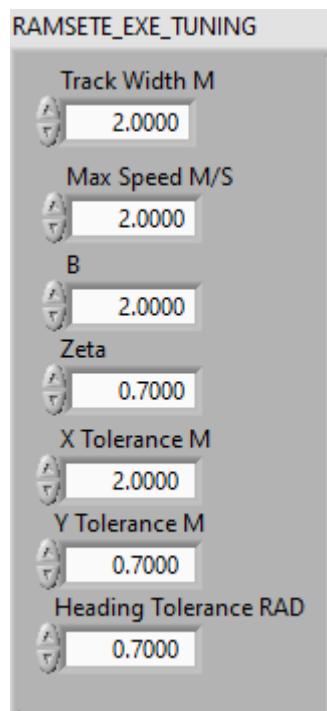
TypeDef-RAMSETE_EXE_TUNING



Tuning parameters for the Ramsete EXE function.

This cluster contains:

- Track Width M -- The effective track width (Meters). If the robot has a lot of "slippage", this value can be made greater than the actual physical track width to compensate.
 - Max Speed M/S -- The maximum drive wheel speed (Meters/Second)
 - B -- Tuning parameter ($b > 0 \text{ rad}^2/\text{m}^2$) for which larger values make convergence more aggressive like a proportional term. (Default 2.0)
 - Zeta -- Tuning parameter ($0 / \text{Rad} < zeta < 1 / \text{rad}$) for which larger values provide more damping in response. (Default 0.7)
-
- X Tolerance -- The X direction tolerance (Meters) used when calculating "on target" (Default 0.051)
 - Y Tolerance -- The Y direction tolerance (Meters) used when calculating "on target" (Default 0.051)
 - Heading Tolerance -- The rotational Heading tolerance (Radians) used when calculating "on target" (Default 0.035)

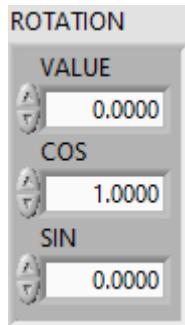


TypeDef-ROTATION2D



A rotation in a 2d coordinate frame represented a point on the unit circle (cosine and sine).

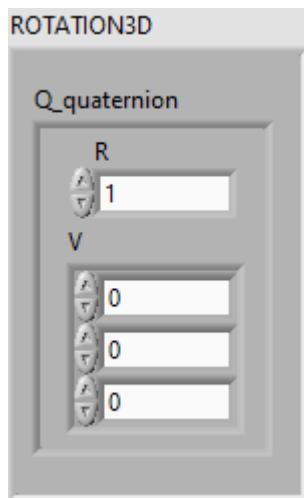
Default values are set for 0 degrees (0 radians).



TypeDef-ROTATION3D



A rotation in a 3D coordinate



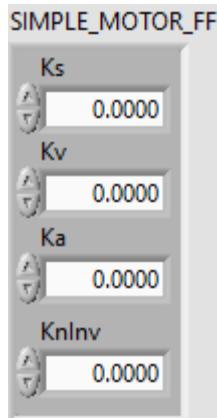
TypeDef-SIMPLE_MOTOR_FF



A helper class that computes feedforward outputs for a simple permanent-magnet DC motor.

Elements"

- Ks - The static gain.
- Kv - The velocity gain.
- Ka - The acceleration gain.
- KnInv -- Optional 1/Kn constant used with the Advanced PID. This helps to match the output units.



TypeDef-SIMPLE_MOTOR_FF_KA_TUNE_PARAMS

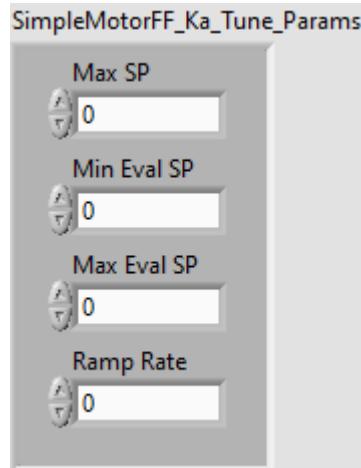


Cluster data structure used to pack parameters that define the test to be performed when calculating the Simple Motor Feedforward Ka value.

Elements:

- Max Sp -- Double -- The maximum value for the setpoint when ramping.
- Min Eval SP -- Double -- The lowest setpoint value to use when evaluating Ka
- Max Eval SP -- Double -- The highest setpoint value to use when evaluating Ka

-- Ramp Rate -- Double -- The rate at which to ramp the setpoint (Units/Sec)



TypeDef-SINGLE_JOINT_ARM_SIM



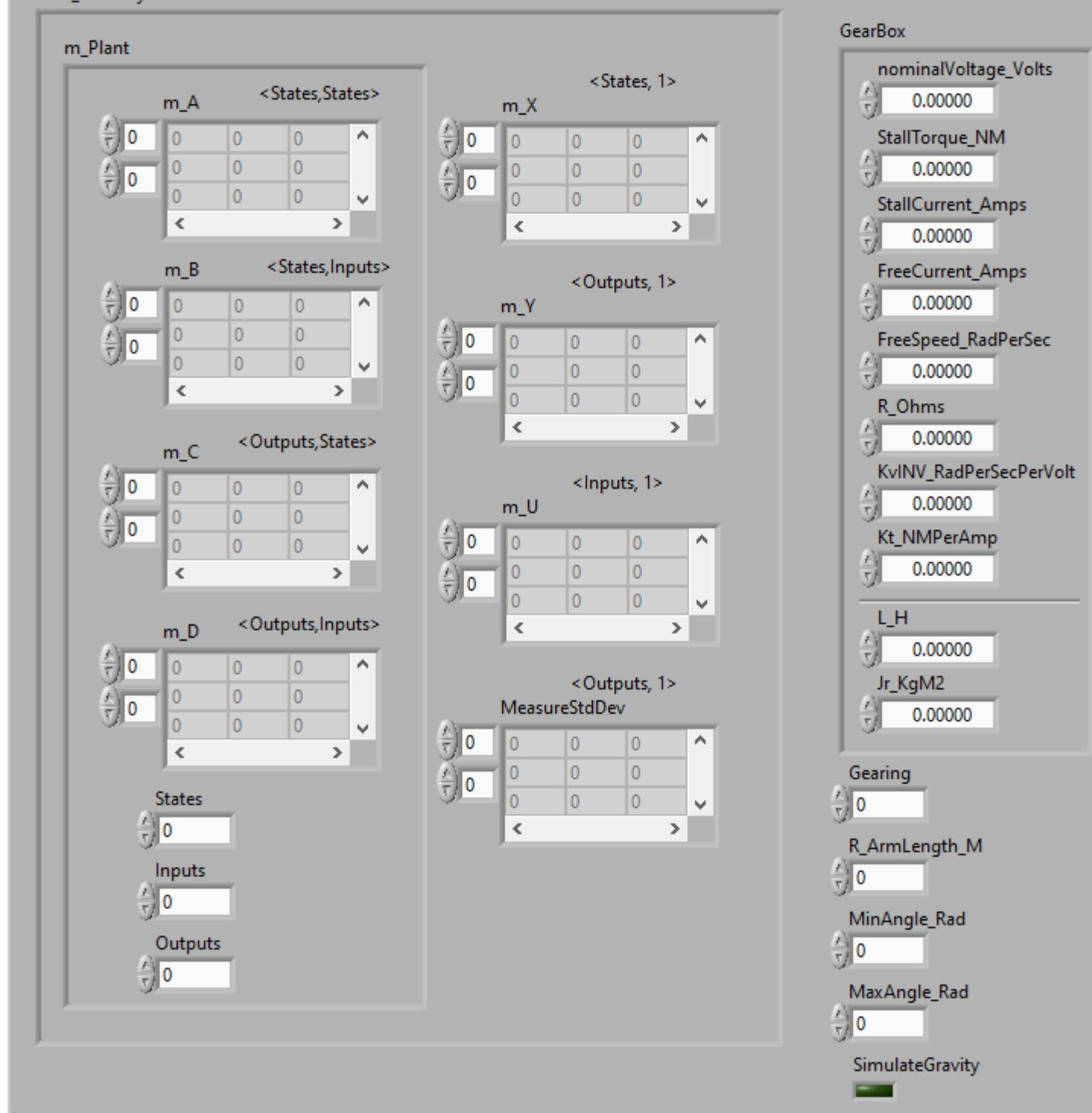
Represents a simulated single jointed arm mechanism.

This cluster contains:

- GearBox -- DCMotor cluster, the gearbox for the arm.
- Gearing -- The gearing between the motors and the output. (numbers greater than 1 represent reductions)
- R_ArmLength -- The length of the arm. (Meters)
- MinAngle -- The minimum angle that the arm is capable of (Radians)
- MaxAngle -- The maximum angle that the arm is capable of. (Radans)
- ArmMass -- The mass of the arm (Kilograms)
- SimulateGravity -- Boolean indicating whether the simulator should simulate gravity.

SINGLE_JOINT_ARM_SIM

Arm_LinearSystem



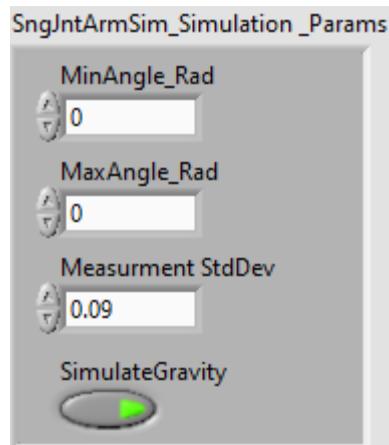
TypeDef-SINGLE_JOINT_ARM_SIMULATION_PARAMS



Cluster contains simulation parameters used by the Single Jointed Arm Simulation Execute routine.

Contains:

- Min Angle -- double -- Minimum physical arm angle (radians)
- Max Angle -- double -- Maximum physical arm angle (radians)
- Pos Sim Std Dev -- double -- Standard deviation for the arm position (radians Default: 0.09)
- SimulateGravity -- boolean -- Simulate gravity (Default: True)



TypeDef-SLEW_RATE_LIMITER

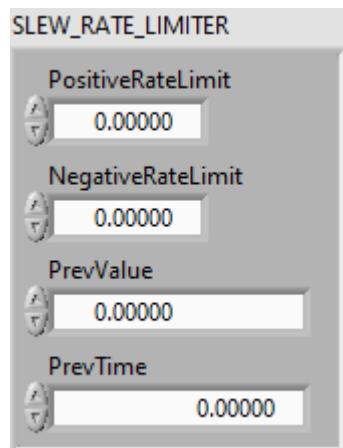


Data cluster for a set of functions that limits the rate of change of an input value. Useful for implementing voltage, setpoint, and/or output ramps. A slew-rate limit is most appropriate when the quantity being controlled is a velocity or a voltage; when controlling a position, consider using a TrapezoidProfile} instead.

Data:

- RateLimit -- Desired maximum rate THING/SEC
- PrevValue -- Previous value of THING

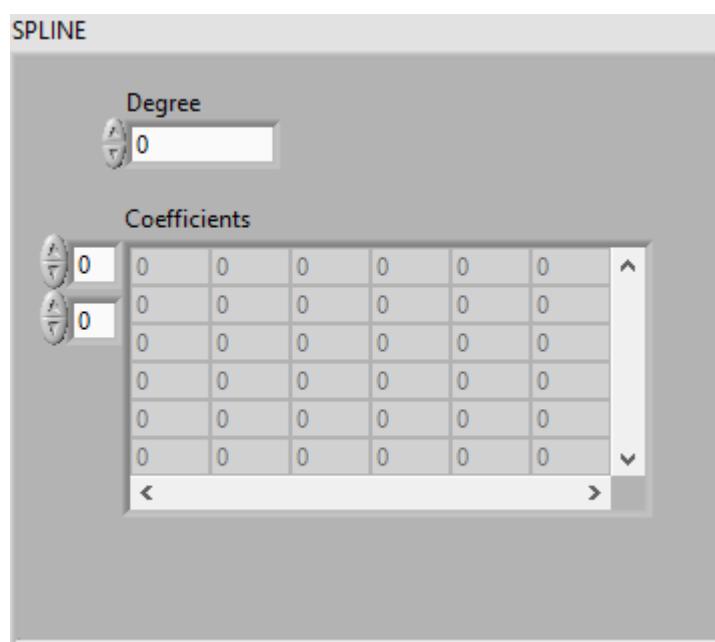
- PrevTime -- Time (Seconds) of last call when PrevValue was gathered.



TypeDef-SPLINE



Represents a two-dimensional parametric spline that interpolates between two points.

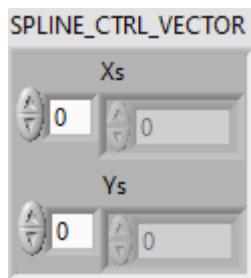


TypeDef-SPLINE_CTRL_VECTOR



Represents a control vector for a spline.

Each element in each array represents the value of the derivative at the index. For example, the value of $x[2]$ is the second derivative in the x dimension.



TypeDef-SWERVE_DRIVE_KINEMATICS



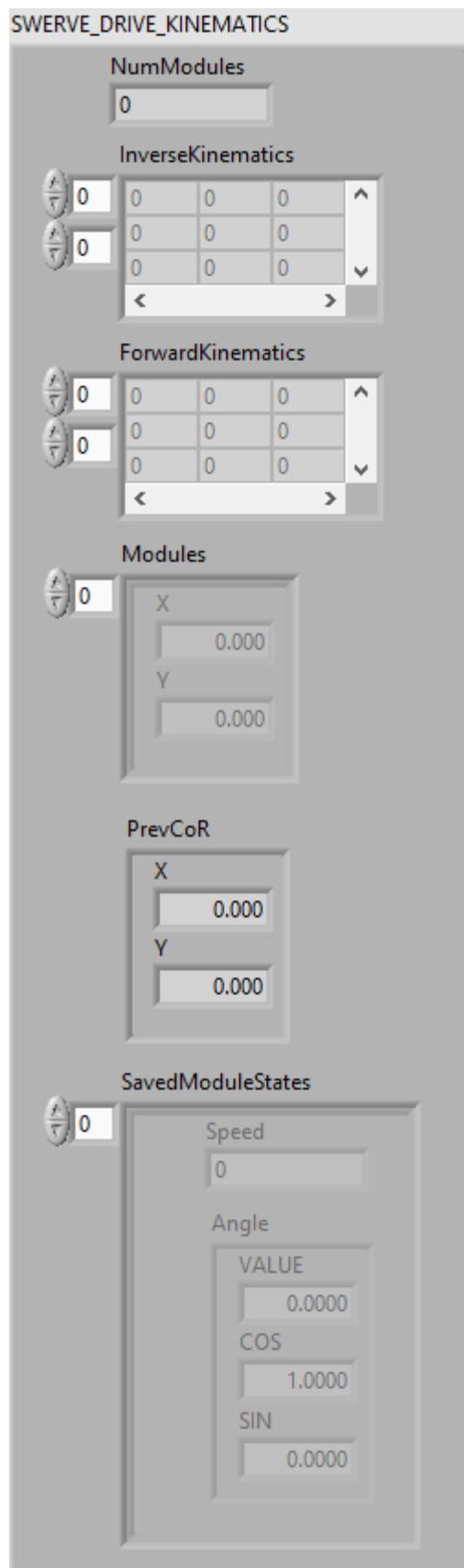
Helper class that converts a chassis velocity (dx , dy , and $d\theta$ components) into individual module states (speed and angle).

The inverse kinematics (converting from a desired chassis velocity to individual module states) uses the relative locations of the modules with respect to the center of rotation. The center of rotation for inverse kinematics is also variable. This means that you can set your center of rotation in a corner of the robot to perform special evasion manuevers.

Forward kinematics (converting an array of module states into the overall chassis motion) is performs the exact opposite of what inverse kinematics does. Since this is an overdetermined system (more equations than variables), we use a least-squares approximation.

The inverse kinematics: $[moduleStates] = [moduleLocations][chassisSpeeds]$ We take the Moore-Penrose pseudoinverse of $[moduleLocations]$ and then multiply by $[moduleStates]$ to get our chassis speeds.

Forward kinematics is also used for odometry -- determining the position of the robot on the field using encoders and a gyro.

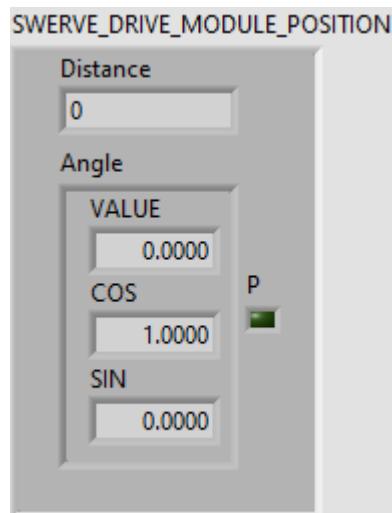


TypeDef-SWERVE_DRIVE_MODULE_POSITION



Represents the position of one swerve module. This cluster includes:

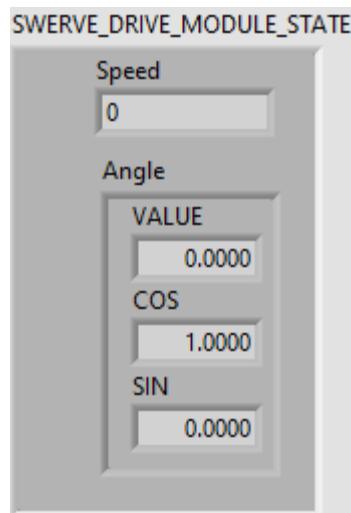
- Distance measured by the wheel of the module.
- Angle of the module. (Rotation2d)



TypeDef-SWERVE_DRIVE_MODULE_STATE



Represents the state of one swerve module.

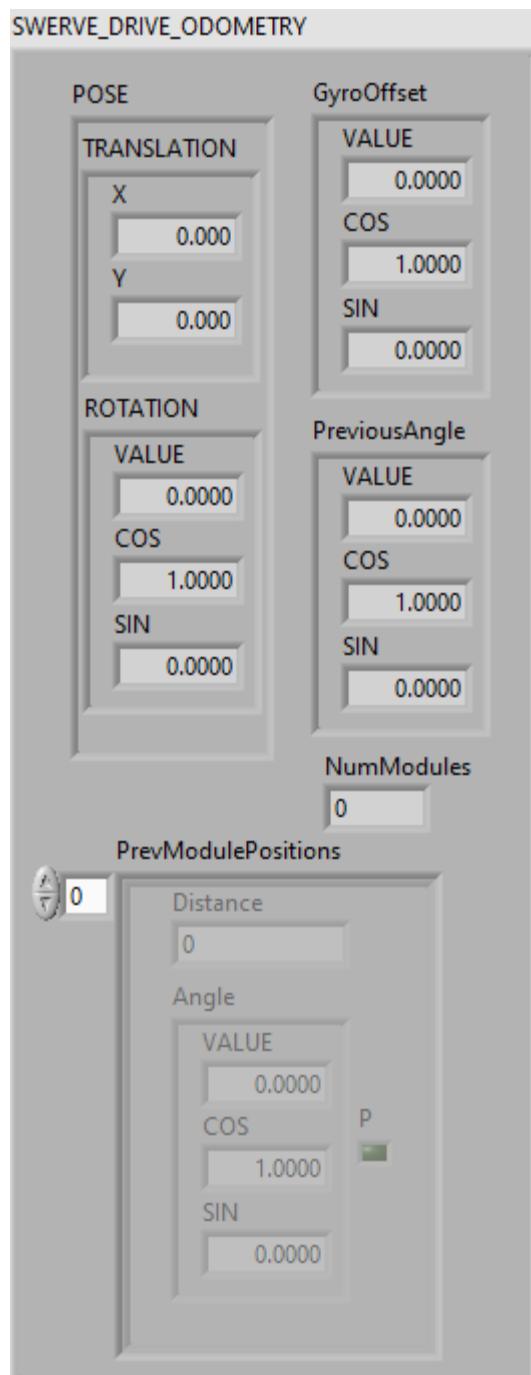


TypeDef-SWERVE_DRIVE_ODOMETRY



Class for swerve drive odometry. Odometry allows you to track the robot's position on the field over a course of a match using readings from your swerve drive encoders and swerve azimuth encoders.

Teams can use odometry during the autonomous period for complex tasks like path following. Furthermore, odometry can be used for latency compensation when using computer-vision systems.



TypeDef-SWERVE_DRIVE_POSE_EST



This set of functions is deprecated. Suggest using the new Swerve Drive Pose Est 2 instead.

The functions that use this data cluster wrap an UnscentedKalmanFilter Unscented Kalman Filter to fuse latency-compensated vision measurements with swerve drive encoder velocity measurements. It will correct for noisy measurements and encoder drift. It is intended to be an easy but more accurate drop-in for SwerveDriveOdometry.

The generic arguments to this data cluster define the size of the state, input and output vectors used in the underlying UnscentedKalmanFilter Unscented Kalman Filter. Num States must be equal to the module count + 3. Num Inputs must be equal to the module count + 3. Num Outputs must be equal to the module count + 1.

SwerveDrivePoseEstimator_update should be called every robot loop. If your loops are faster or slower than the default of 20 ms, then you should change the nominal delta time using the secondary constructor:
`SwerveDrivePoseEstimator_SwerveDrivePoseEstimator(Nat, Nat, Nat, Rotation2d, Pose2d, SwerveModulePosition[], SwerveDriveKinematics, Matrix, Matrix, Matrix, double).`

SwerveDrivePoseEstimator_addVisionMeasurement can be called as infrequently as you want; if you never call it, then this class will behave mostly like regular encoder odometry.

The state-space system used internally has the following states (x), inputs (u), and outputs (y):

$$x = [x, y, \theta, s_0, \dots, s_n]t$$

in the field coordinate system containing x position, y position, and heading, followed by the distance travelled by each wheel.

$$u = [v_x, v_y, \omega, v_0, \dots, v_n]t$$

containing x velocity, y velocity, and angular rate in the field coordinate system, followed by the velocity measured at each wheel.

$$y = [x, y, \theta]t$$

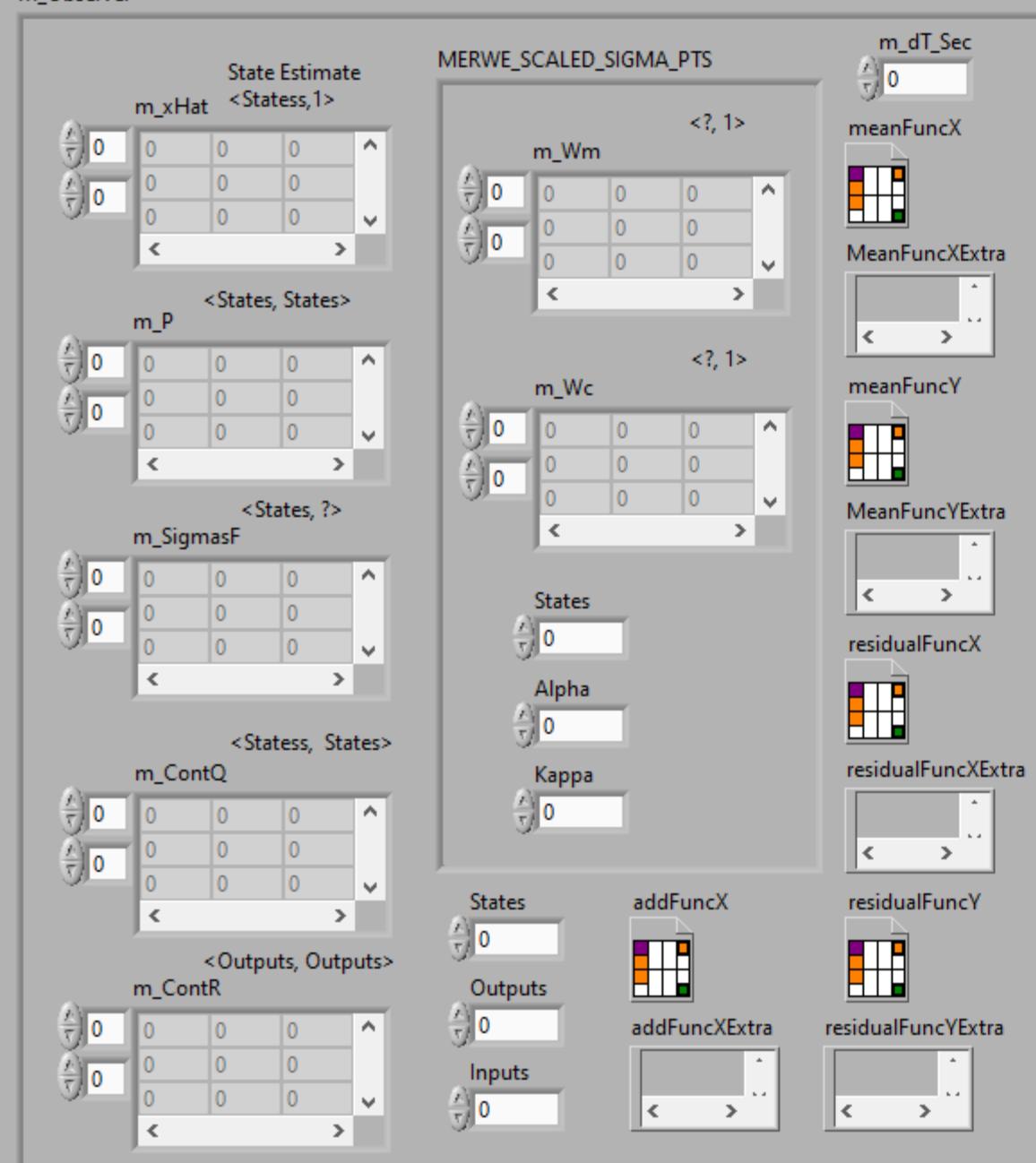
from vision containing x position, y position, and heading; or

$$y = [\theta, s_0, \dots, s_n]t$$

containing gyro heading, followed by the distance travelled by each wheel.

SWERVE_DRIVE_POSE_EST

m_Observer



TypeDef-SWERVE_DRIVE_POSE_EST2



This class wraps Swerve Drive Odometry to fuse latency-compensated vision measurements with swerve drive encoder measurements. It is intended to be a drop-in replacement for SwerveDrvOdom; in fact, if you never call SwerveDrvPoseEst_AddVisionMeasurement and only call SwerveDrvPoseEst_Update then this will behave exactly the same as SwerveDrvOdom.

SwerveDrvPoseEst2_Update should be called every robot loop.

SwerveDrvPoseEst2_AddVisionMeasurement can be called as infrequently as you want. If you never call it then this set of VI will behave exactly like regular encoder odometry.

This data cluster contains:

- Swerve_Drive_Kinematics -- Swerve_Drive_Kinematics -- Kinematics data cluster that defines this drive.
- Swerve_Pose_Odom -- Swerve_Drive_Odom -- Odometry data cluster for this drive.
- M_Q -- <3,1> Matrix -- Matrix that holds standard deviations for the robot pose X, Y, theta (meters, meters, radians)
- M_Vision_K -- <3,3> Matrix -- Matrix that holds the gain matrix for closed loop continuous Kalman filter.
- Pose_Buffer -- TimeInterpBufferVariant -- Holds last 1.5 seconds of position measurements.

SWERVE_DRIVE_POSE_EST2

Swerve_Drive_Kinematics

- NumModules: Value 0
- InverseKinematics: A 3x3 matrix with all values set to 0.
- ForwardKinematics: A 3x3 matrix with all values set to 0.
- Modules: A 2D array with X and Y fields both set to 0.000.
- PrevCoR: A 2D array with X and Y fields both set to 0.000.
- SavedModuleStates: Speed: Value 0. Angle: A group with VALUE (0.0000), COS (1.0000), and SIN (0.0000).

Swerve_Drive_Odom

- POSE: TRANSLATION X: Value 0.000, COS: 1.0000, SIN: 0.0000. ROTATION VALUE: Value 0.0000, COS: 1.0000, SIN: 0.0000.
- GyroOffset: VALUE: Value 0.0000, COS: 1.0000, SIN: 0.0000.
- PreviousAngle: VALUE: Value 0.0000, COS: 1.0000, SIN: 0.0000.
- NumModules: Value 0.
- PrevModulePositions: Distance: Value 0. Angle: A group with VALUE (0.0000), COS (1.0000), SIN (0.0000), and P (green checkmark).

M_Q <3, 1>

0	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

M_Vision_K <3, 3>

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

Pose_Buffer

TimeStamps

0	0
---	---

Min_TimeStamp

0

Max_TimeStamp

0

ItemCount

0

MaxBufferTime

0

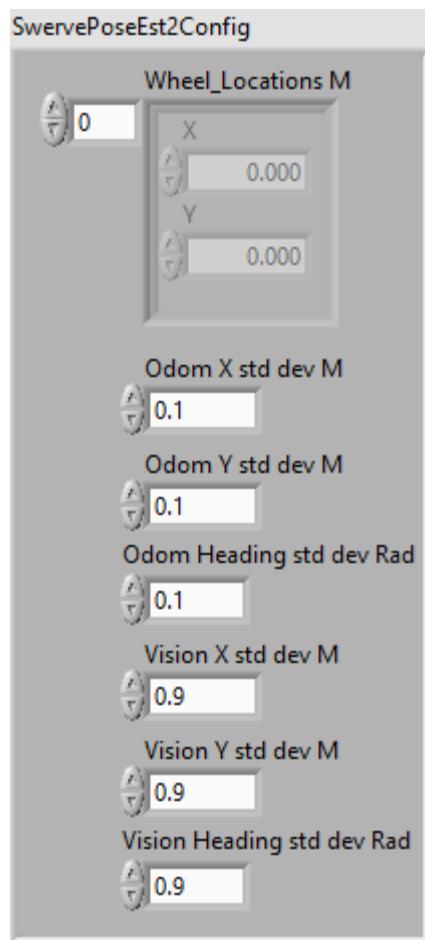
TypeDef-SWERVE_DRIVE_POSE_EST2_CONFIG



Cluster containing data required by the SwerveDrivePoseEst2_Execute function.

Contains:

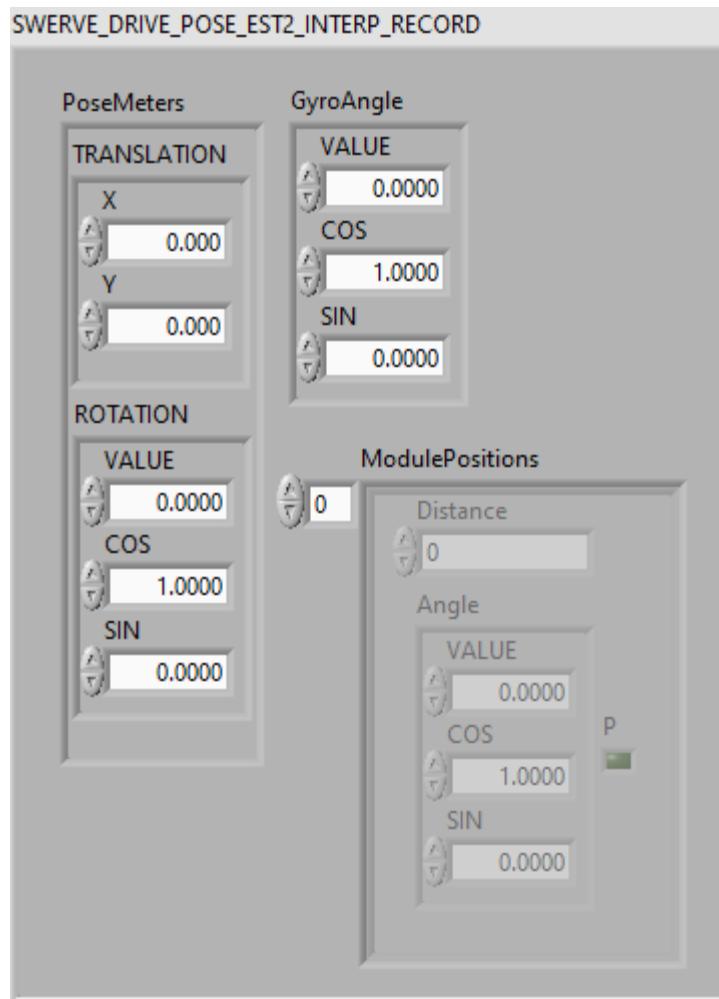
- Wheel Locations M -- Translation2d array -- Locations of module wheels in relation to center of robot. (meters)
- Odom X Std Dev M -- double -- Odometry X position standard deviation (Default: 0.02) (meters)
- Odom Y Std Dev M -- double -- Odometry Y position standard deviation (Default: 0.02) (meters)
- Odom Heading Std Dev M -- double -- Odometry Heading (gyro) position standard deviation (Default: 0.01) (radians)
- Vision X Std Dev M -- double -- Vision X position standard deviation (Default: 0.1) (meters)
- Vision Y Std Dev M -- double -- Vision Y position standard deviation (Default: 0.1) (meters)
- Vision Heading Std Dev M -- double -- Vision Heading (gyro) position standard deviation (Default: 0.05) (radians)



TypeDef-SWERVE_DRIVE_POSE_EST2_INTERP_RECORD



Differential Drive Pose Estimate 2 -- Interporatable Record



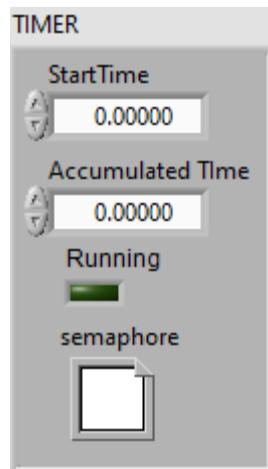
TypeDef-TIMER



Data cluster holding values required by the TImer functions. These functions allow timing between events without interrupting execution of other robot code.

Data:

- StartTIme -- Time the timer was started (Seconds). This is not wall clock time, rather it uses a continuously counting value from some reference, generally the last time the computer was rebooted.
- AccumulatedTime -- Internal time variable
- Running -- If TRUE, the timer is running
- Semaphore -- Used to synchronize interactions between threads.

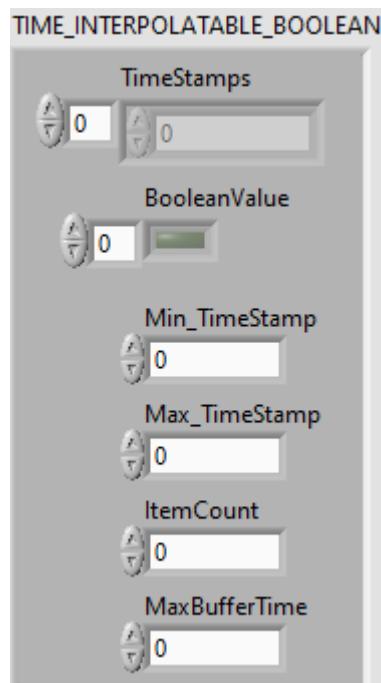


TypeDef-TIME_INTERPOLATABLE_BOOLEAN



The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_BOOLEAN stores and returns boolean values.

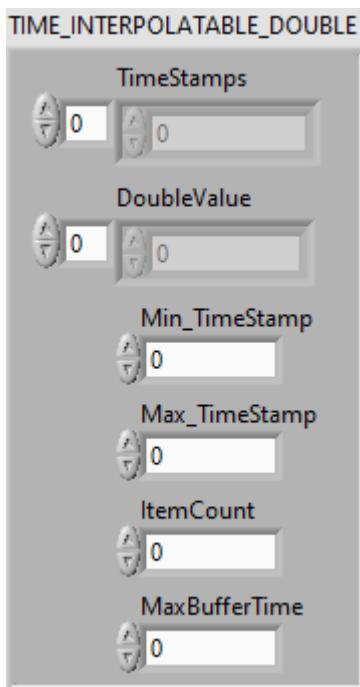


TypeDef-TIME_INTERPOLATABLE_DOUBLE



The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_DOUBLE stores and returns DOUBLE values.

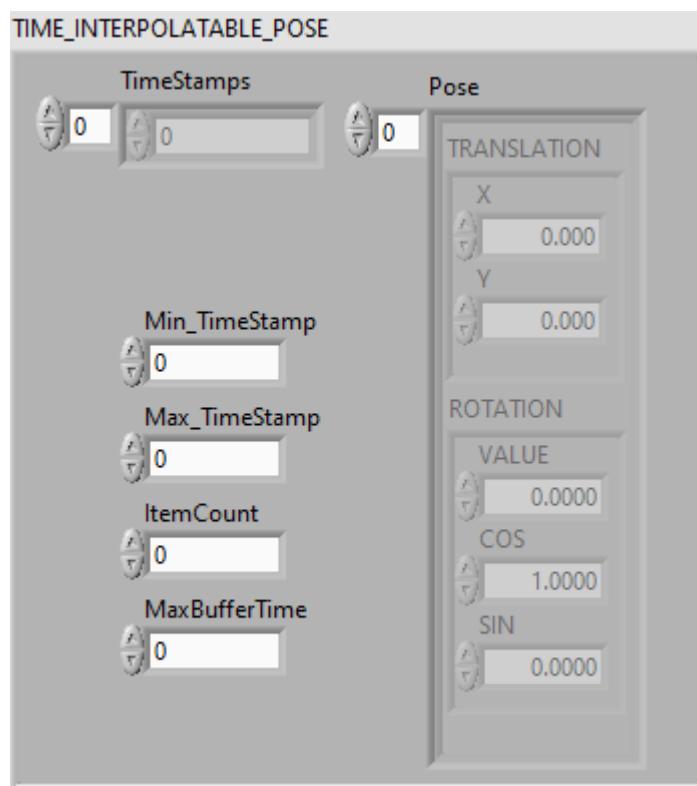


TypeDef-TIME_INTERPOLATABLE_POSE2D



The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_POSE stores and returns POSE2D values.

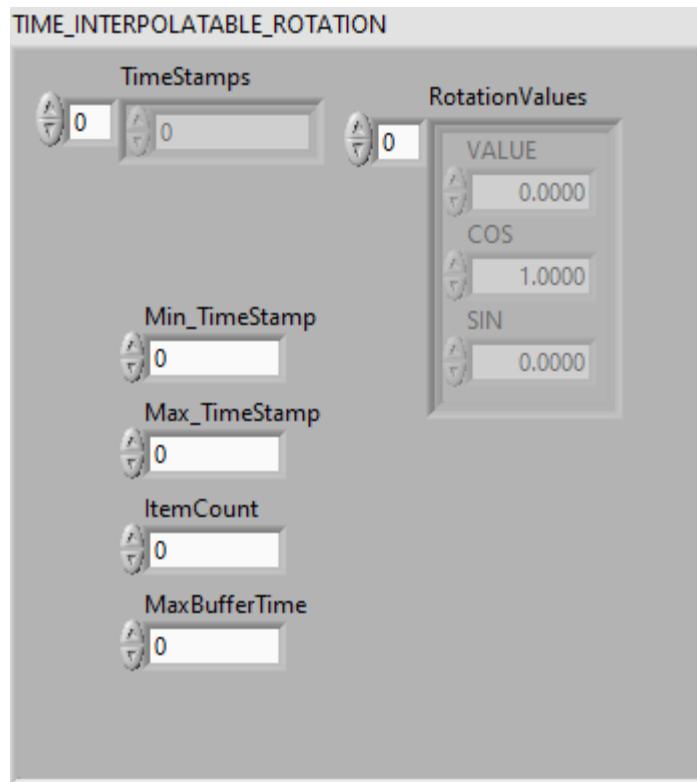


TypeDef-TIME_INTERPOLATABLE_ROTATION2D



The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_ROTATION stores and returns ROTATION2D values.

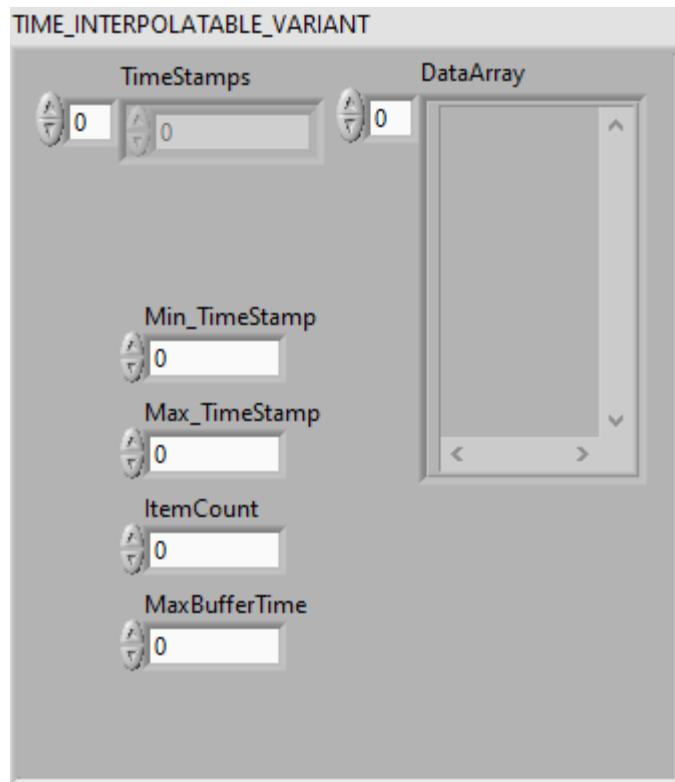


TypeDef-TIME_INTERPOLATABLE_VARIANT



The TimeInterpolatableBuffer provides an easy way to estimate past measurements. One application might be in conjunction with the DifferentialDrivePoseEstimator, where knowledge of the robot pose at the time when vision or other global measurement were recorded is necessary, or for recording the past angles of mechanisms as measured by encoders.

The TIME_INTERPOLATABLE_POSE stores and returns POSE2D values.



TypeDef-TRAJECTORY



Represents a time-parameterized trajectory. The trajectory contains various States that represent the pose, curvature, time elapsed, velocity, and acceleration at that point.



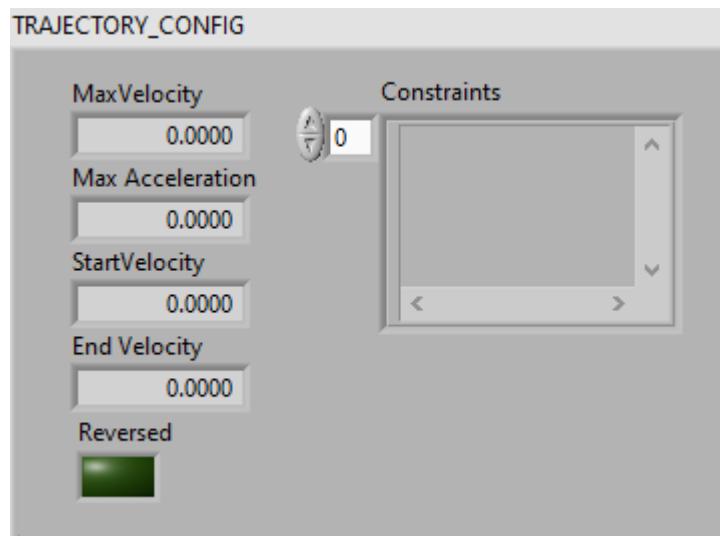
TypeDef-TRAJ_CONFIG



Represents the configuration for generating a trajectory. This class stores the start velocity, end velocity, max velocity, max acceleration, custom constraints, and the reversed flag.

The cluster must be constructed with a max velocity and max acceleration. The other parameters (start velocity, end velocity, constraints, reversed) have been defaulted to reasonable values (0, 0, {}, false). These values can be changed via the setXXX methods.

It also contains the data for each constraint and a flag indicating if a particular constraint is active. As new constraints are added, this cluster will be modified to contain them.



TypeDef-TRAJ_CONSTRAINT_CENTRIPETAL_ACCEL

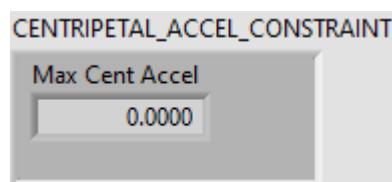


A constraint on the maximum absolute centripetal acceleration allowed when traversing a trajectory. The centripetal acceleration of a robot is defined as the velocity squared divided by the radius of curvature.

Effectively, limiting the maximum centripetal acceleration will cause the robot to slow down around tight turns, making it easier to track trajectories with sharp turns.

Elements:

- MaxCentripetalAccel - The max centripetal acceleration. (meters/Sec²)



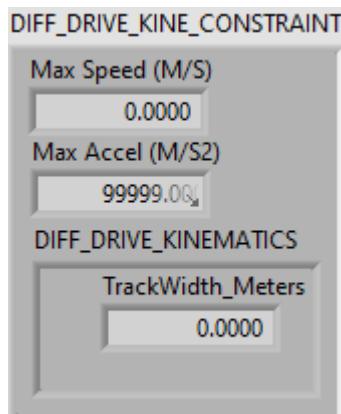
TypeDef-TRAJ_CONSTRAINT_DIFF_DRIVE_KINEMATICS



A class that enforces constraints on the differential drive kinematics. This can be used to ensure that the trajectory is constructed so that the commanded velocities for both sides of the drivetrain stay below a certain limit.

Elements:

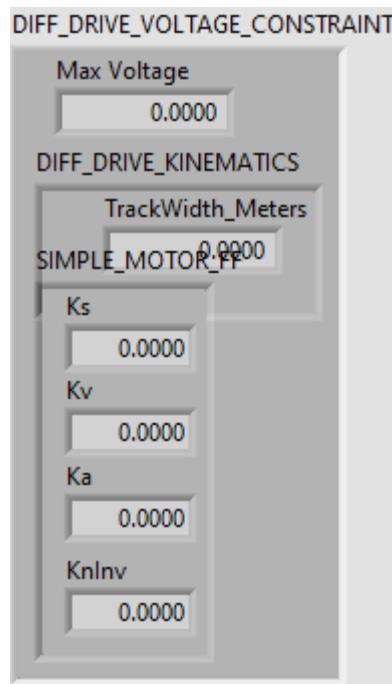
- Max Speed (Meters/Sec)
- DiffDriveKinematics - data structure



TypeDef-TRAJ_CONSTRAINT_DIFF_DRIVE_VOLTAGE



A class that enforces constraints on differential drive voltage expenditure based on the motor dynamics and the drive kinematics. Ensures that the acceleration of any wheel of the robot while following the trajectory is never higher than what can be achieved with the given maximum voltage.



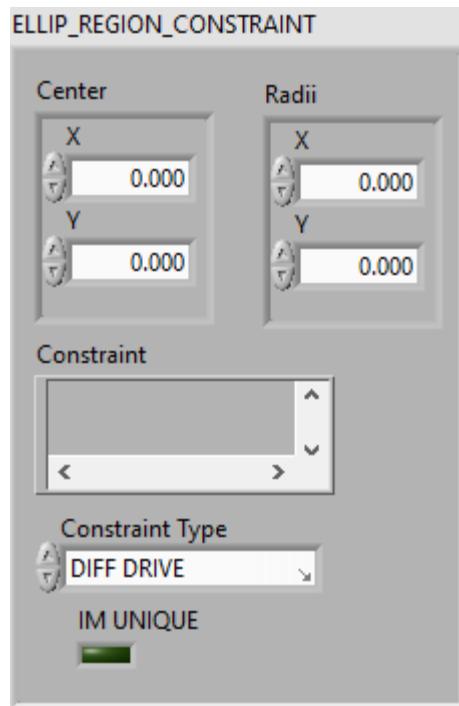
TypeDef-TRAJ_CONSTRAINT_ELLIP_REGION



A class that enforces a constraint when robot pose is within a defined elliptical region.

Elements:

- Center - Translation defining the center of the ellipse.
- Radii -- Translation defining the size of the ellipse.
- Constraint -- Variant holding the constraint data cluster to enforce when inside the ellipse.
- Constraint Type -- Enum that indicates the type of the constraint to enforce.



TypeDef-TRAJ_CONSTRAINT_JERK

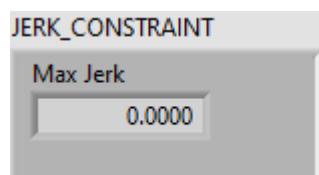


A constraint on the maximum absolute centripetal acceleration allowed when traversing a trajectory. The centripetal acceleration of a robot is defined as the velocity squared divided by the radius of curvature.

Effectively, limiting the maximum centripetal acceleration will cause the robot to slow down around tight turns, making it easier to track trajectories with sharp turns.

Elements:

- MaxCentripetalAccerl - The max centripetal acceleration. (meters/Sec²)



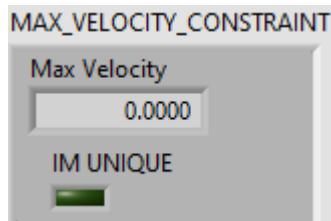
TypeDef-TRAJ_CONSTRAINT_MAX_VELOCITY



A constraint on the maximum absolute velocity allowed when traversing a trajectory.

Elements:

- MaxVelocity - The max velocity. (meters/Sec)



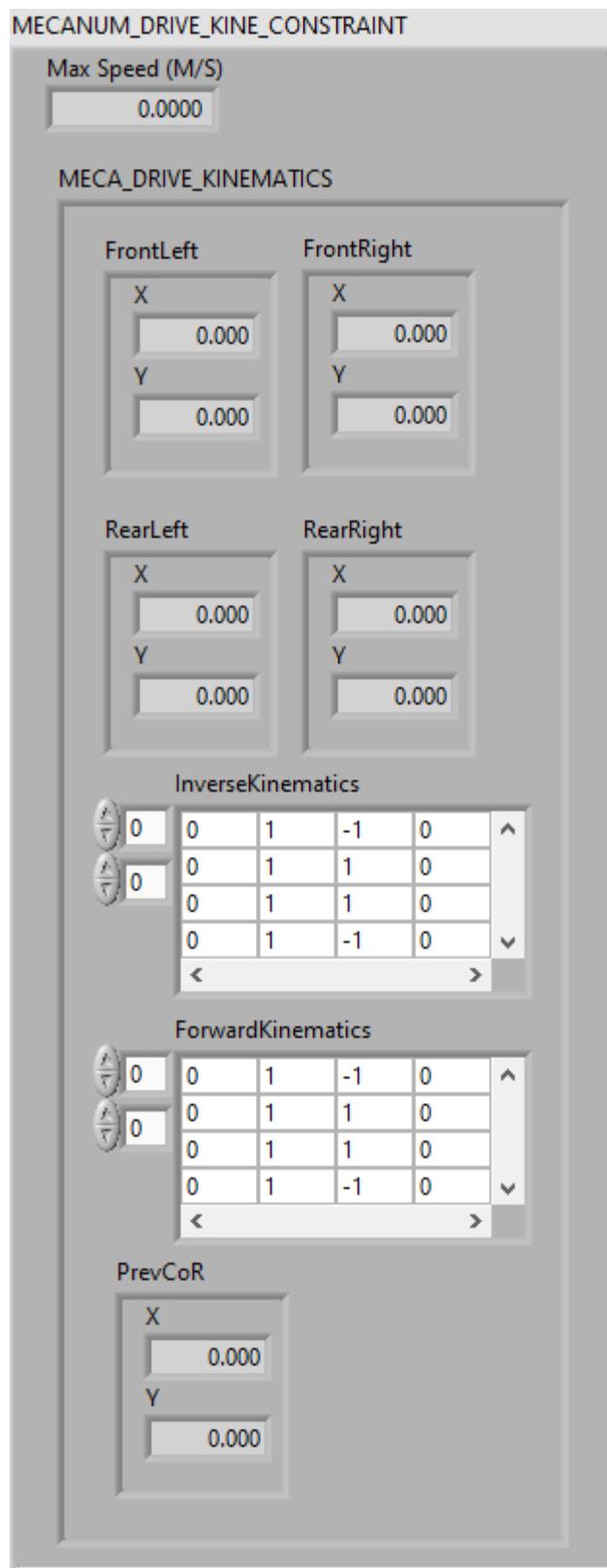
TypeDef-TRAJ_CONSTRAINT_MECA_DRIVE_KINEMATICS



A class that enforces constraints on the mecanum drive kinematics. This can be used to ensure that the trajectory is constructed so that the commanded velocities all wheels of the drivetrain stay below a certain limit.

Elements:

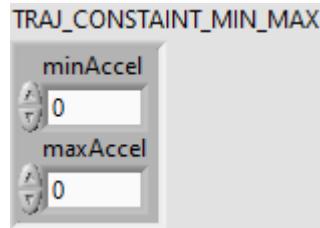
- Max Speed (Meters/Sec)
- MecanumDriveKinematics - data structure



TypeDef-TRAJ_CONSTRAINT_MINMAX



Represents a minimum and maximum acceleration. This is used exclusively by TrajectoryParam_timeParam and its internal routines.



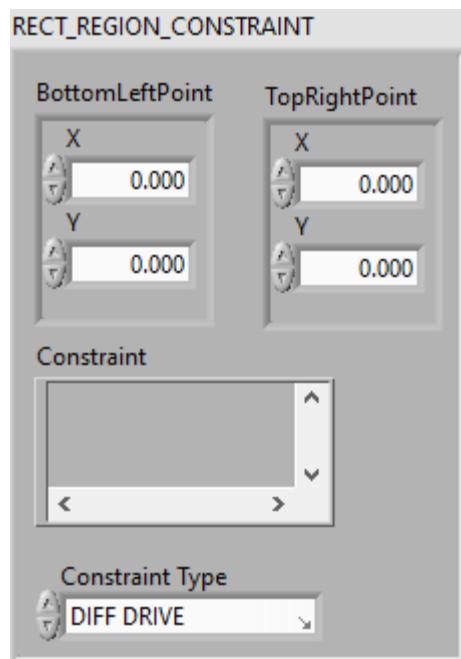
TypeDef-TRAJ_CONSTRAINT_RECT_REGION



A class that enforces constraints when inside a rectangular area.

Elements:

- BottomLeftPoint -- Defines the rectangle.
- TopRightPoint -- Defines the rectangle.
- Constraint - Variant holding the constraint definition to use when inside the rectangle
- Constraint Type -- Enum containing the type of constraint.
- DiffDriveKinematics - data structure



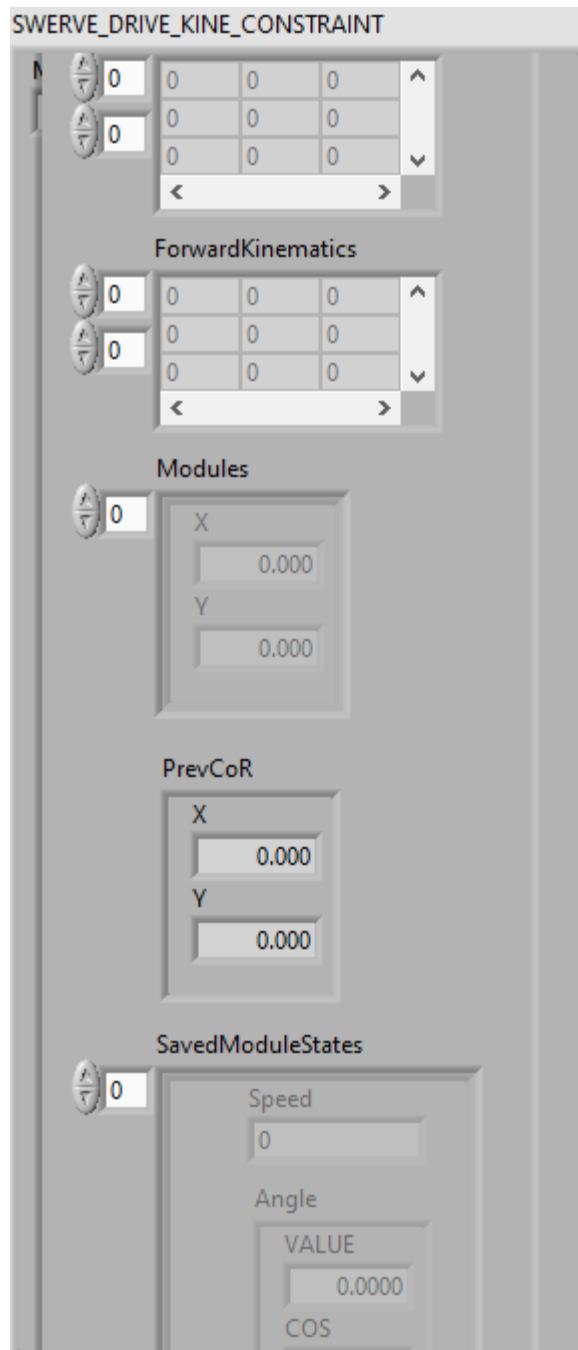
TypeDef-TRAJ_CONSTRAINT_SWERVE_DRIVE_KINEMATICS



A class that enforces constraints on the swerve drive kinematics. This can be used to ensure that the trajectory is constructed so that the commanded velocities drivetrain modules stay below a certain limit.

Elements:

- Max Speed (meters/sec)
- Swerve Drive Kinematics data cluster



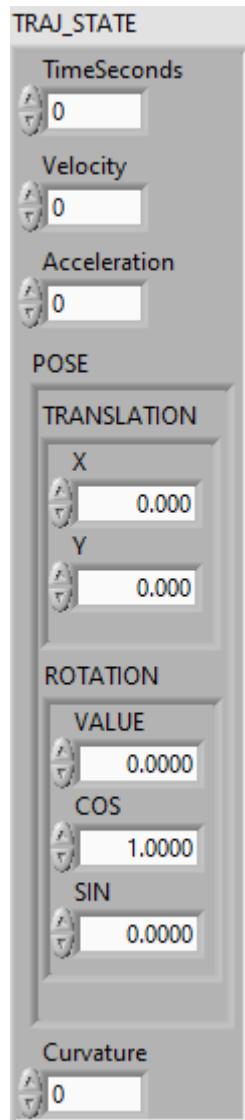
TypeDef-TRAJ_STATE



Represents a time-parameterized trajectory. The trajectory contains of various States that represent the pose, curvature, time elapsed, velocity, and acceleration at that point.

Elements:

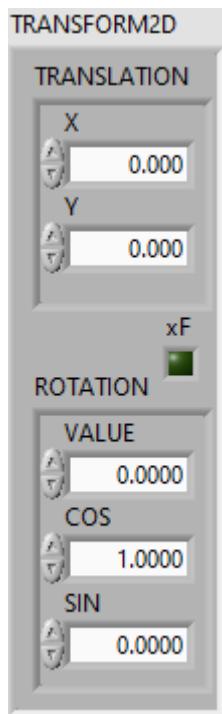
- TimeSeconds - The time elapsed since the beginning of the trajectory. (seconds)
- Velocity - The speed at that point of the trajectory. (meters/sec)
- Acceleration - The acceleration at that point of the trajectory. (meters/Sec²)
- POSE - The pose at that point of the trajectory.
- Curvature - The curvature at that point of the trajectory. (Radians/Meter)



TypeDef-TRANSFORM2D



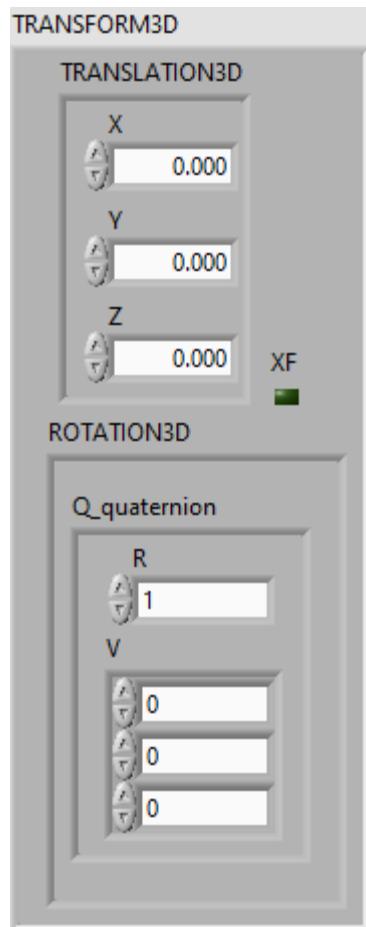
Represents a transformation for a Pose2d.



TypeDef-TRANSFORM3D



Represents a transformation for a Pose3d.

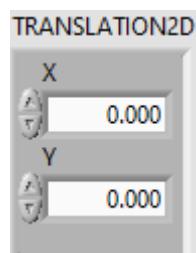


TypeDef-TRANSLATION2D



Represents a translation in 2d space. This object can be used to represent a point or a vector.

This assumes that you are using conventional mathematical axes. When the robot is placed on the origin, facing toward the X direction, moving forward increases the X, whereas moving to the left increases the Y.

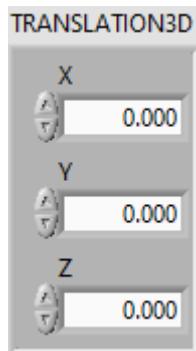


TypeDef-TRANSLATION3D



Represents a translation in 3D space. This object can be used to represent a point or a vector.

This assumes that you are using conventional mathematical axes. When the robot is at the origin facing in the positive X direction, forward is positive X, left is positive Y, and up is positive Z.



TypeDef-TRAPEZOID_PROFILE



A trapezoid-shaped velocity profile.

While this class can be used for a profiled movement from start to finish, the intended usage is to filter a reference's dynamics based on trapezoidal velocity constraints. To compute the reference obeying this constraint, do the following.

Initialization:

- Create a new trapezoid profile constraint and provide Max Velocity and Max Acceleration
- Create an initial previous profile reference state and provide current position (distance, angle, or other) and Velocity

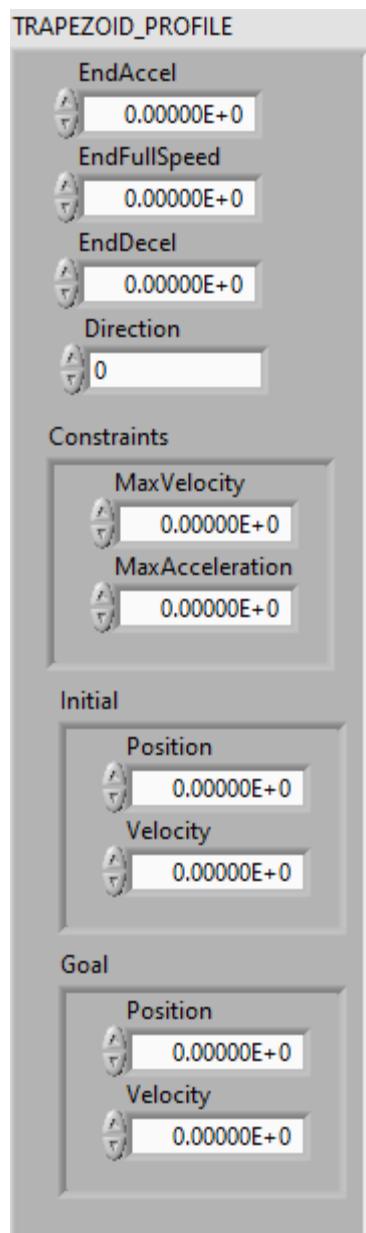
Run on update:

- Create a new trapezoid profile given the constraints, unprofiled (current) reference and the previous profile reference
- Calculate providing the time since last update, the result is a new previous profile reference

where `unprofiledReference` is free to change between calls. Note that when the unprofiled reference is within the constraints, `calculate()` returns the unprofiled reference unchanged.

Otherwise, a timer can be started to provide monotonic values for `calculate()` and to determine when the profile has completed via `isFinished()`.

- EndAccel --
- EndFullSpeed --
- EndDeccel --
- Direction -- The direction of the profile, either 1 for forwards or -1 for inverted
- Constraints --
- Initial --
- Goal --

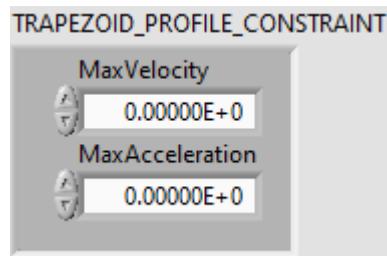


TypeDef-TRAPEZOID_PROFILE_CONSTRAINT



Contains the constraints for a TrapezoidProfile.

- maxVelocity -- maximum velocity
- maxAcceleration -- maximum acceleration

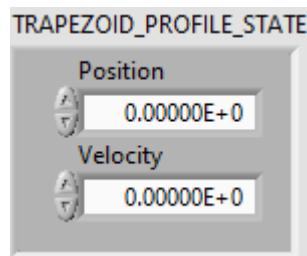


TypeDef-TRAPEZOID_PROFILE_STATE



Contains the State, initial, current, or goal, for a Trapezoid Profile

- Position -- Distance, or heading, or other parameter for which the profile is created.
- Velocity -- Velocity

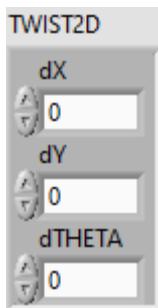


TypeDef-TWIST2D



A change in distance along arc since the last pose update. We can use ideas from differential calculus to create new Pose2ds from a Twist2d and vice versa.

A Twist can be used to represent a difference between two poses.

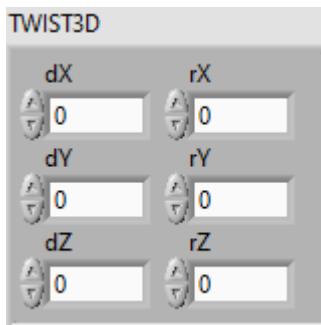


TypeDef-TWIST3D



A change in distance along a 3D arc since the last pose update. We can use ideas from differential calculus to create new Pose3ds from a Twist3d and vice versa.

A Twist can be used to represent a difference between two poses.



TypeDef-UNSCENTED_KALMAN_CORRECT_FUNC_GROUP

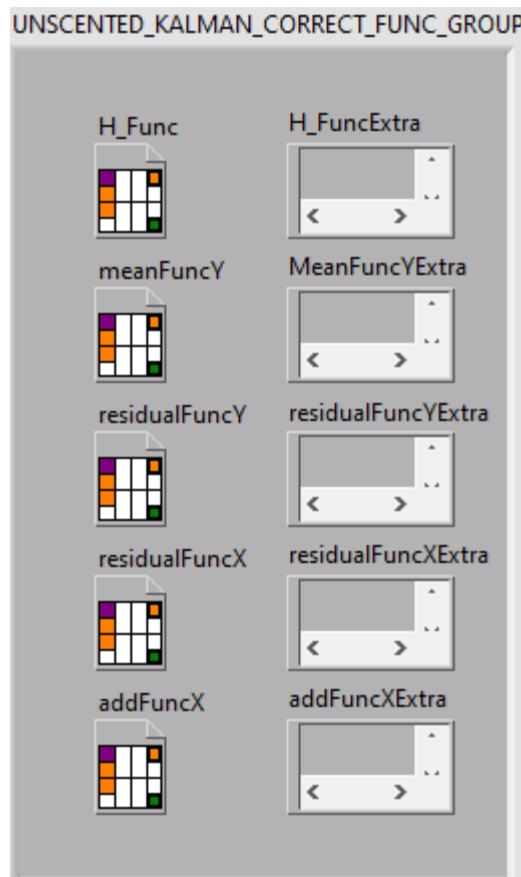


Cluster containing the packed function group data to pass to the Correct subVI.

Data:

- H_Func -- A vector-valued function of x and u that returns the measurement vector.

- H_ExtraData -- Variant containing extra data used by the H function. The contents of the variant are specific to the H function.
- meanFuncY -- A strict function reference that computes the mean of 2 States + 1 measurement vectors using a given set of weights.
- meanFuncYExtra -- Variant containing extra data used by the meanY function. The contents of the variant are specific to the function.
- residualFuncY -- A strict function reference that computes the residual of two measurement vectors (i.e. it subtracts them.)
- residualFuncYExtra -- Variant containing extra data used by the residualY function. The contents of the variant are specific to the function.
- residualFuncX -- A strict function reference that computes the residual of two state vectors (i.e. it subtracts them.)
- residualFuncXExtra -- Variant containing extra data used by the residualX function. The contents of the variant are specific to the function.
- addFuncX -- A strict function reference that adds two state vectors.
- addFuncXExtra -- Variant containing extra data used by the addY function. The contents of the variant are specific to the function.



TypeDef-UNSCENTED_KALMAN_FILTER



A Kalman filter combines predictions from a model and measurements to give an estimate of the true system state. This is useful because many states cannot be measured directly as a result of sensor noise, or because the state is "hidden".

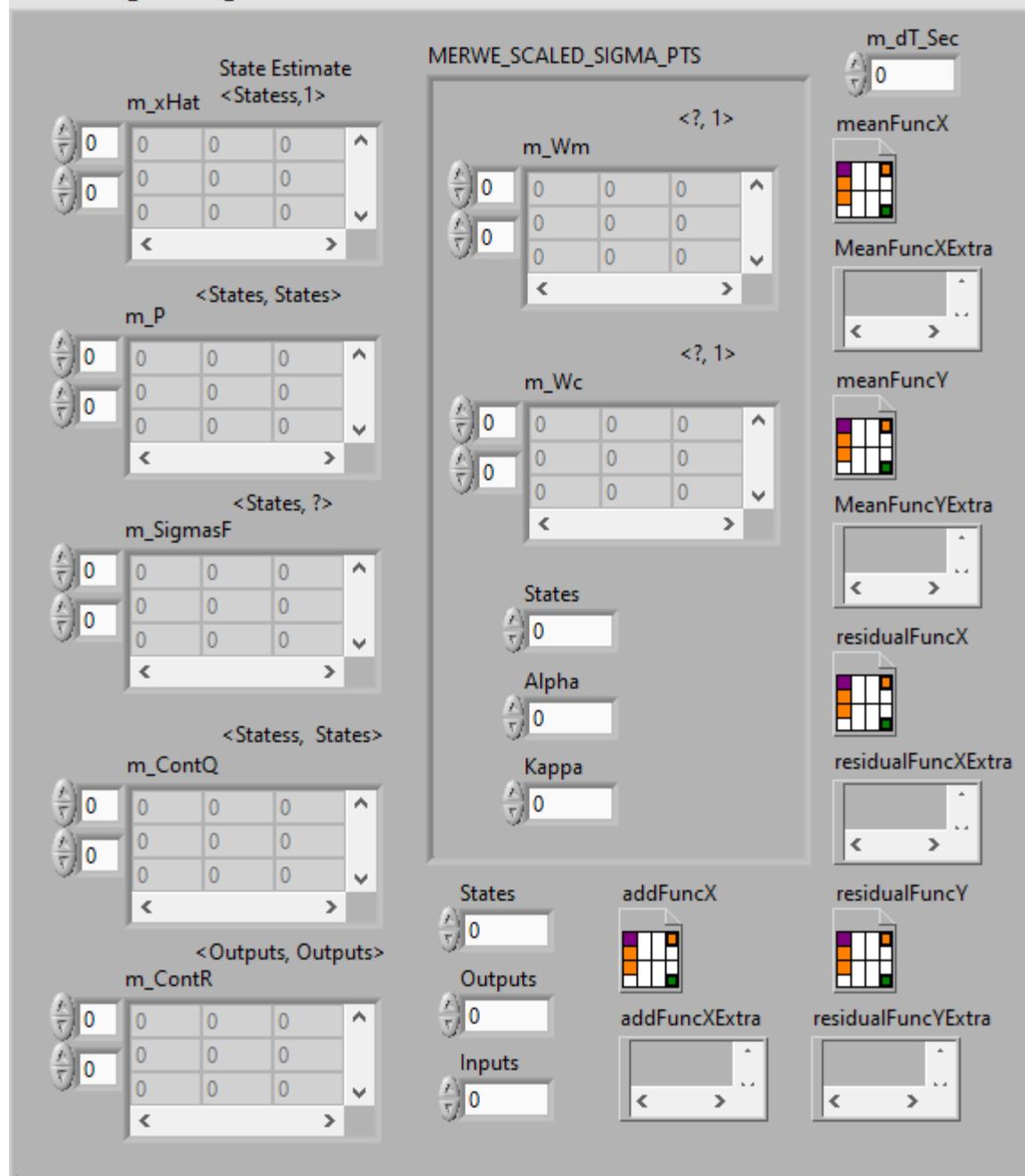
Kalman filters use a K gain matrix to determine whether to trust the model or measurements more. Kalman filter theory uses statistics to compute an optimal K gain which minimizes the sum of squares error in the state estimate. This K gain is used to correct the state estimate by some amount of the difference between the actual measurements and the measurements predicted by the model.

An unscented Kalman filter uses nonlinear state and measurement models. It propagates the error covariance using sigma points chosen to approximate the true probability distribution.

The data contained in this cluster is:

- xHat --
- P --
- SigmasF --
- ContQ --
- ContR --
- MerweScaledSigmaPts --
 - States -- Number of states
 - Outputs -- Number of outputs
 - Inputs -- Number of inputs
 - dT_Sec -- Time between calls (Seconds)
- addFuncX -- A strict function reference that adds two state vectors.
- addFuncXExtra -- A variant containing extra data for the addX function
- meanFuncX -- A strict function reference that computes the mean of 2States + 1 state vectors using a given set of weights.
 - meanFuncXExtra -- A variant containing extra data for the meanX function
 - meanFuncY -- A strict function reference that computes the mean of 2 States + 1 measurement vectors using a given set of weights.
 - meanFuncYExtra -- A variant containing extra data for the meanY function
- residualFuncX -- A strict function reference that computes the residual of two state vectors (i.e. it subtracts them.)
- residualFuncXExtra -- A variant containing extra data for the residualX function
- residualFuncY -- A strict function reference that computes the residual of two measurement vectors (i.e. it subtracts them.)
- residualFuncYExtra -- A variant containing extra data for the residualY function

UNSCENTED_KALMAN_FILTER



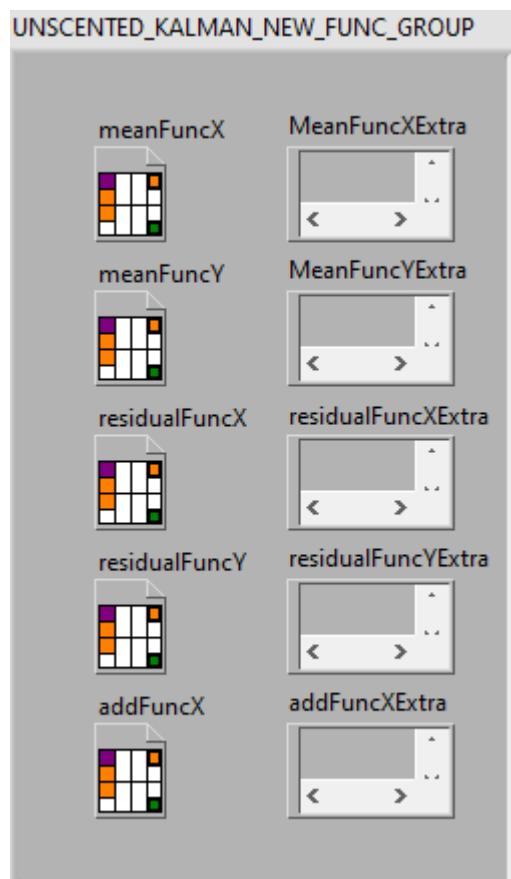
TypeDef-UNSCENTED_KALMAN_NEW_FUNC_GROUP



A packed cluster to hold the call back function reference information:

The cluster contains:

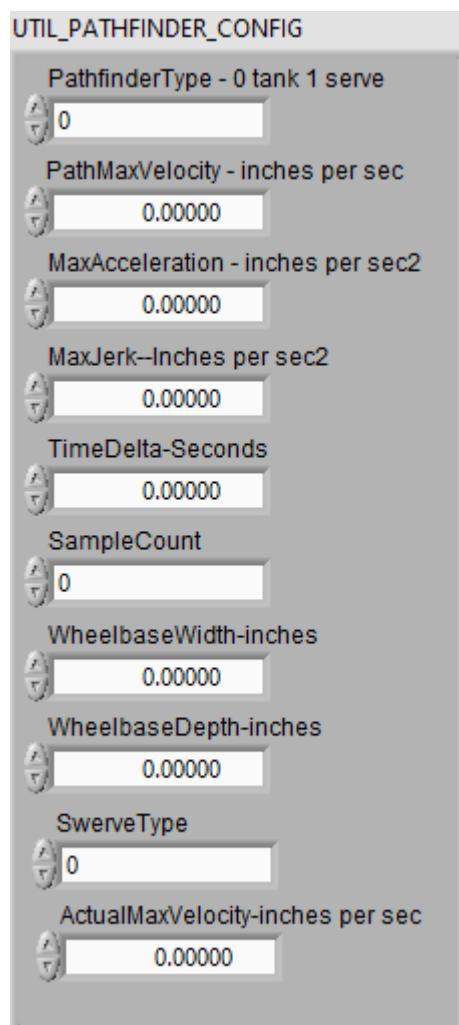
- meanFuncX -- A strict function reference that computes the mean of 2States + 1 state vectors using a given set of weights.
- meanFuncXExtra -- A variant containing extra data for the meanX function
- meanFuncY -- A strict function reference that computes the mean of 2 States + 1 measurement vectors using a given set of weights.
- meanFuncYExtra -- A variant containing extra data for the meanY function
- residualFuncX -- A strict function reference that computes the residual of two state vectors (i.e. it subtracts them.)
- residualFuncXExtra -- A variant containing extra data for the residualX function
- residualFuncY -- A strict function reference that computes the residual of two measurement vectors (i.e. it subtracts them.)
- residualFuncYExtra -- A variant containing extra data for the residualY function
- addFuncX -- A strict function reference that adds two state vectors.
- addFuncXExtra -- A variant containing extra data for the addX function



TypeDef-UTIL_PATHFINDER_CONFIG



Util data structure for printing of pathfinder path data.



TypeDef-WEIGHTED_WAYPOINT



Spline / Trajectory way point with scalar weight.

Data:

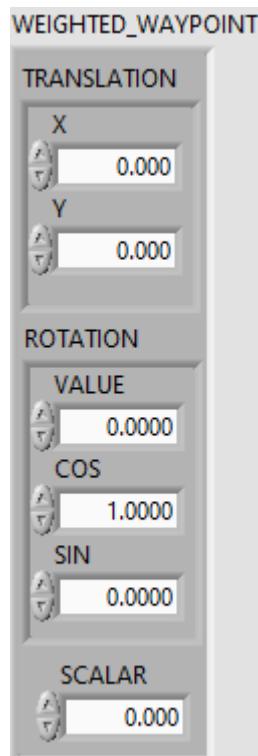
- Translation

- X -- X location (Generally meters)

- Y -- Y location (Generally meters)

- Rotation -- Heading (direction of travel)

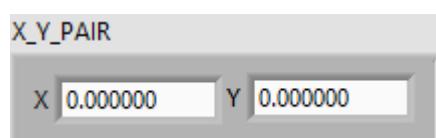
- Scalar -- Weight value. When using weights, the value must be greater than 0. Larger values cause the spline to be straighter entering and leaving this waypoint.



TypeDef-X_Y_PAIR



Pair of X and Y value



Enumerated Type Definitions

Enum

Enum-AprilTagFieldLayoutOriginPosition_ENUM



Enum typedef allowing selection of the origin of a field definition.



Enum-AprilTagFields_ENUM



Enum type to select a particular FRC Field definition.

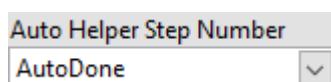


Enum-AutoHelperSequence_Step_Enum



Enumerated data type for Drum Sequence Step output.

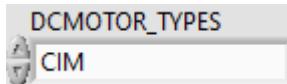
This contains upto 32 steps, Cancel, Error, Initialize, and Do Nothing states



Enum-DCMOTOR_TYPES_ENUM



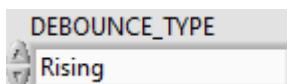
An enumerated variable type containing all the standard motors.



Enum-DEBOUNCER_TYPE_ENUM



An enumerated variable type containing all the standard motors.



Enum-DIFF_DRIVE_KitBot_WheelSize_ENUM



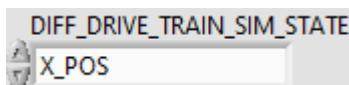
Enumerated variable for standard wheel diameters (in inches) for the Andy Mark Kit robot.



Enum-DIFF_DRIVE_TRAIN_SIM_STATE_ENUM



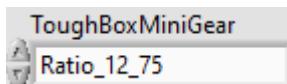
An enumerated variable containing pneumonics for the state indices of the Differential Drive Train sim system.



Enum-DIFF_DRIVE_ToughBoxMini_GearChoice_ENUM



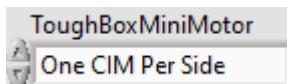
An enumerated variable listing the gear ratio choices for the Andy Mark ToughBox Mini gearbox that is part of the standard kit of parts.



Enum-DIFF_DRIVE_ToughBoxMini_MotorChoice_ENUM



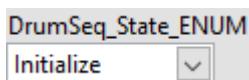
An enumerated variable listing the standard motor choices for the Andy Mark kit of parts drive gear box.



Enum-DrumSequence_State_Enum



Internal enumerated data type used for internal operations of the Drum Sequencer.



Enum-DrumSequence_Step_Enum



Enumerated data type for Drum Sequence Step output.

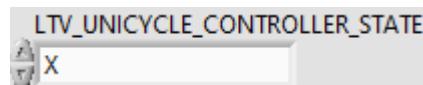
This contains up to 32 steps, Cancel, Error, Initialize, and Do Nothing states



Enum-LTV_UNICYCLE_CONTROLLER_STATE_ENUM



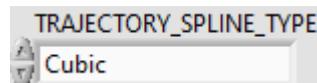
An enumerated variable containing States of the LV Unicycle Controller drivetrain system



Enum-TRAJECTORY_SPLINE_TYPE_ENUM



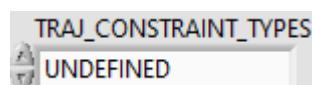
A ENUM (Enumerated variable) allowing the choice of the types of splines that can be used to generate trajectories.



Enum-TRAJ_CONSTRAINT_TYPES_ENUM



An enumerated variable type containing all the standard motors.



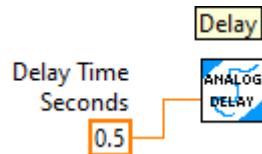
Appendix A – Snippet List

Snippets are used by menus to place entire sections of code into a block diagram. The contents of the menu snippets are listed here for easy reference.

In this library snippets are also referred to as macros.

Macro

macro_Analog_Delay



macro_AngleStats_AngleAdd_CallbackHelp



macro_AngleStats_AngleMean_CallbackHelp



macro_AngleStats_AngleResidual_CallbackHelp



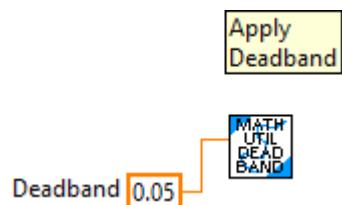
macro_Angle_Modulus



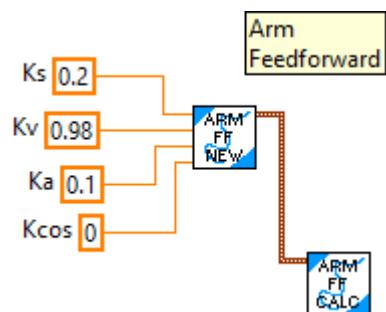
Wraps an angle in Radians to +/- PI
For other ranges use Input Modulus.



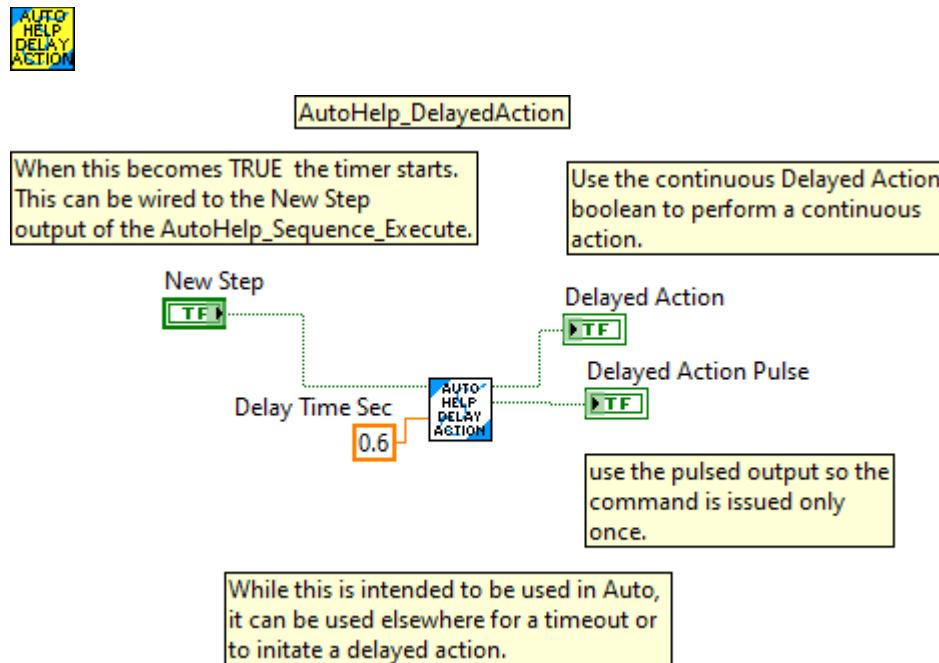
macro_Apply_Deadband



macro_ArmFF_Calculate



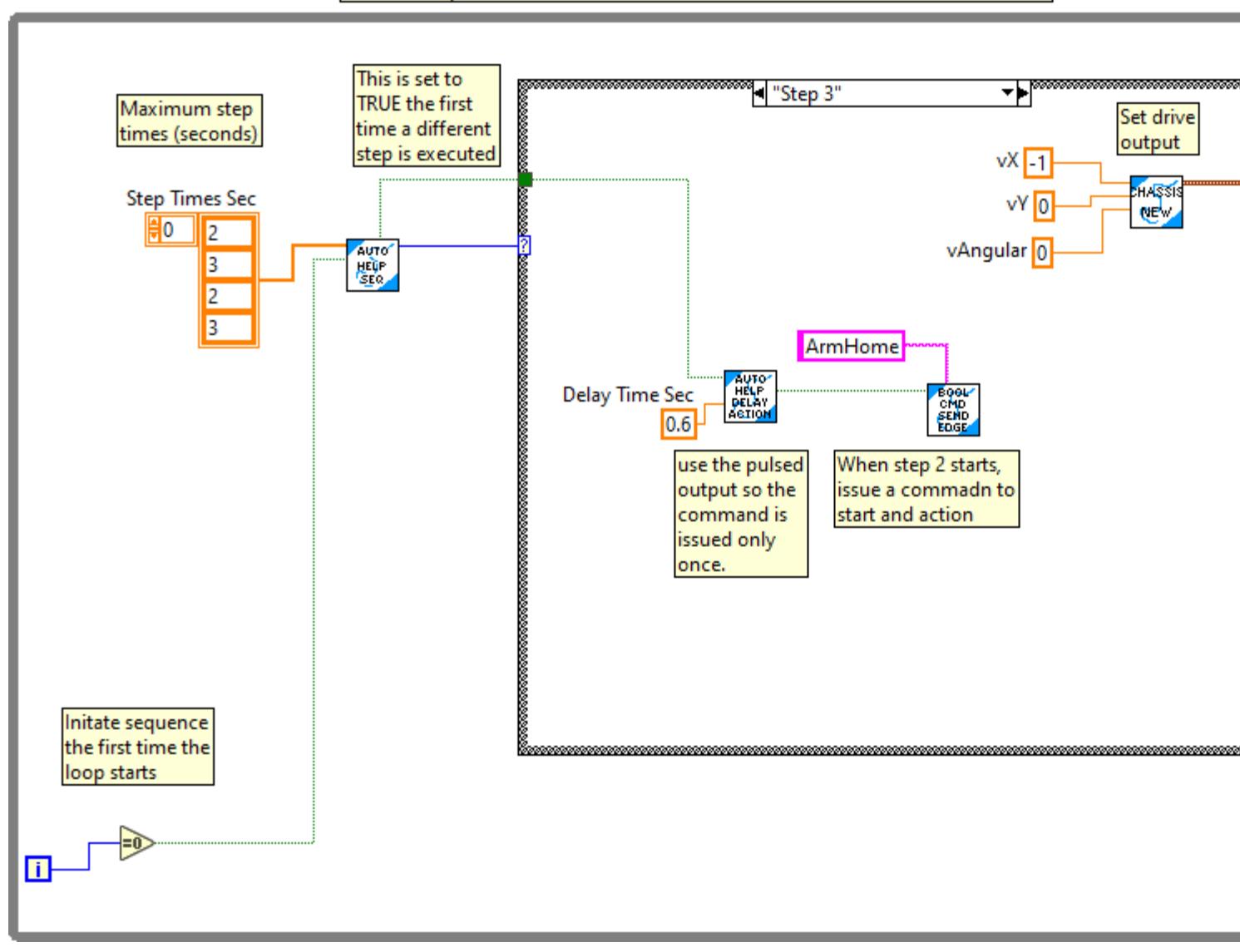
macro_AutoHelp_DelayedAction



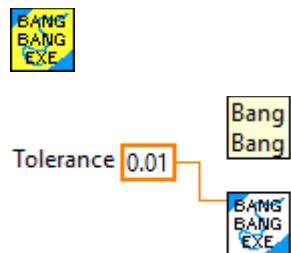
macro_AutoHelp_Sequence

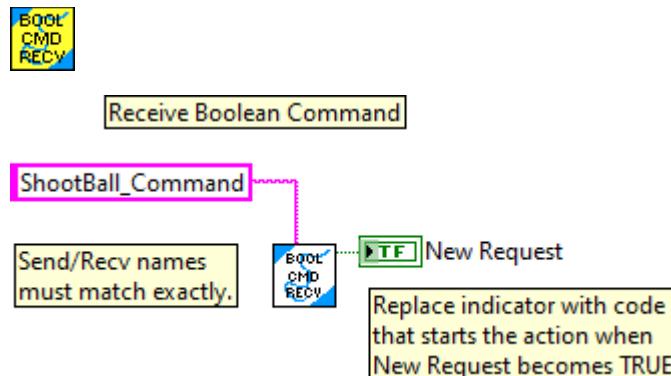
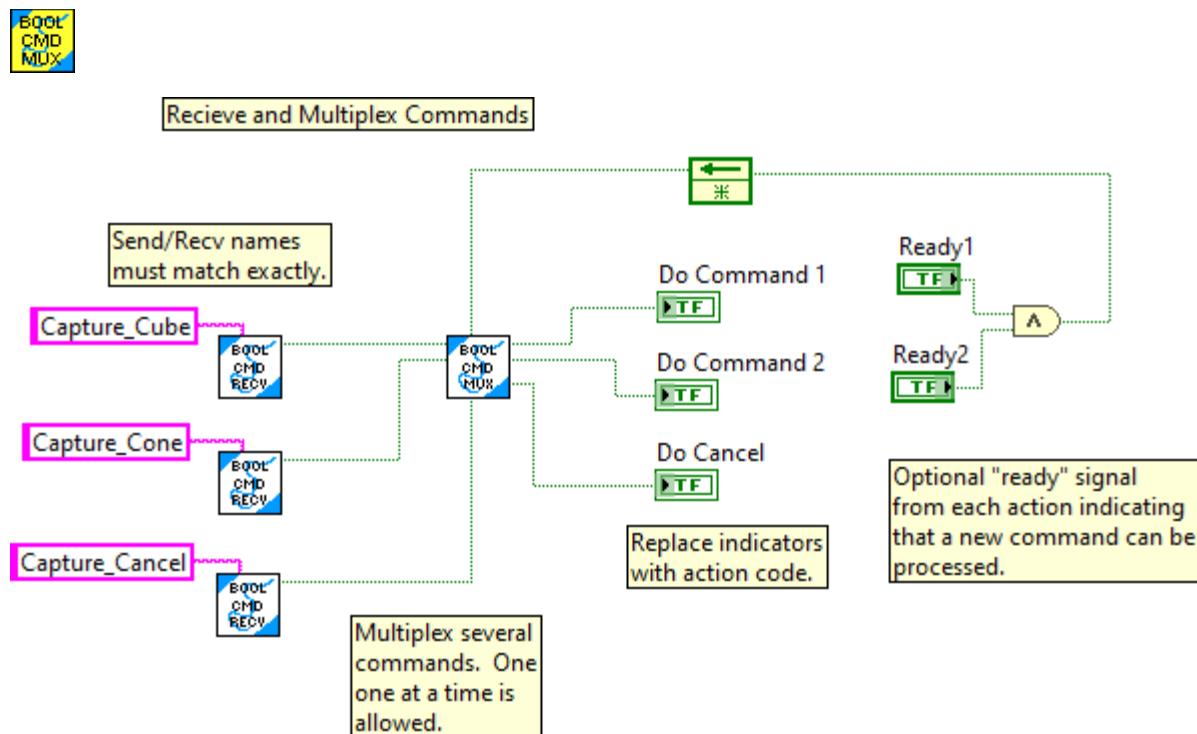


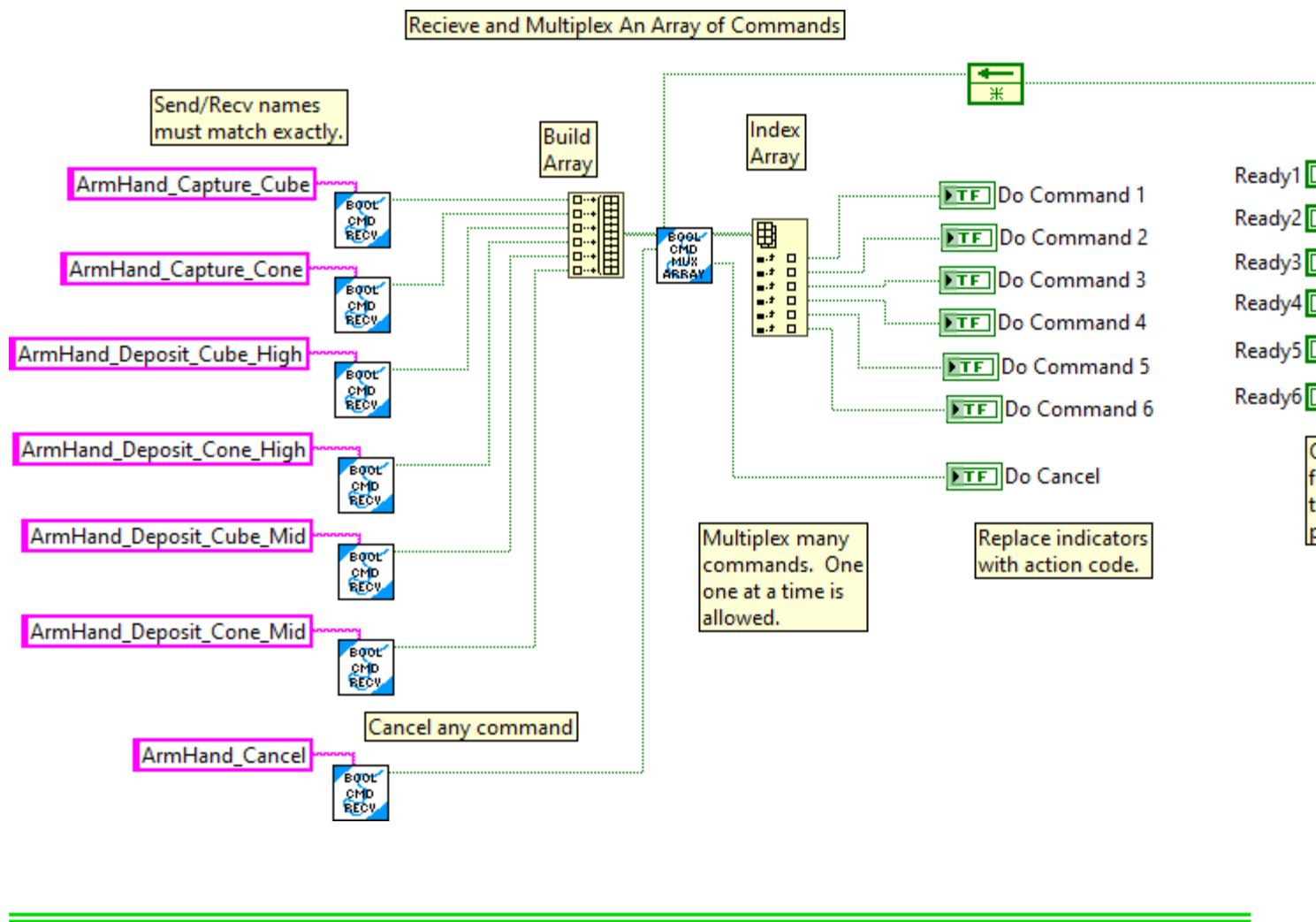
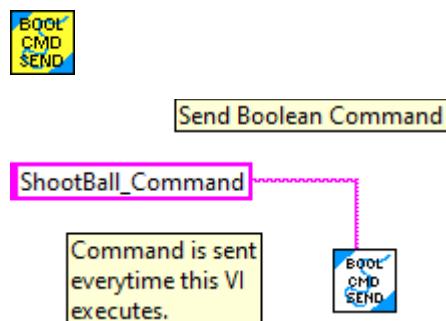
Add this loop inside the case structure that selects which auto routine to execute

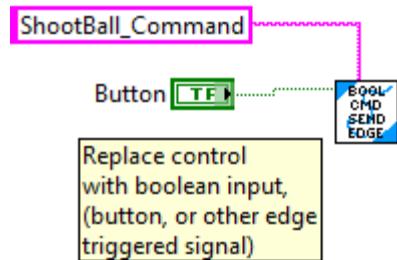
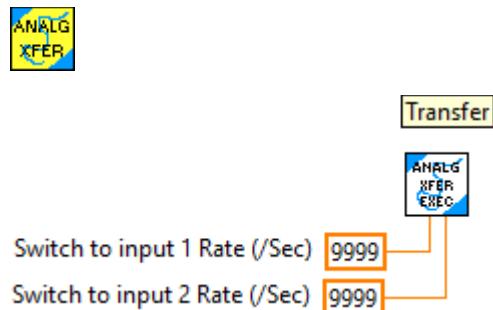


macro_BangBang_Execute



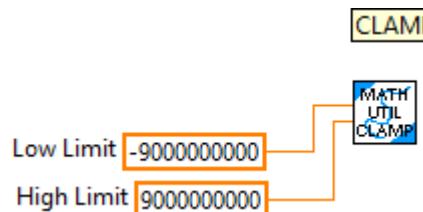
macro_BoolCmd_Recv**macro_BoolCmd_RecvMultiplexer****macro_BoolCmd_RecvMultiplexerArray**

**macro_BoolCmd_Send****macro_BoolCmd_Send_OnEdge**

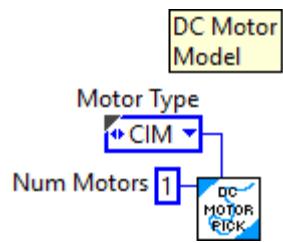
Boolean Edge On Send Command**macro_BumplessTransfer****macro_CallbackHelp_MatrixMinus****macro_CallbackHelp_MatrixMult**

macro_CallbackHelp_MatrixMult_CoerceSizeB

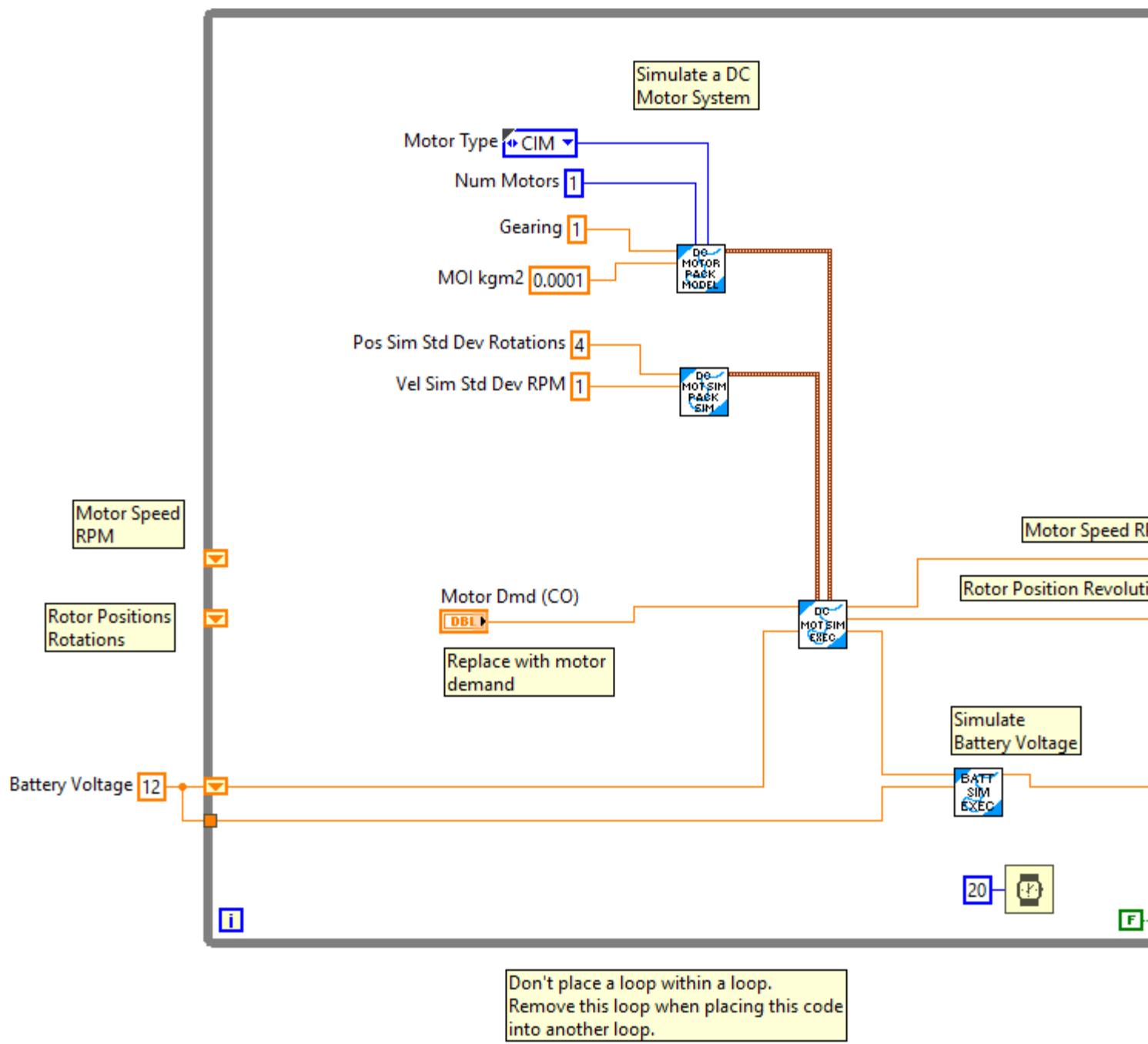
macro_CallbackHelp_MatrixPlus

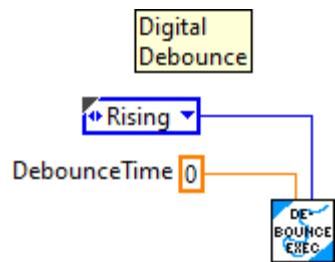
macro_Clamp

macro_DCMotor_Pick

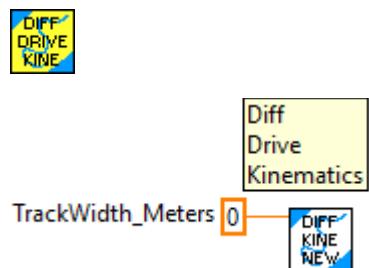


macro_DCMotor_Sim_Execute

**macro_Debouncer_Execute**

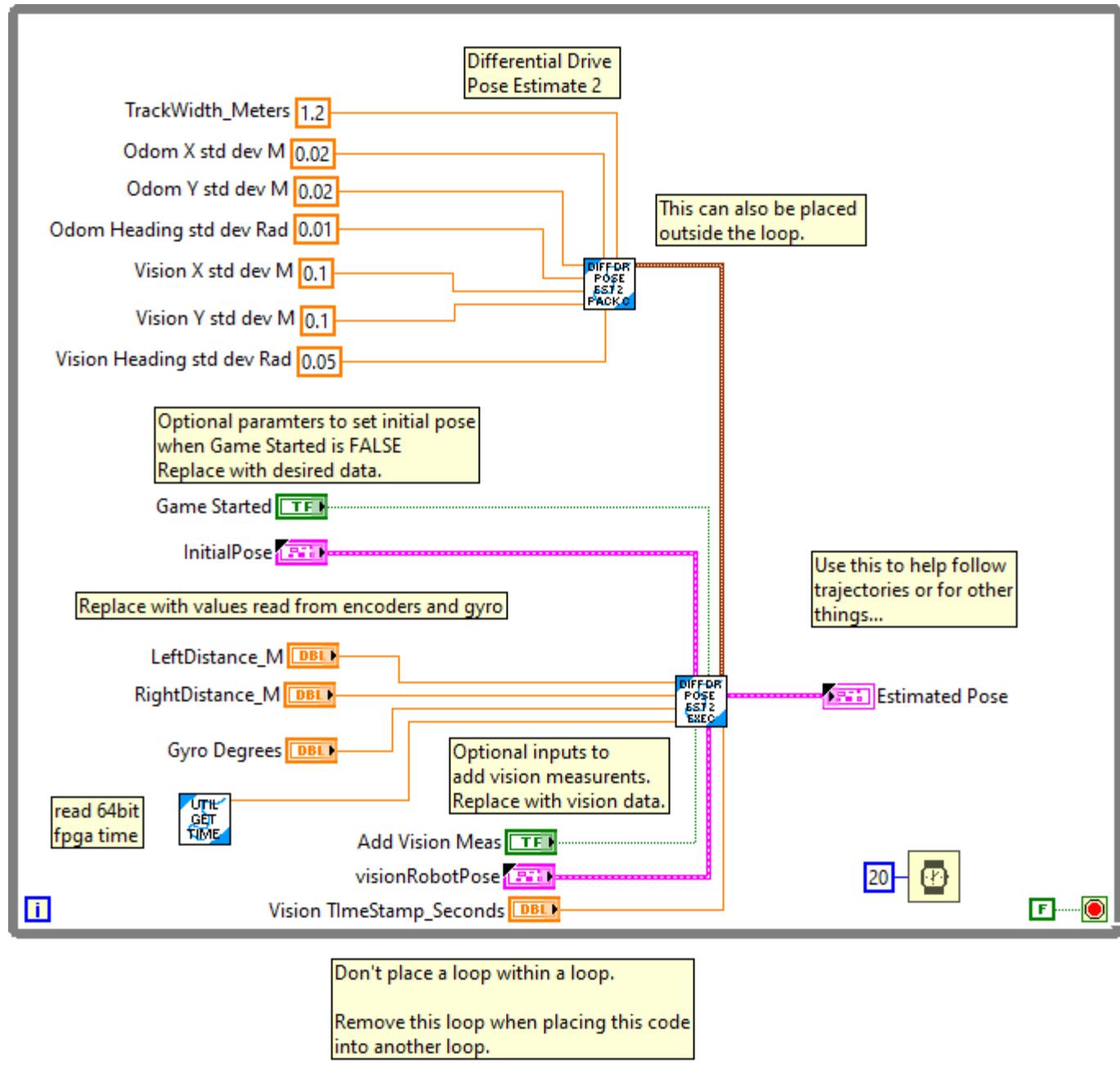


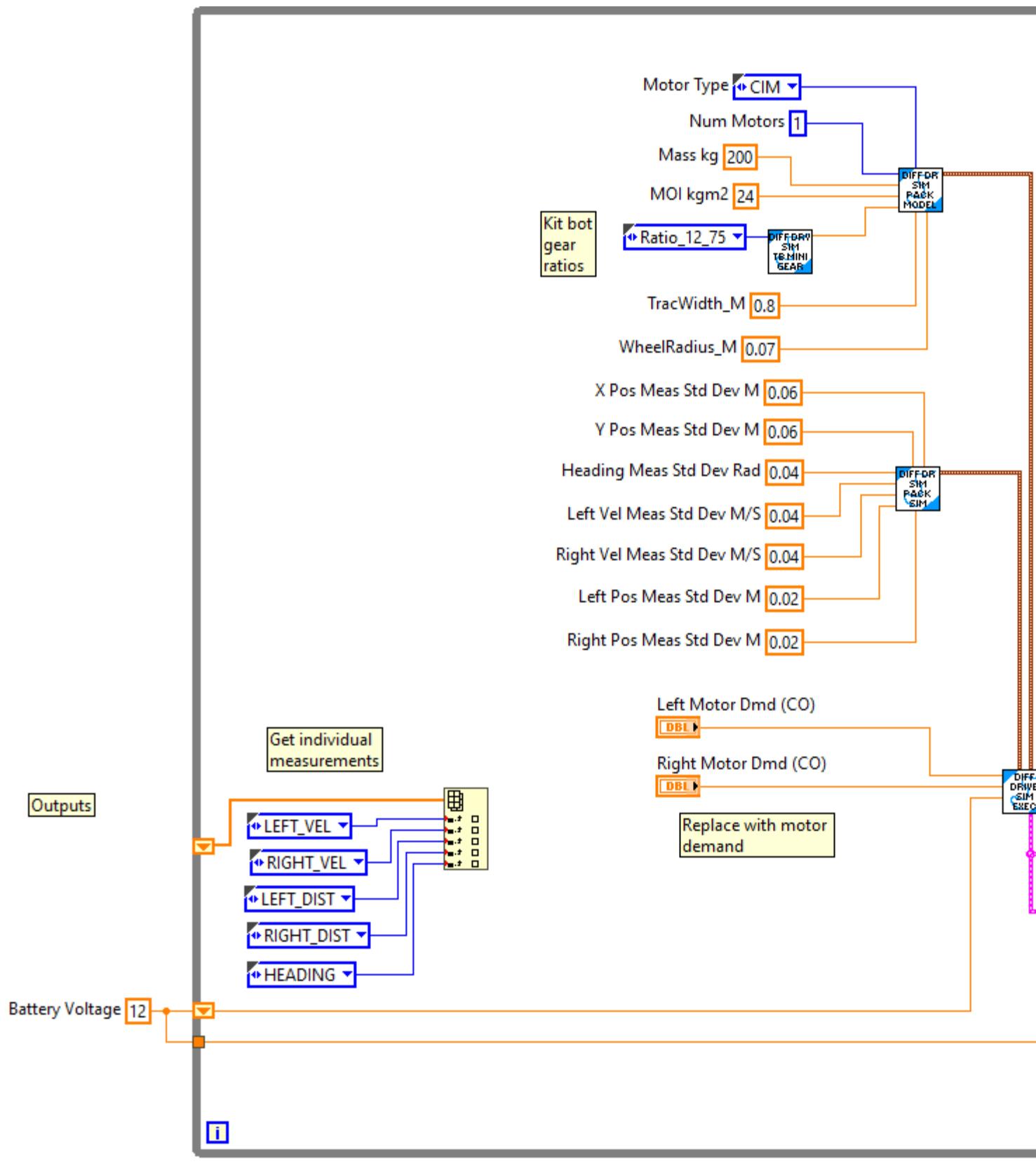
macro_DiffDriveKineNew



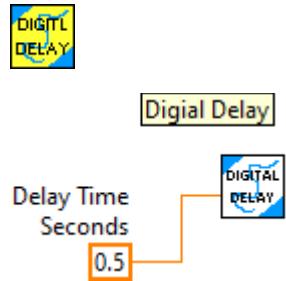
macro_DiffDrivePoseEst2_Execute



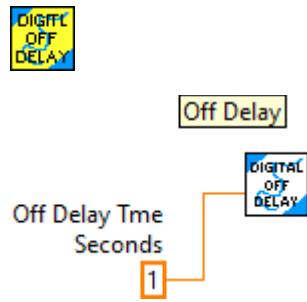
**macro_DiffDrv_Sim_Execute**



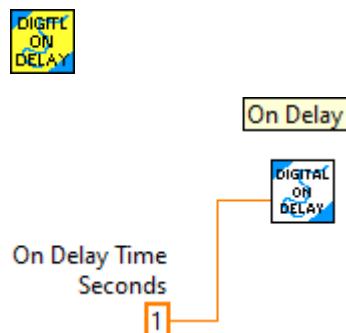
macro_DigSeqLogic_Delay

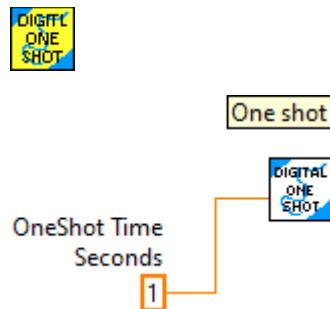
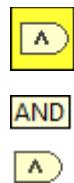
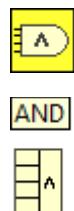
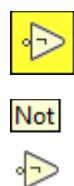


macro_DigSeqLogic_Off_Delay



macro_DigSeqLogic_On_Delay



macro_DigSeqLogic_OneShot**macro_Digital_And****macro_Digital_AndMany****macro_Digital_Not**

macro_Digital_Or

Or

**macro_Digital_OrMany**

Or

**macro_DoubleSolenoid_Pulse_Execute**

Double Solenoid Pulse

Flag from driver station indicating that robot is enabled. This will re-pulse the solenoid outputs to ensure that they match the Solenoid Demand.

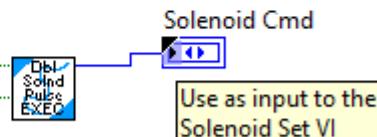
Robot Enabled

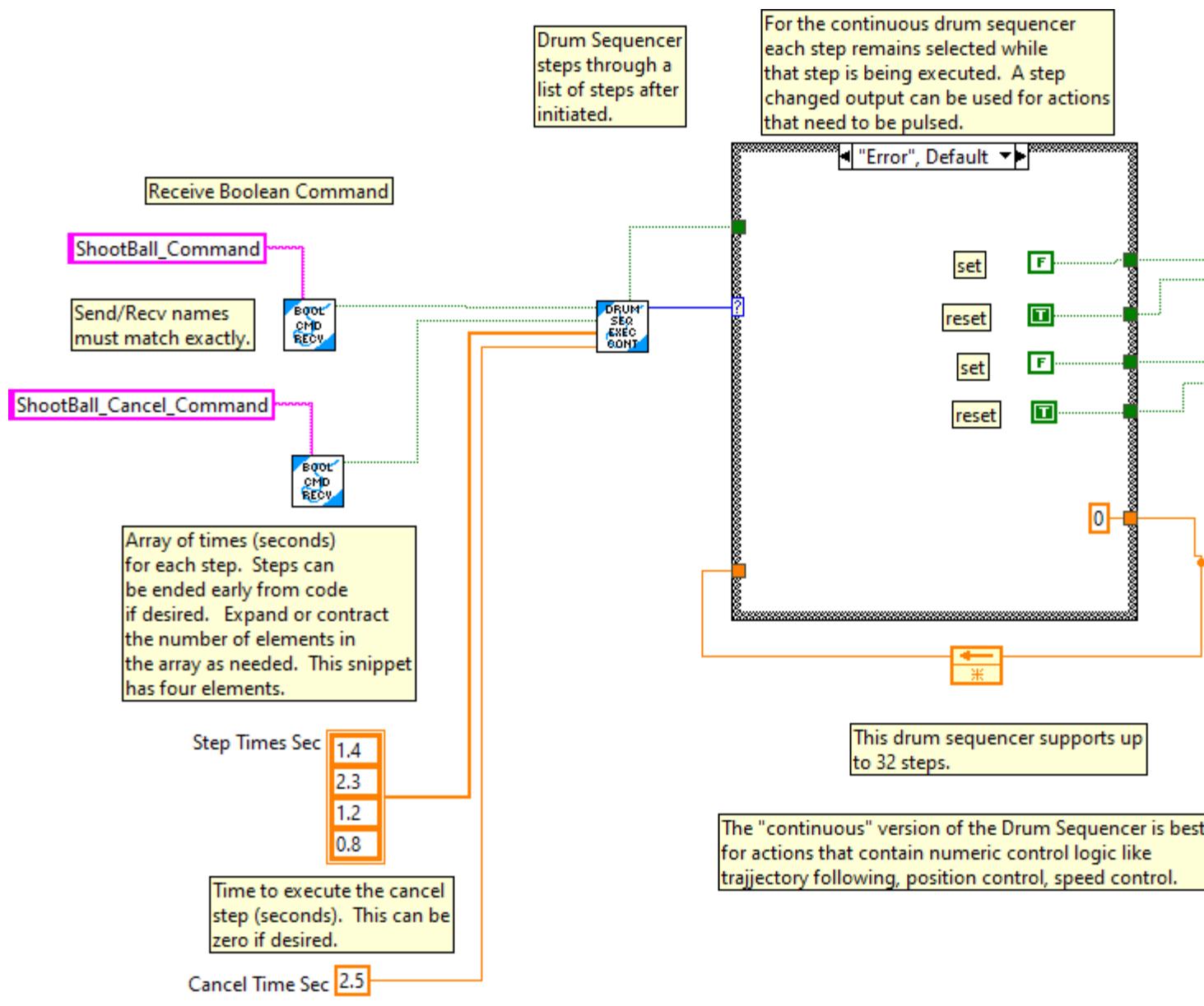


Solenoid Demand



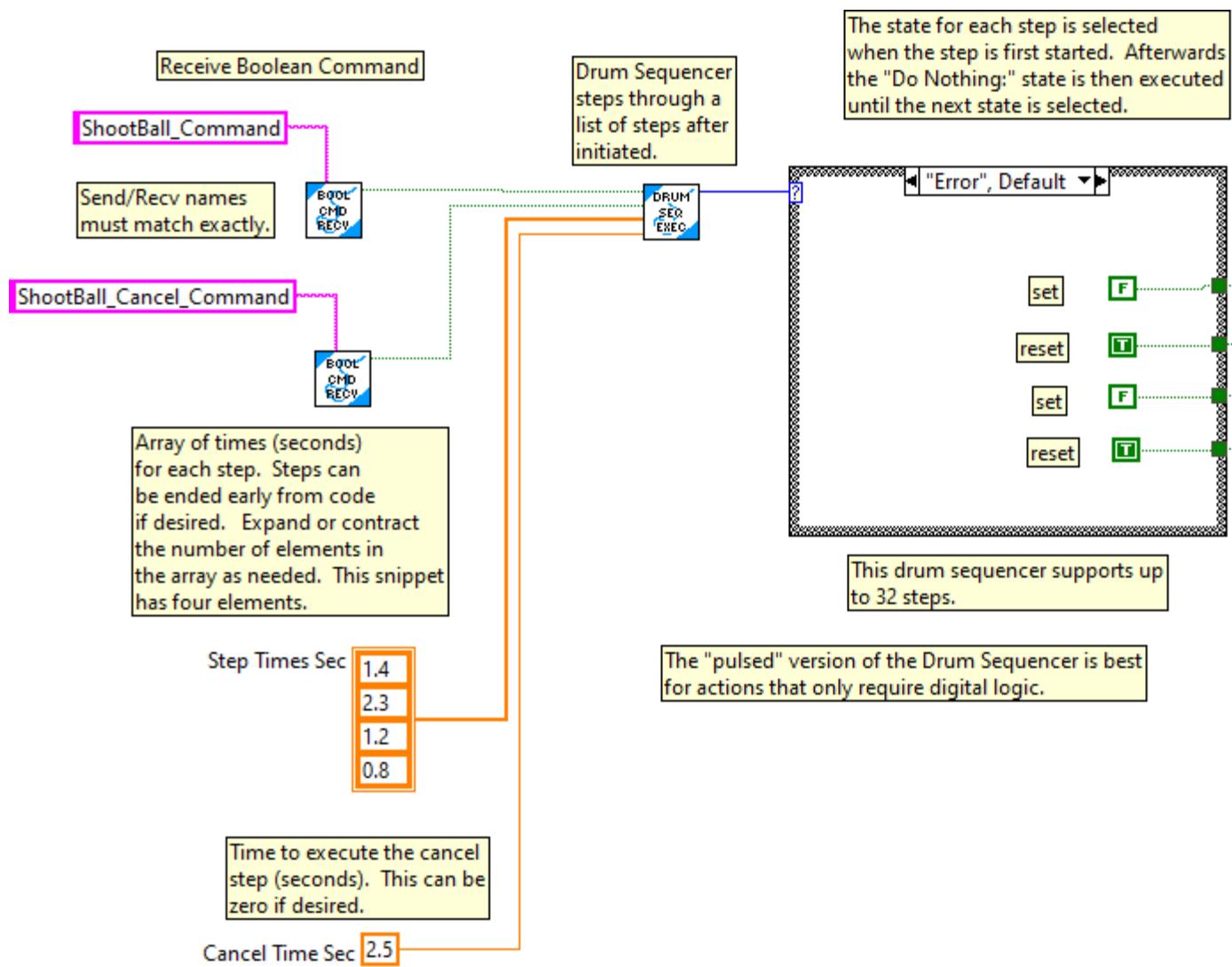
Boolean indicating desired solenoid output

**macro_DrumSeq_Continuous**



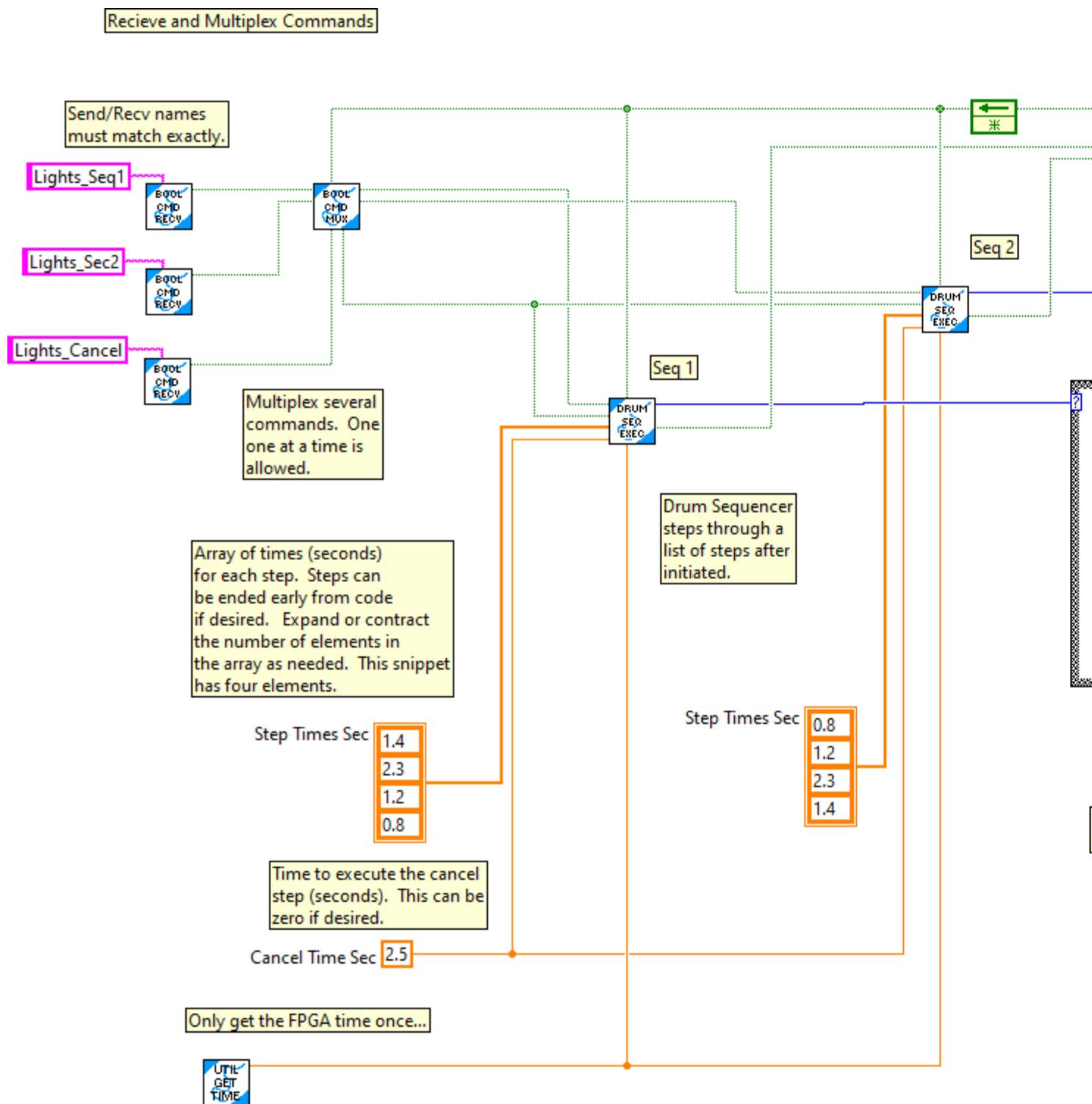
macro_DrumSeq_Pulse

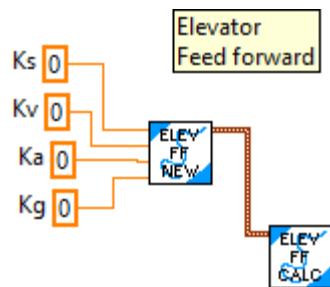




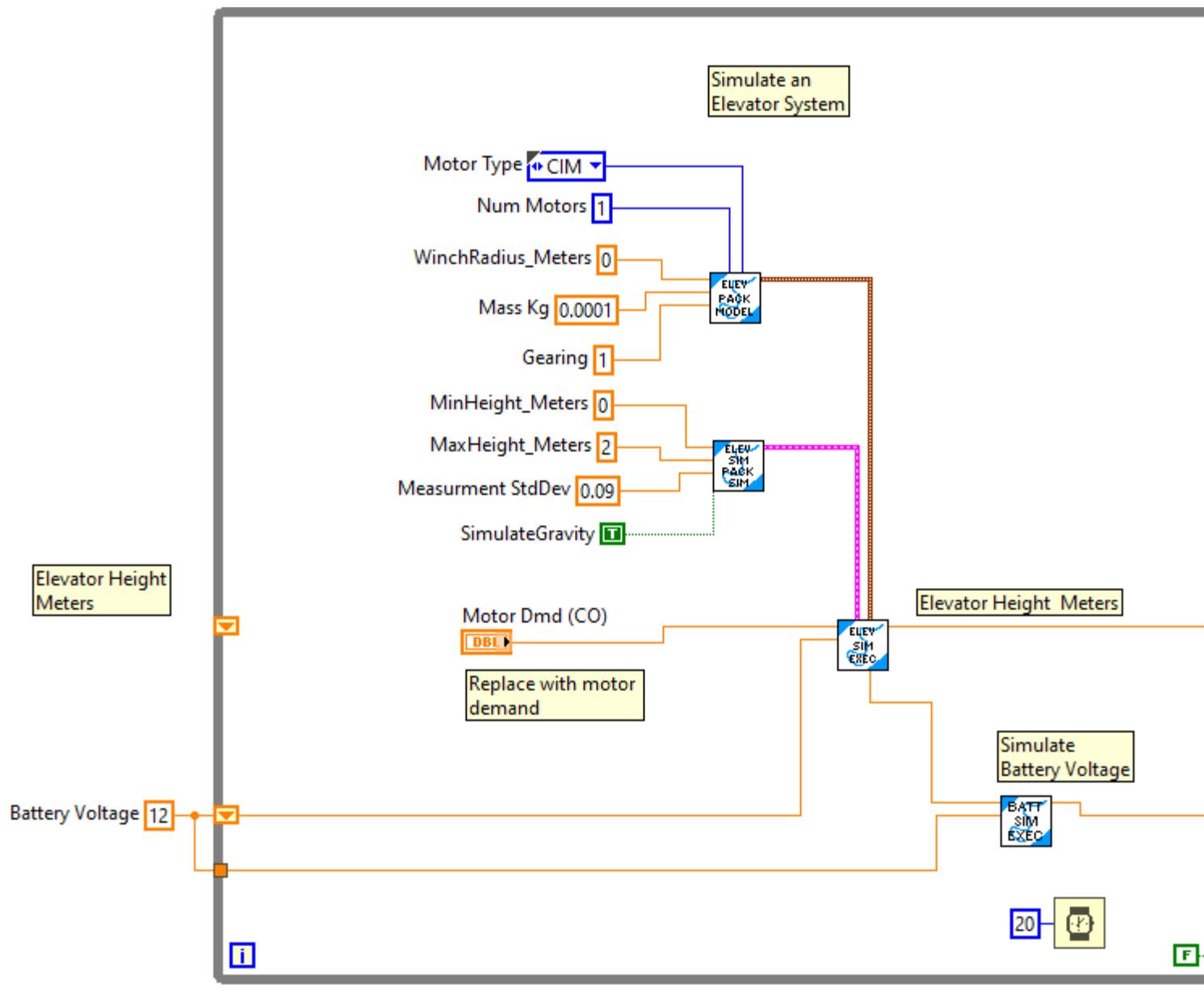
macro_DrumSeq_Pulse_Mux



**macro_ElevatorFF_Calculate**

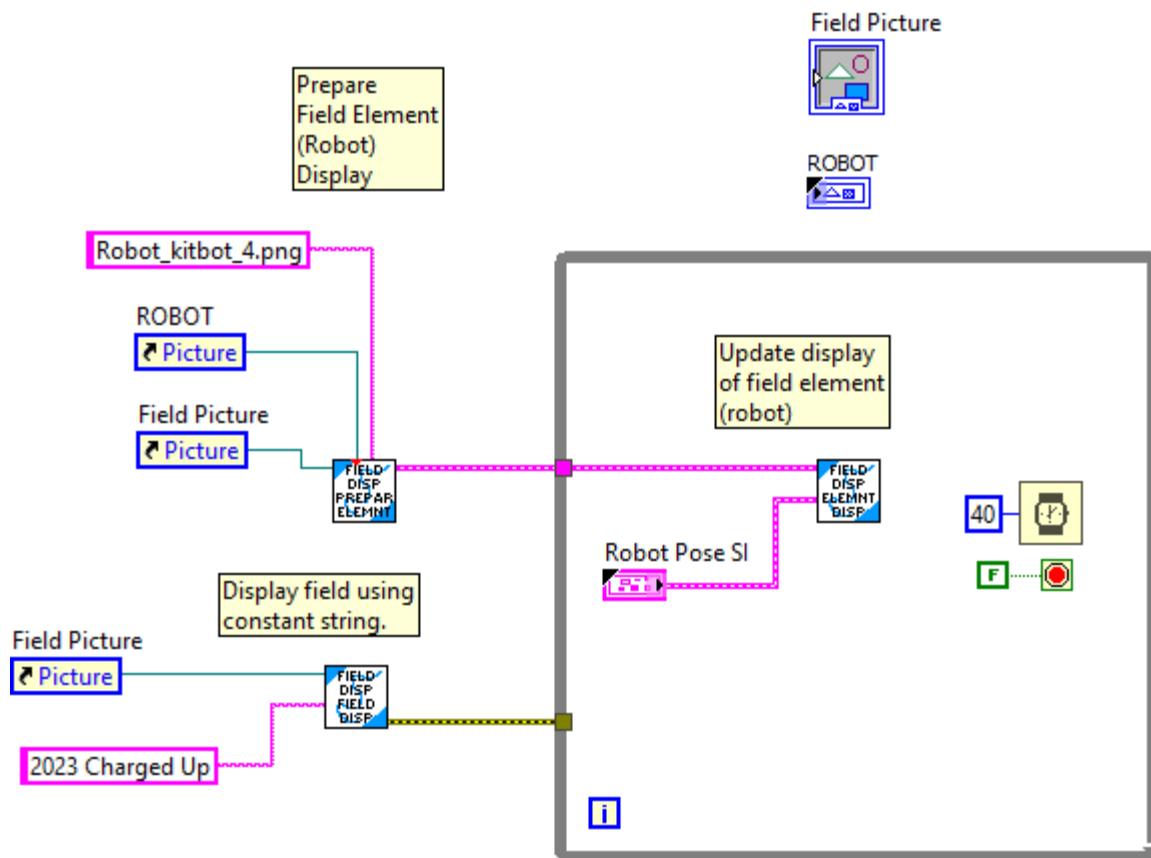


macro_Elevator_Sim_Execute



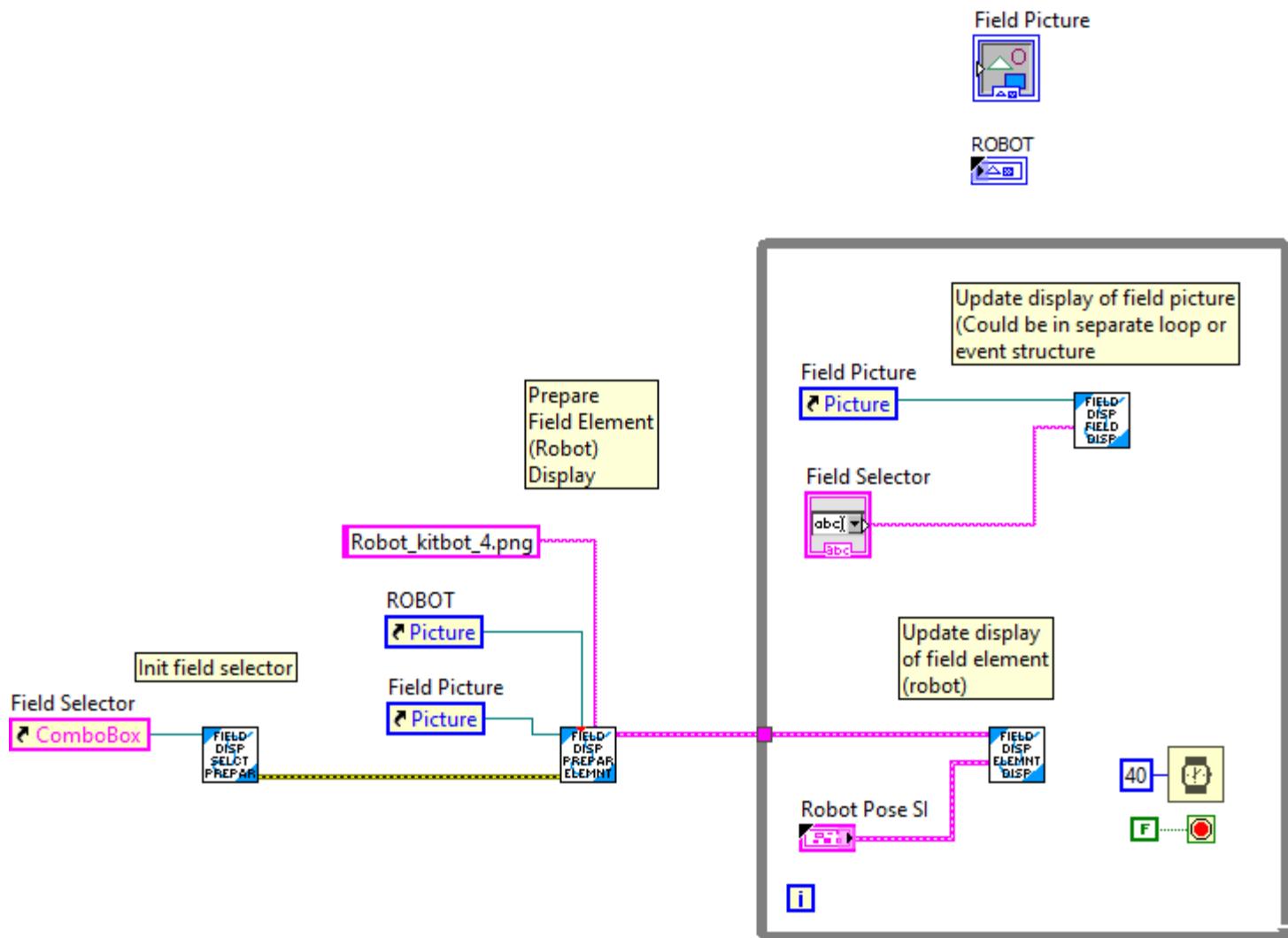
macro_Field_Display_Constant





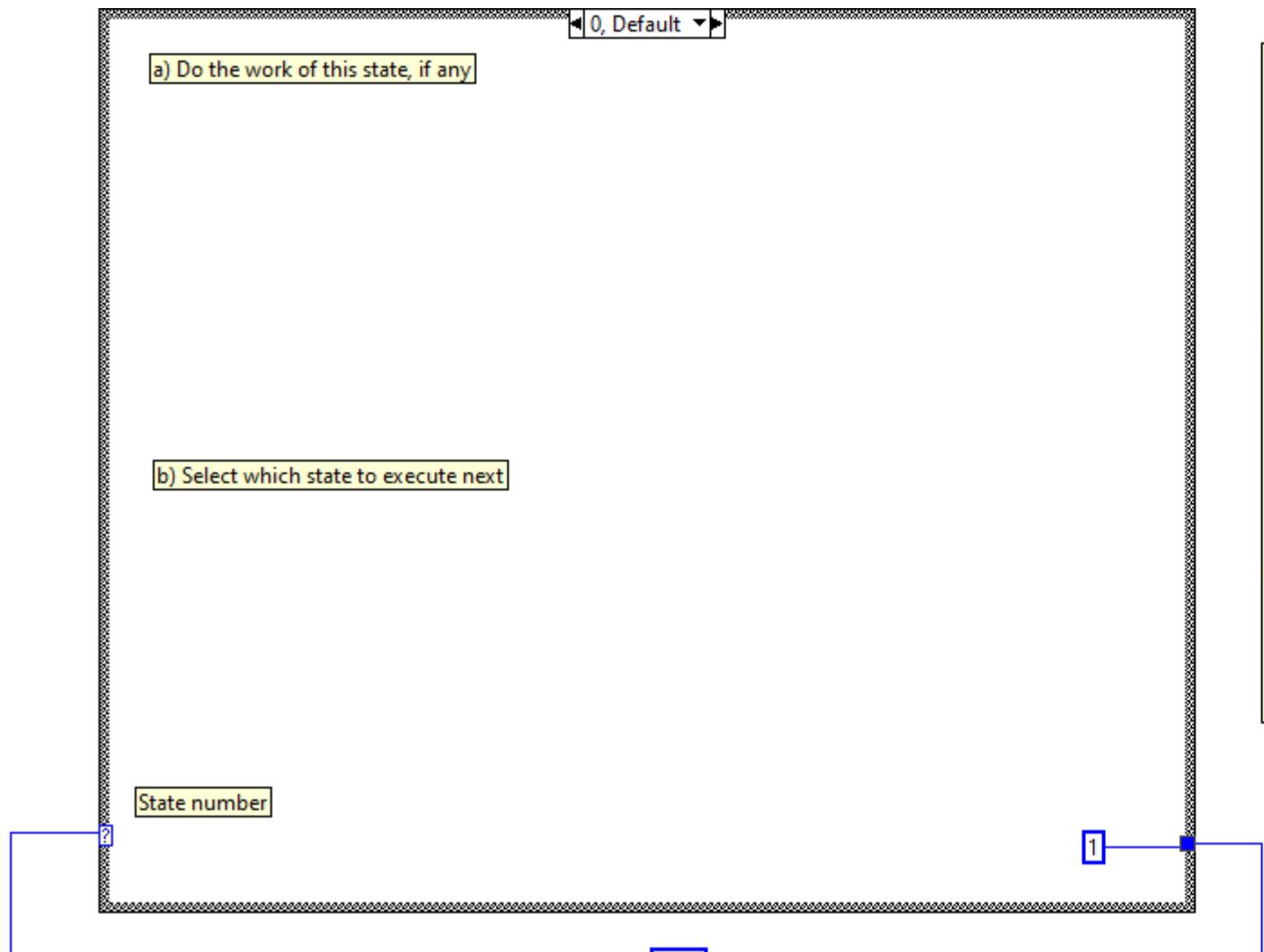
macro_Field_Display_Selector





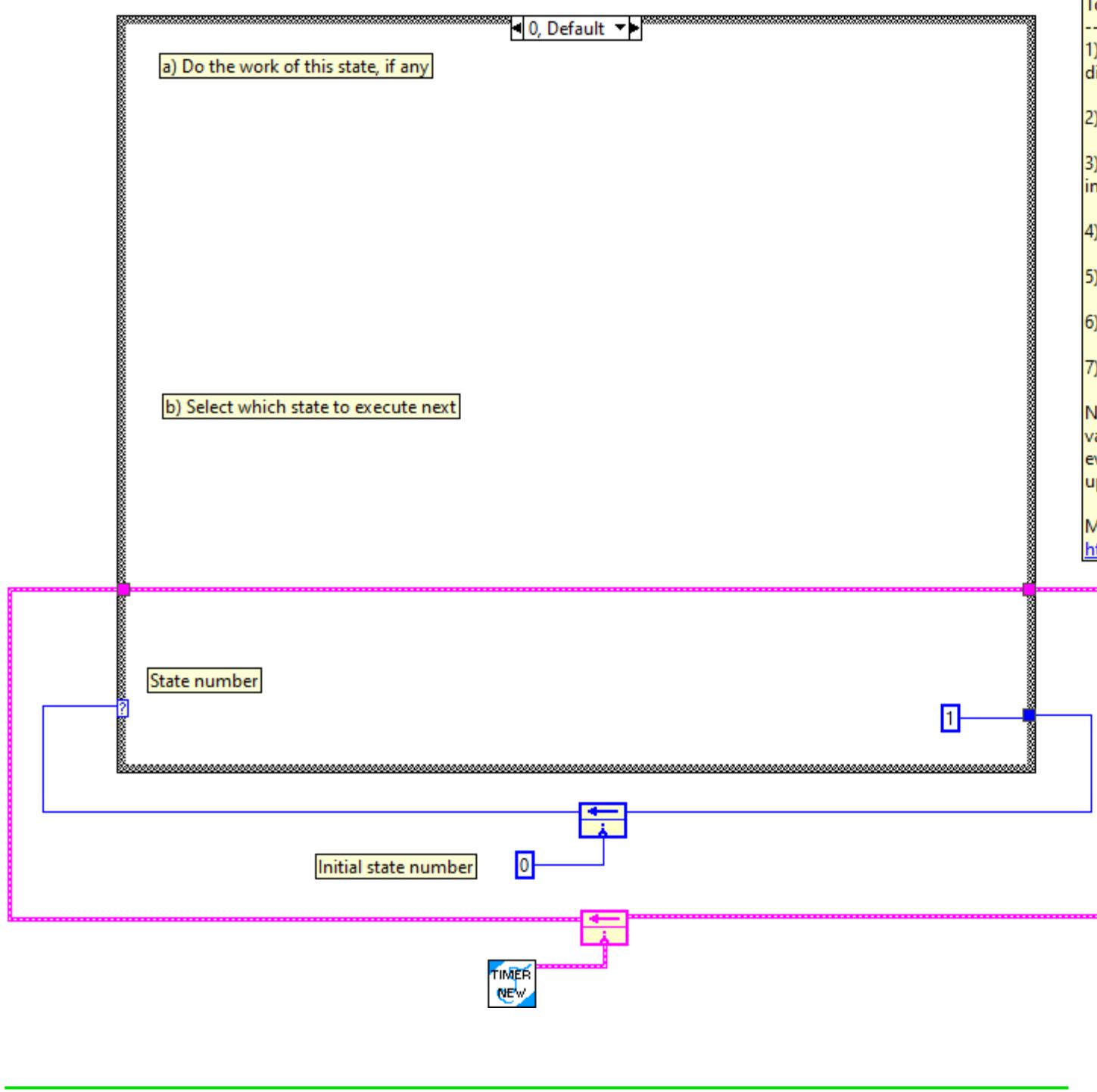
macro_FiniteStateMachine_Template





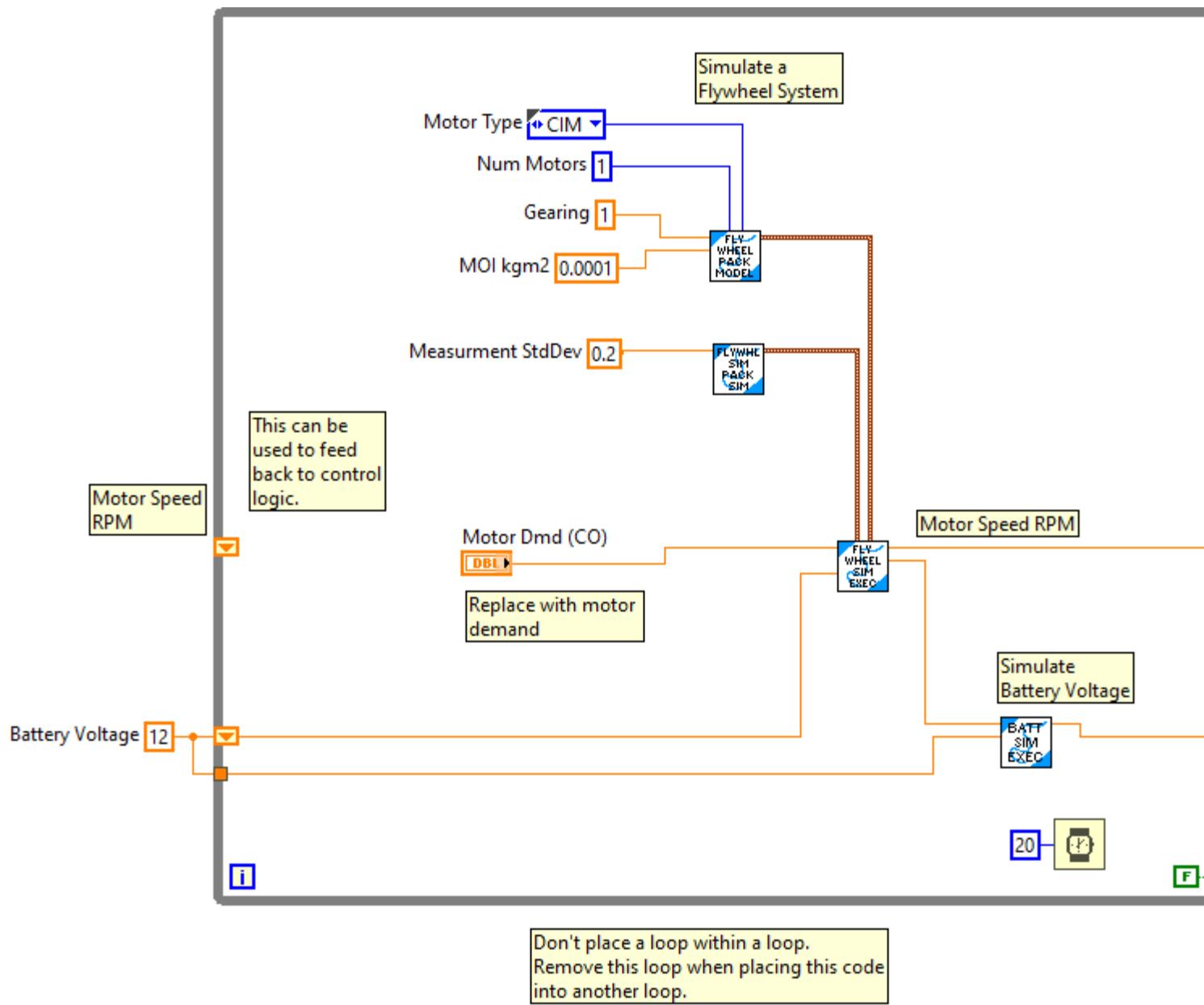
macro_FiniteStateMachine_w_Timer_Template





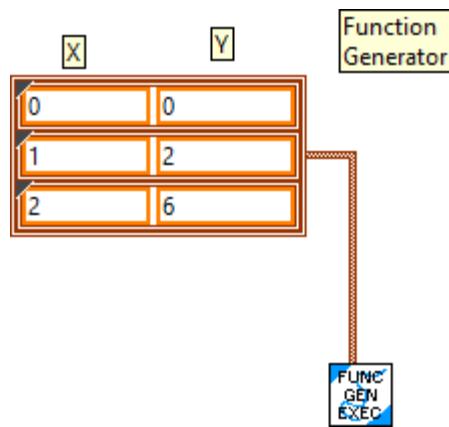
macro_FlyWheel_Sim_Execute



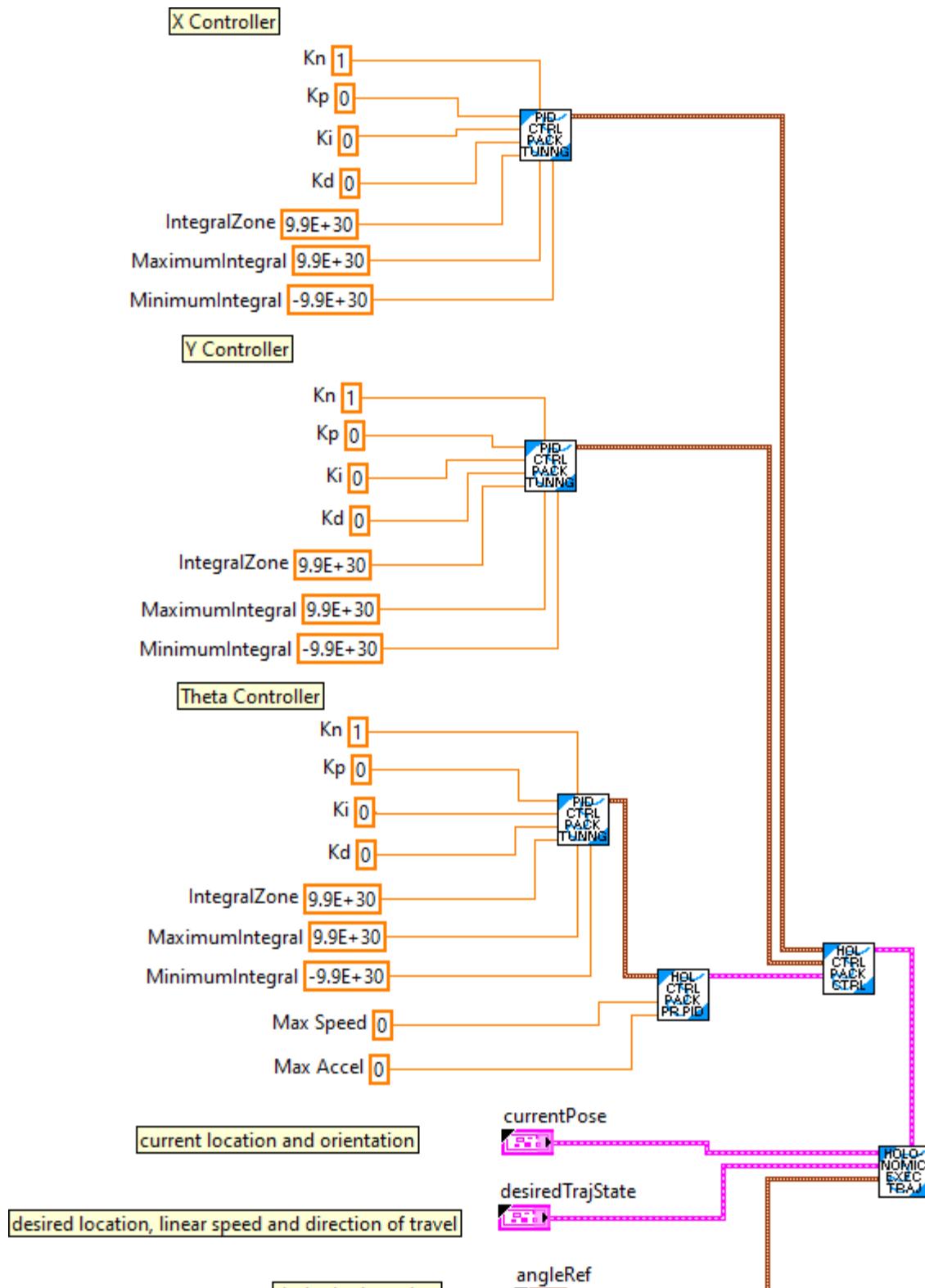


macro_Function_Generator

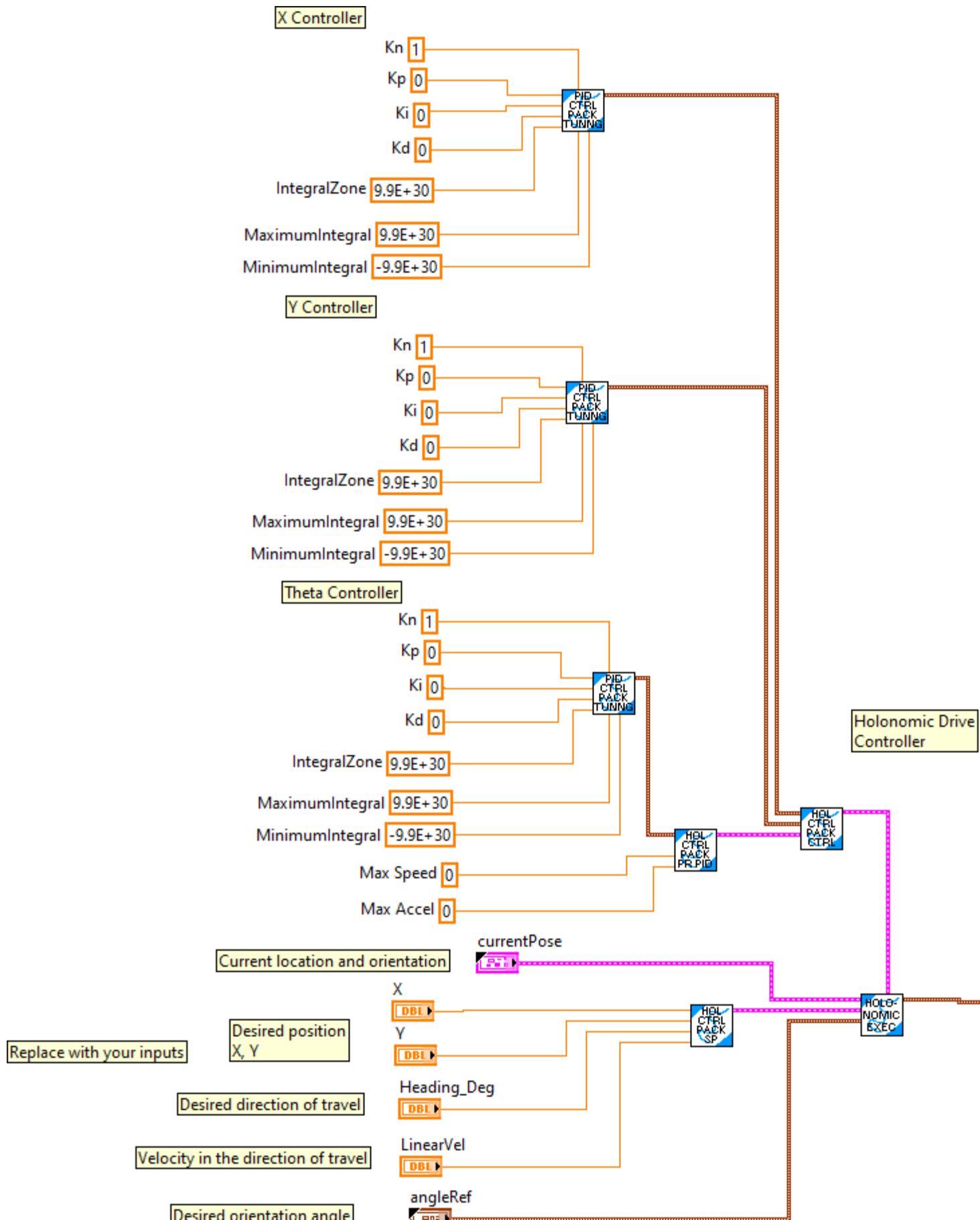


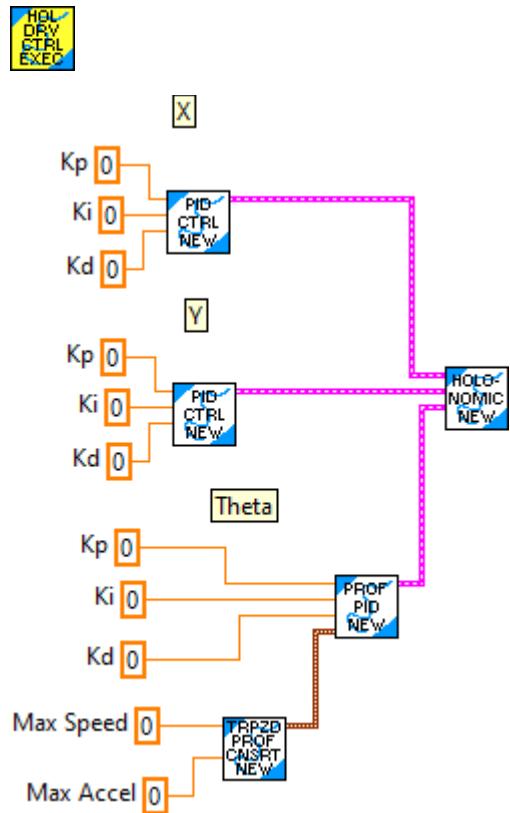
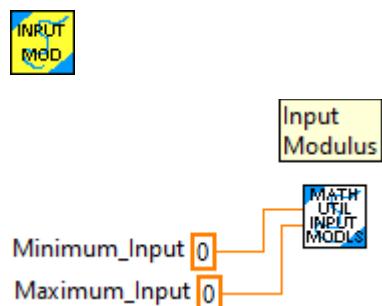


macro_HolDrvController_Execute

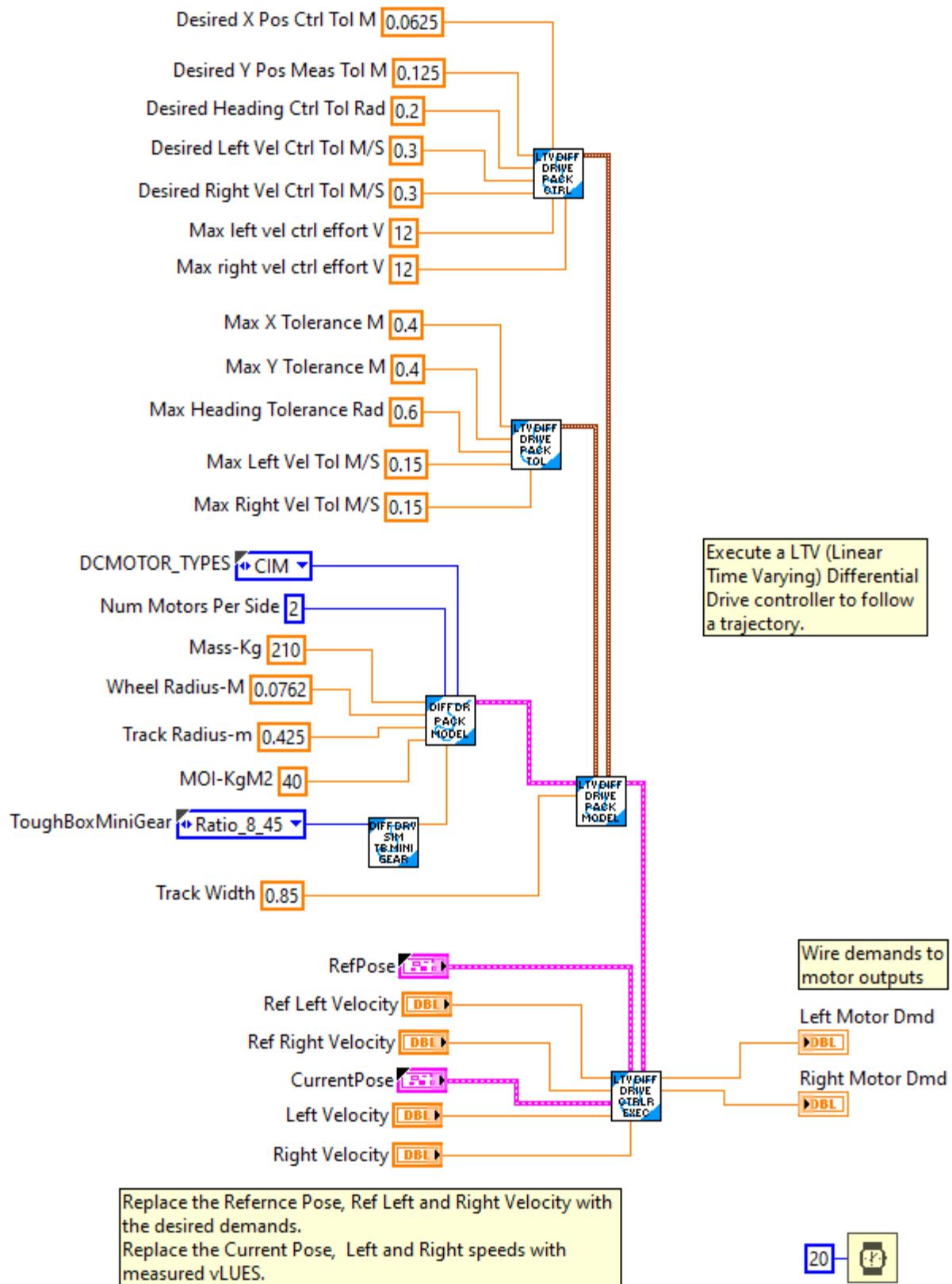


macro_HolDrvController_ExecuteSP

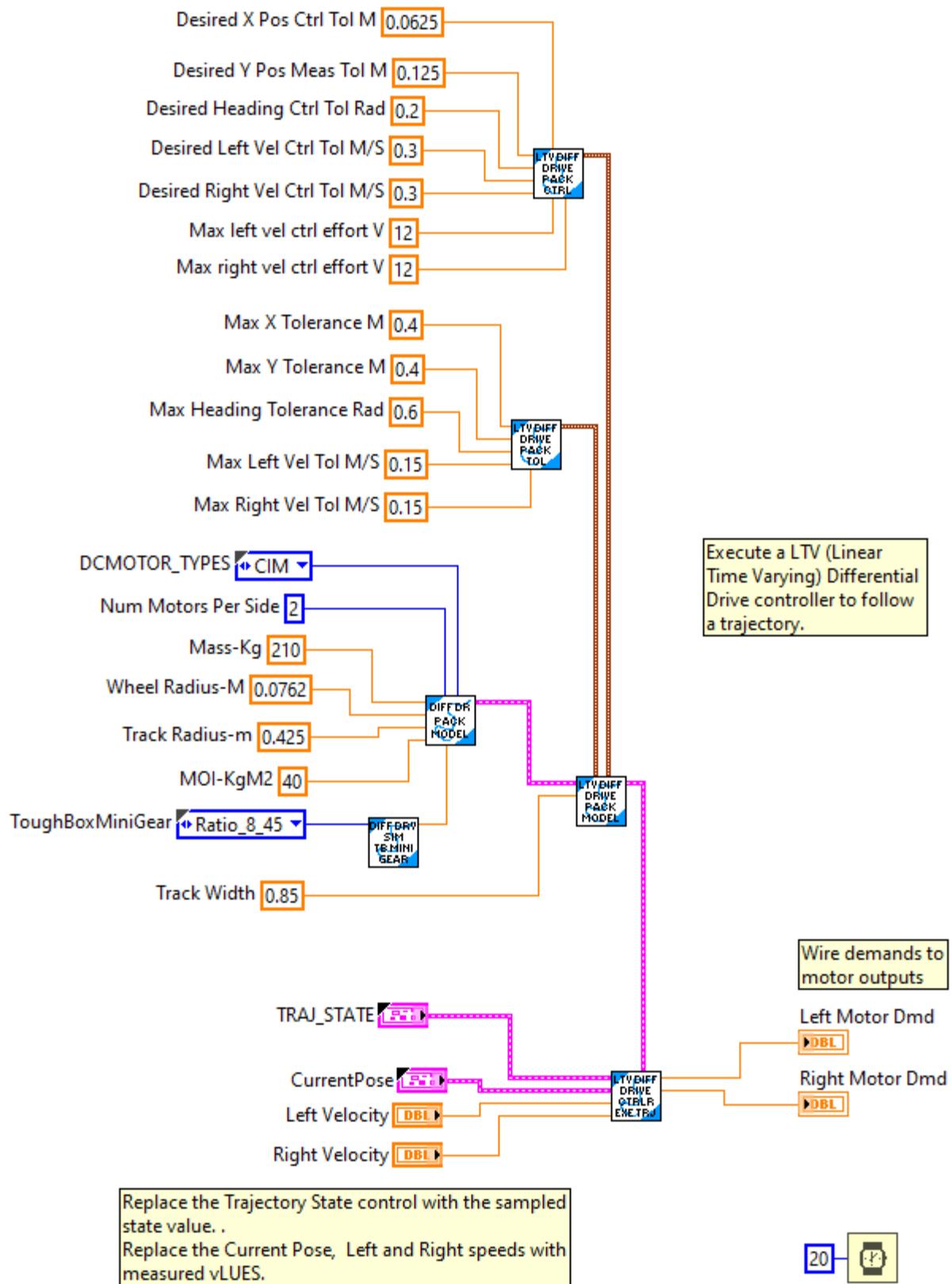


macro_HolDrvCtrl_New**macro_Input_Modulus**

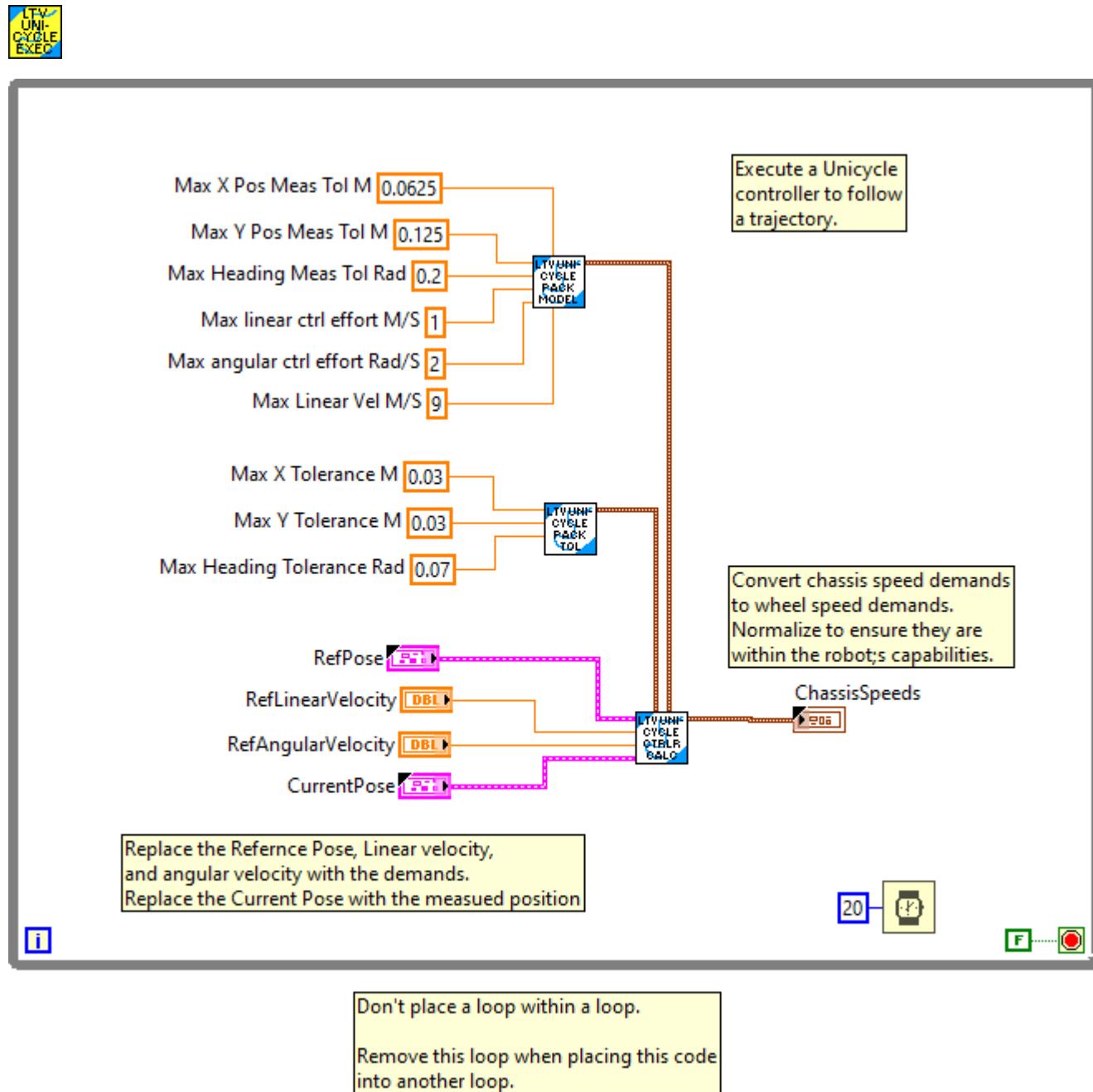
macro_LTV_DiffDriveCtrl_Execute



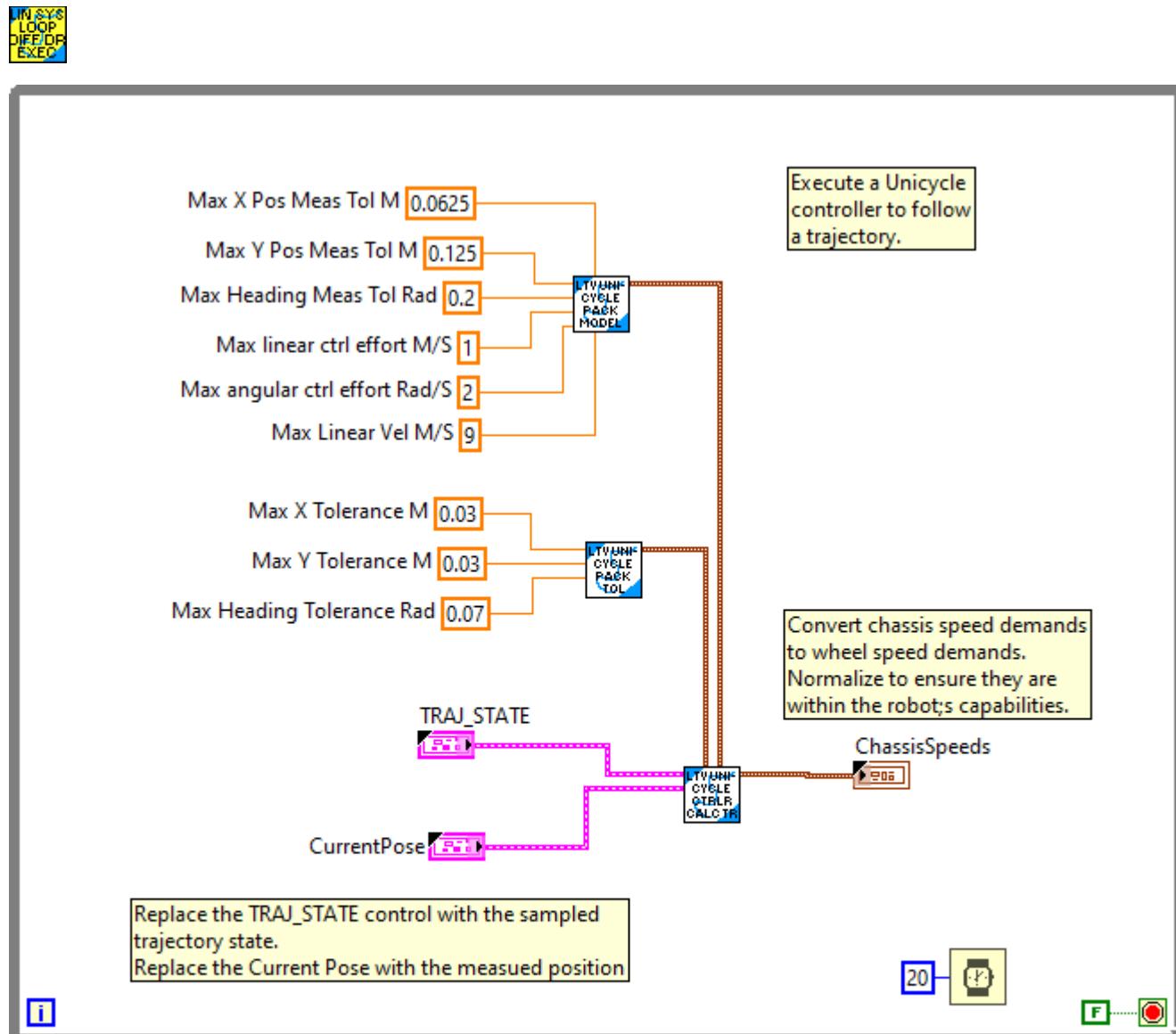
macro_LTV_DiffDriveCtrl_Traj_Sample_Execute



macro_LTV_Uncicycle_Ctrl_Execute

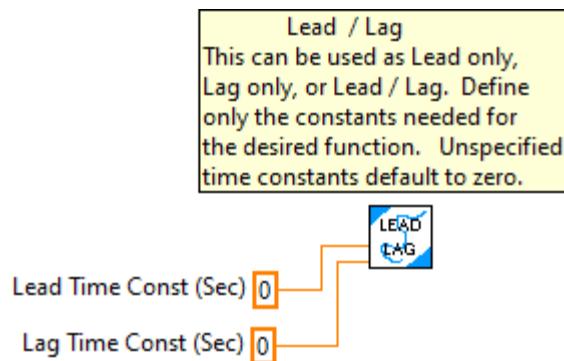


macro_LTV_Unicycle_Ctrl_TrajSample_Execute

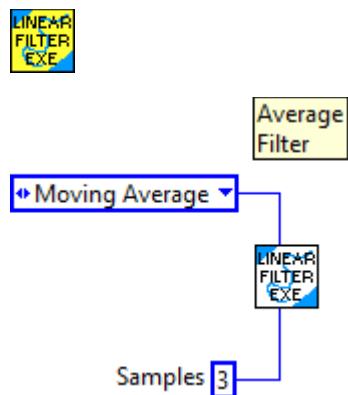


macro_LeadLag

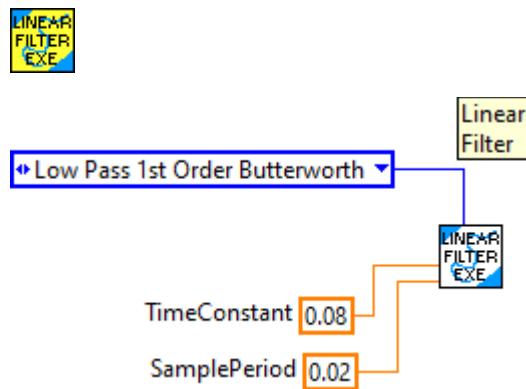




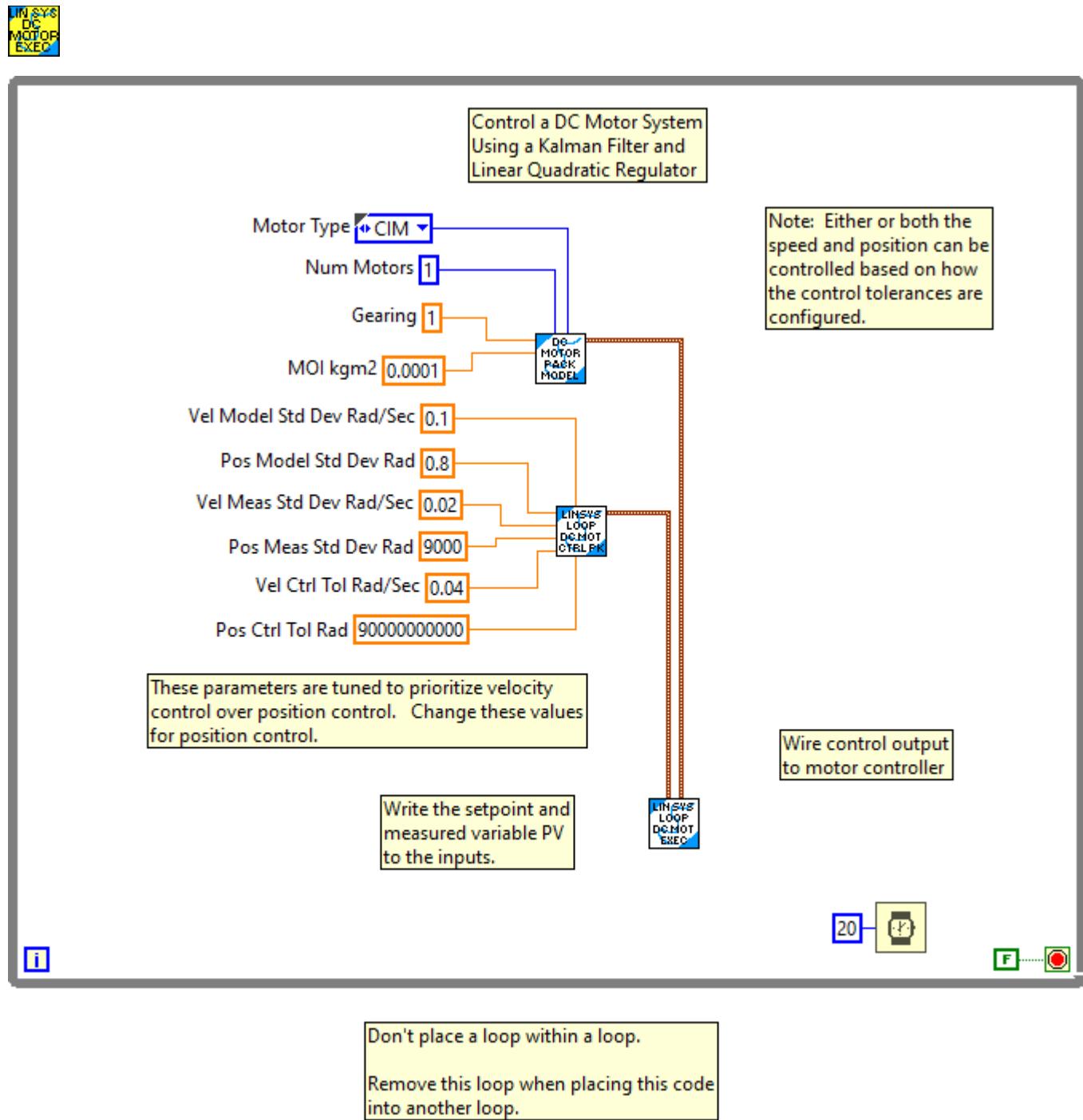
macro_LinearFilter_ExecuteAVG



macro_LinearFilter_ExecuteTC

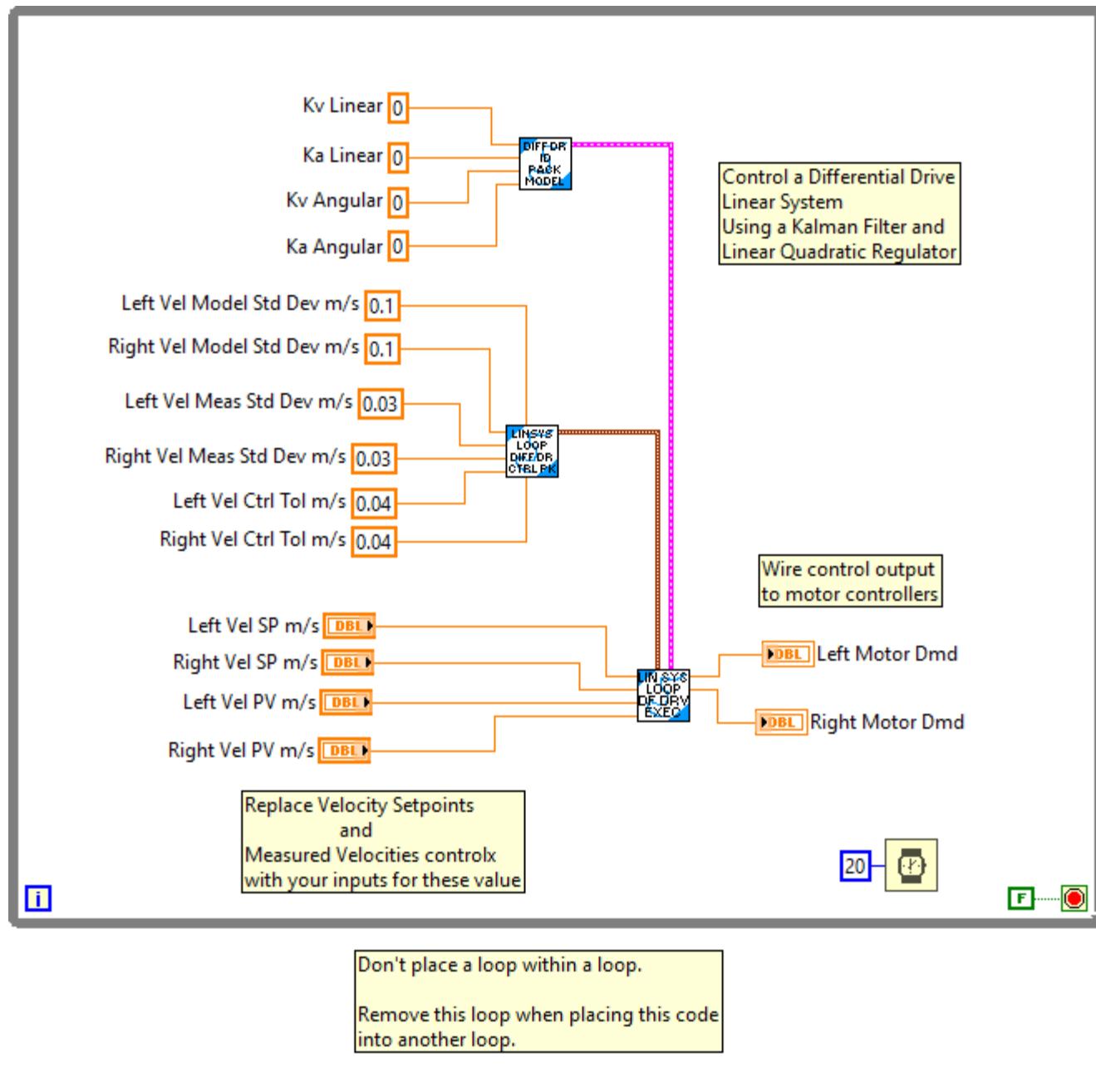


macro_LinearSystemLoop_DCMotor_Execute



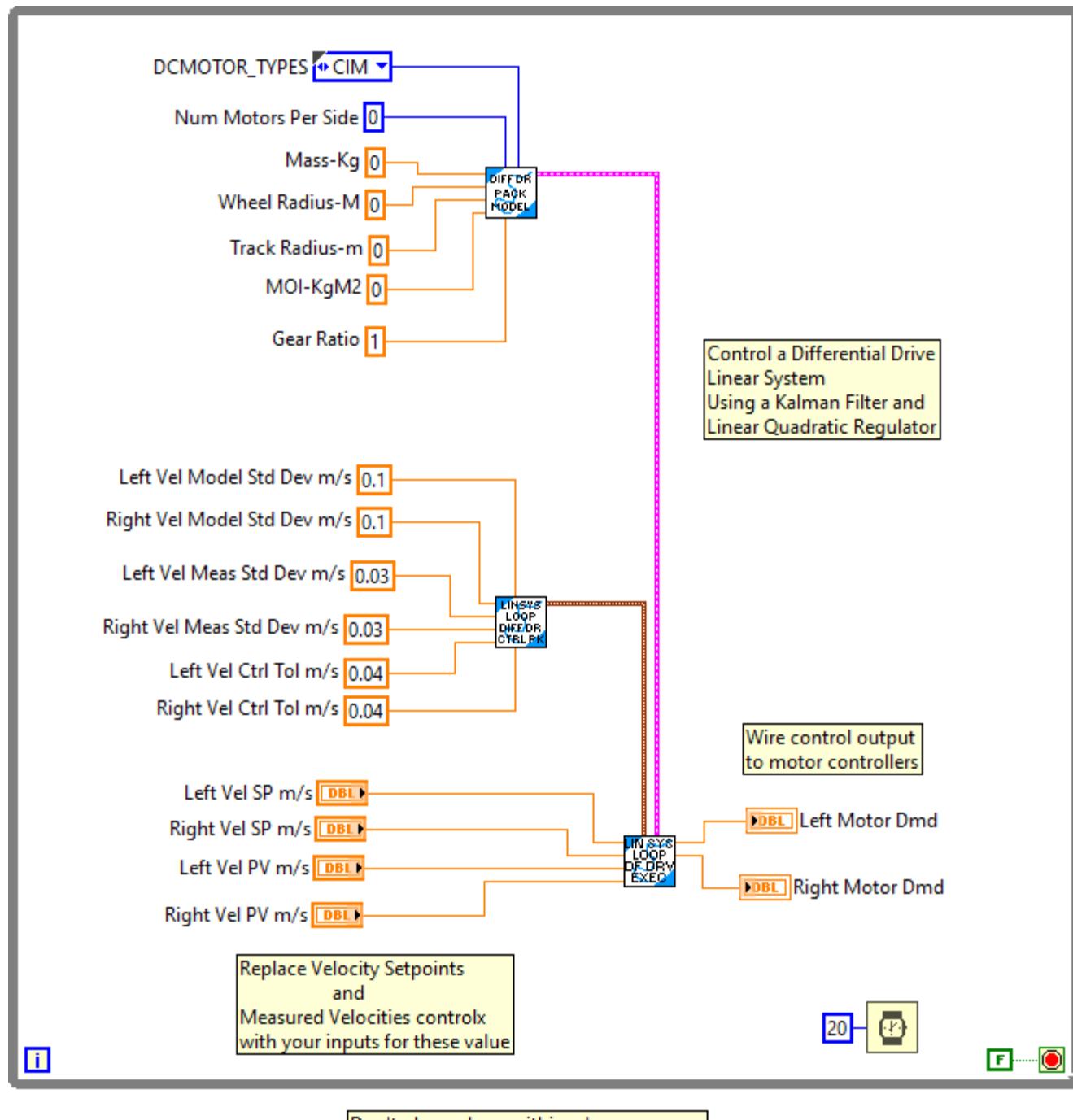
macro_LinearSystemLoop_DiffDrv_ID_Execute





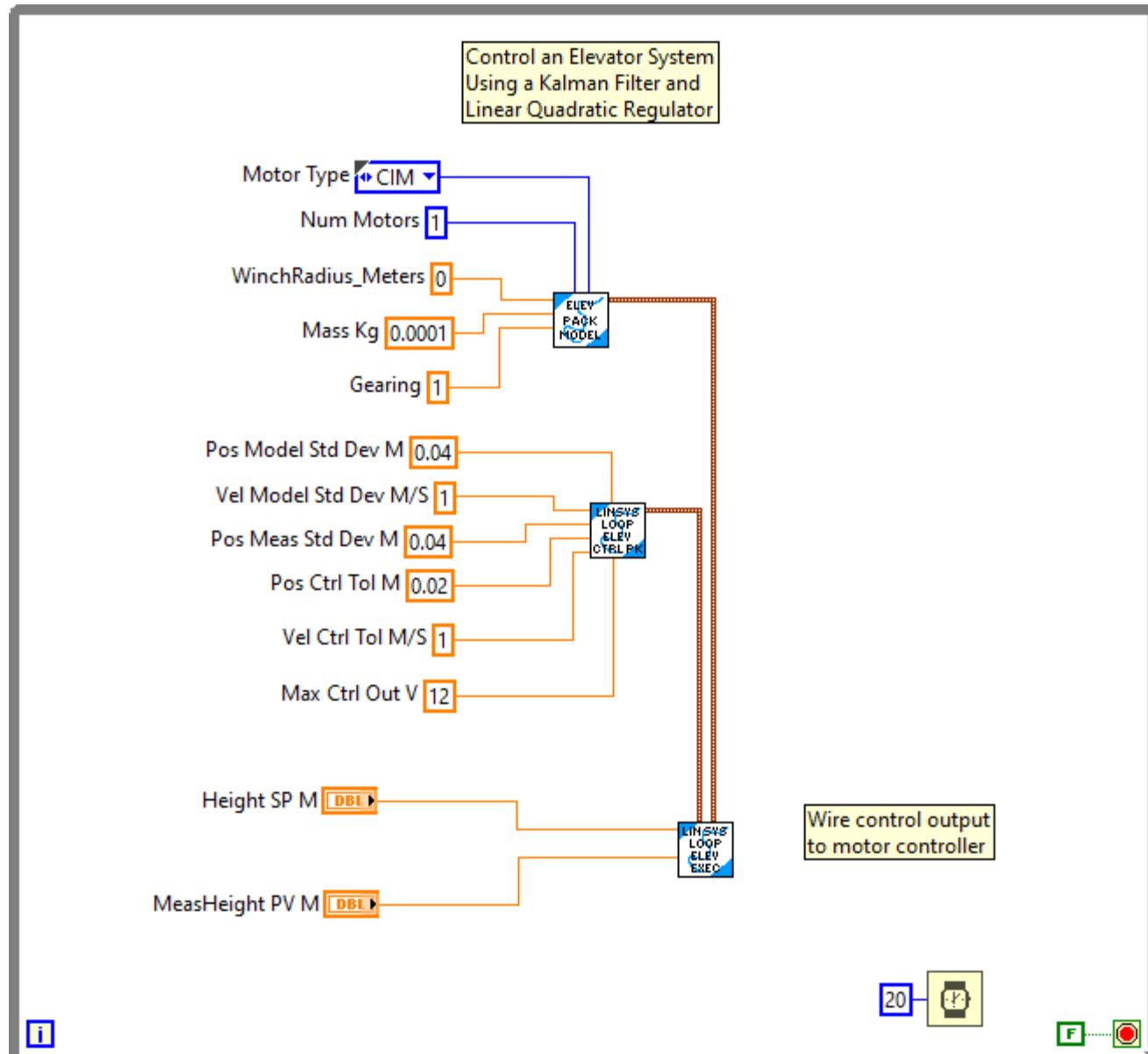
macro_LinearSystemLoop_DiffDrv_Model_Execute





macro_LinearSystemLoop_Elevator_Execute

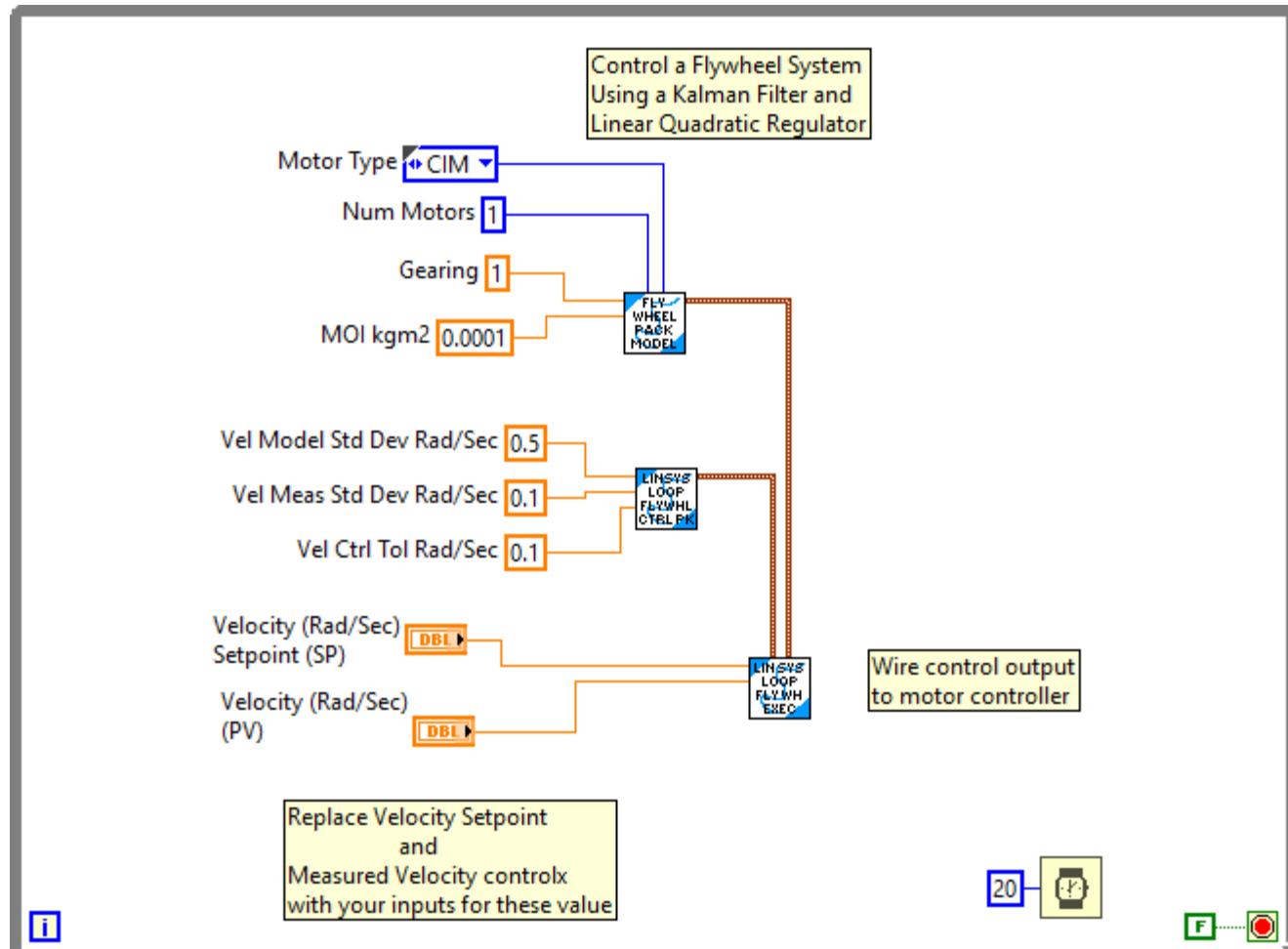




Don't place a loop within a loop.
Remove this loop when placing this code into another loop.

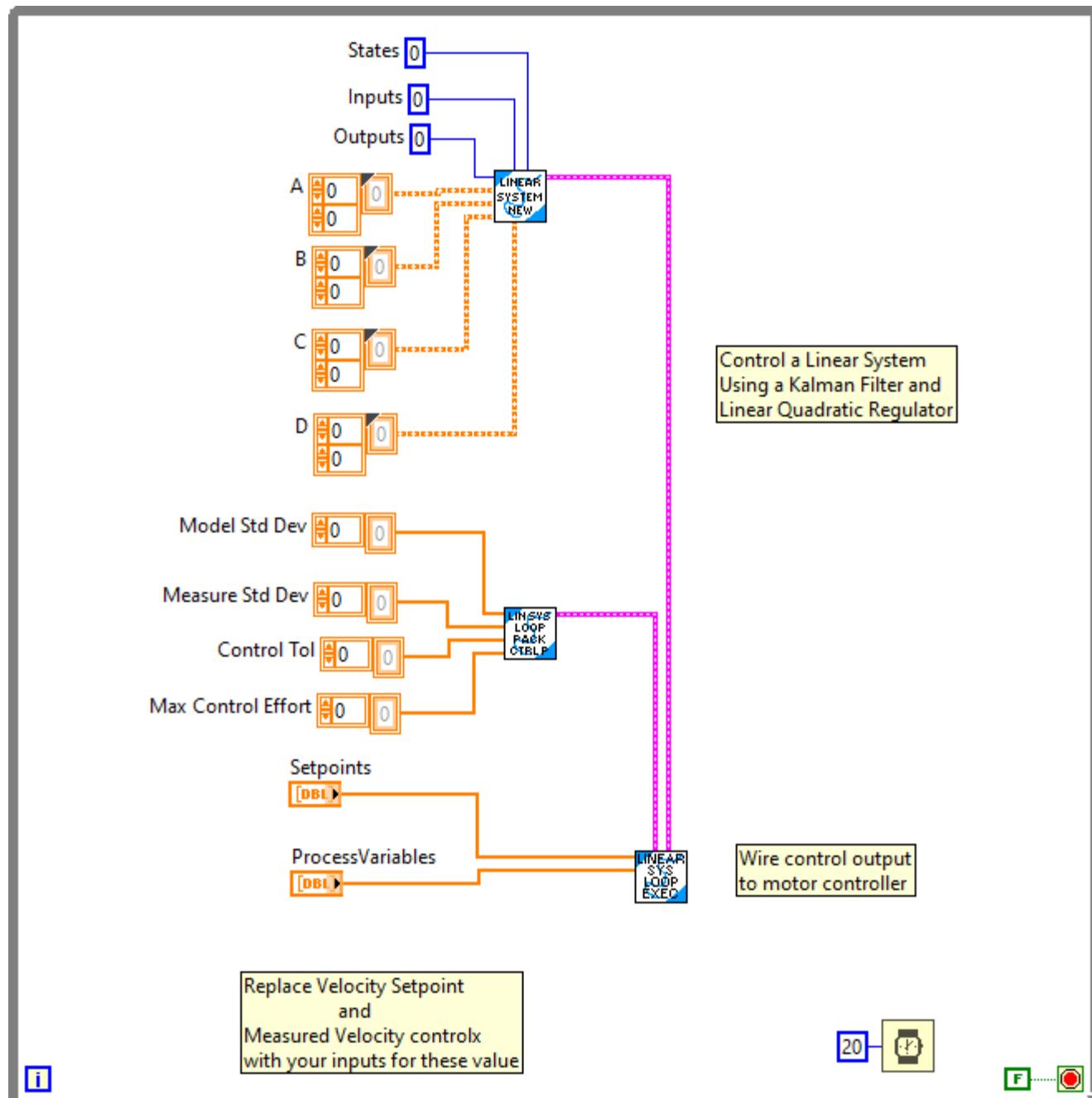
macro_LinearSystemLoop_FlyWheel_Execute





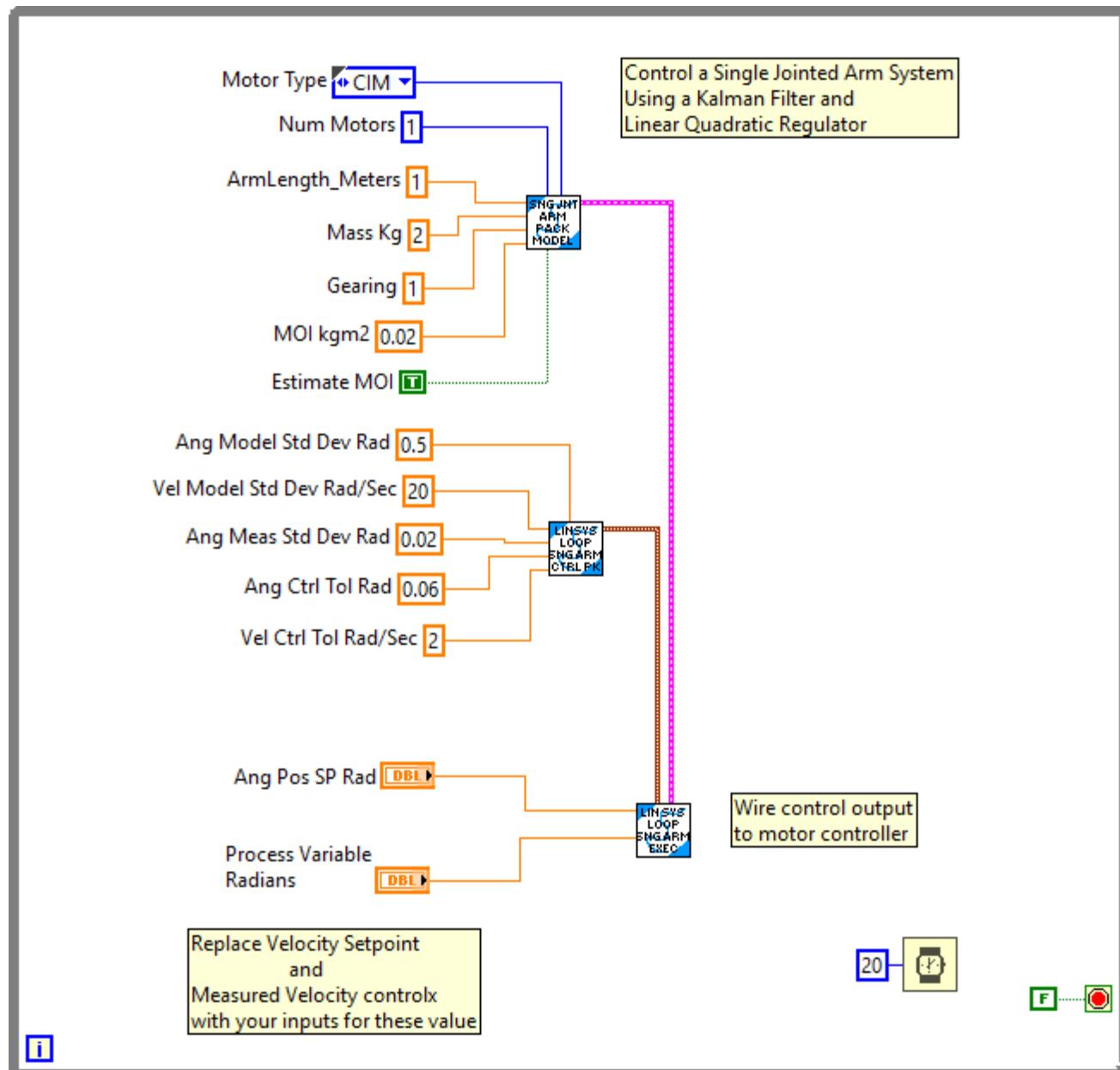
macro_LinearSystemLoop_LinearSystem_Execute





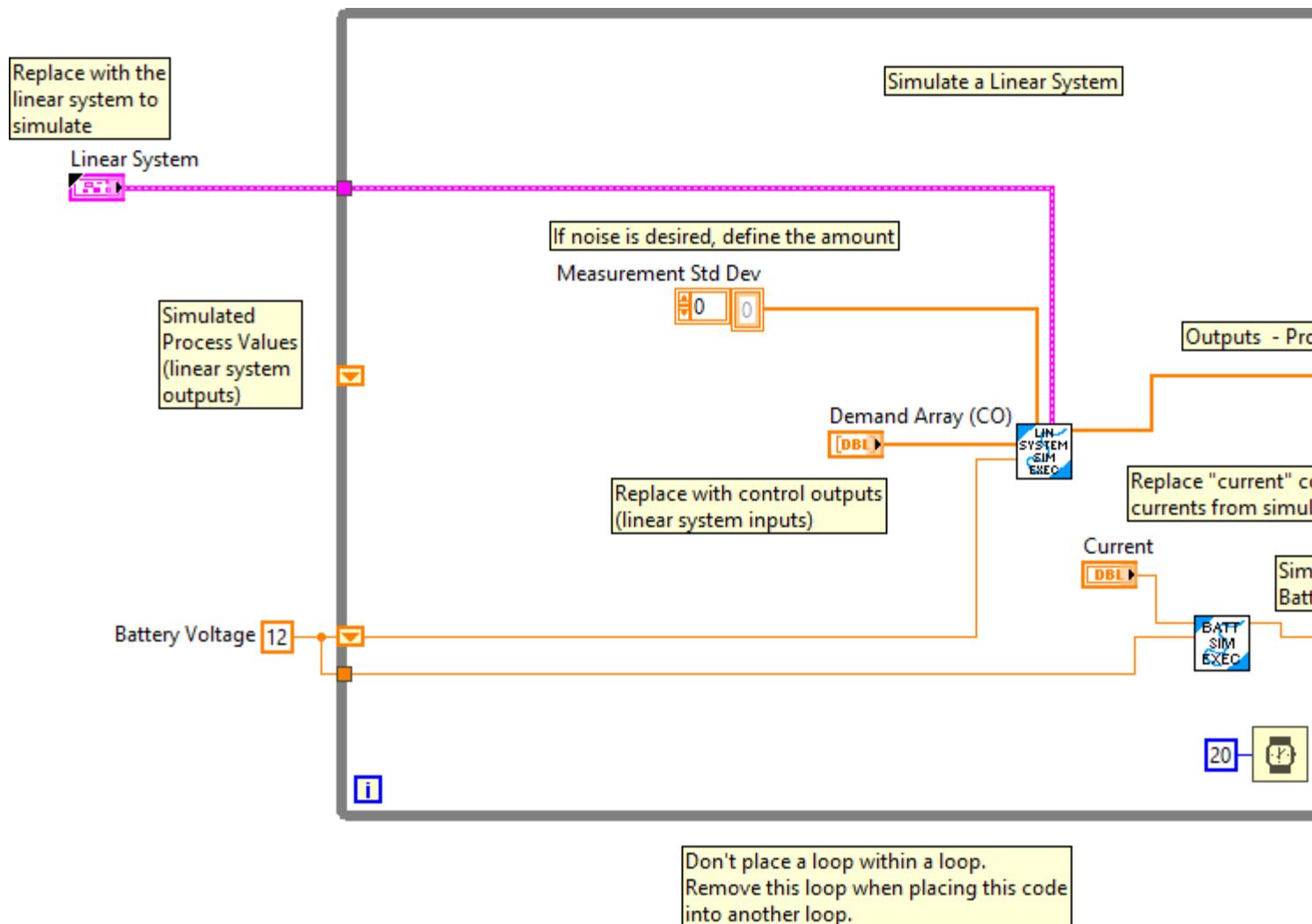
macro_LinearSystemLoop_SngJntArm_Execute



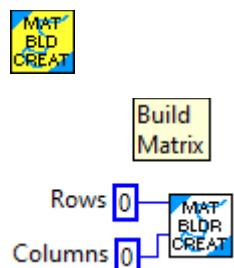


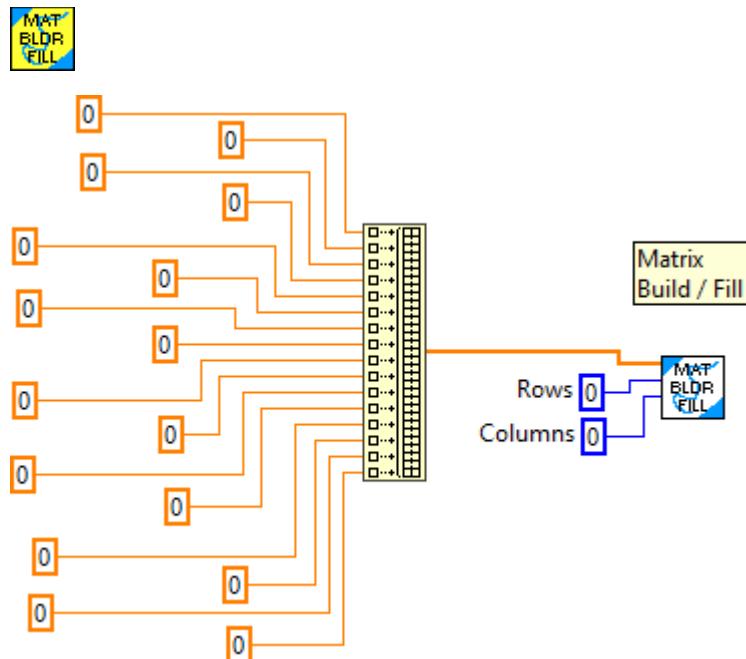
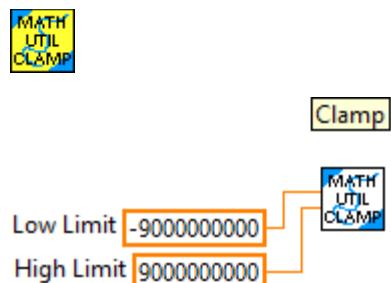
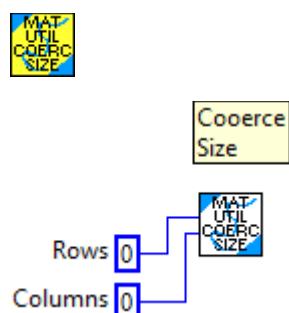
macro_LinearSystem_Sim_Execute

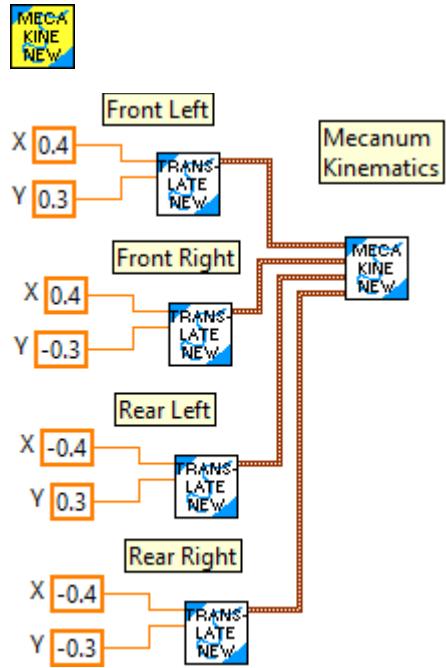




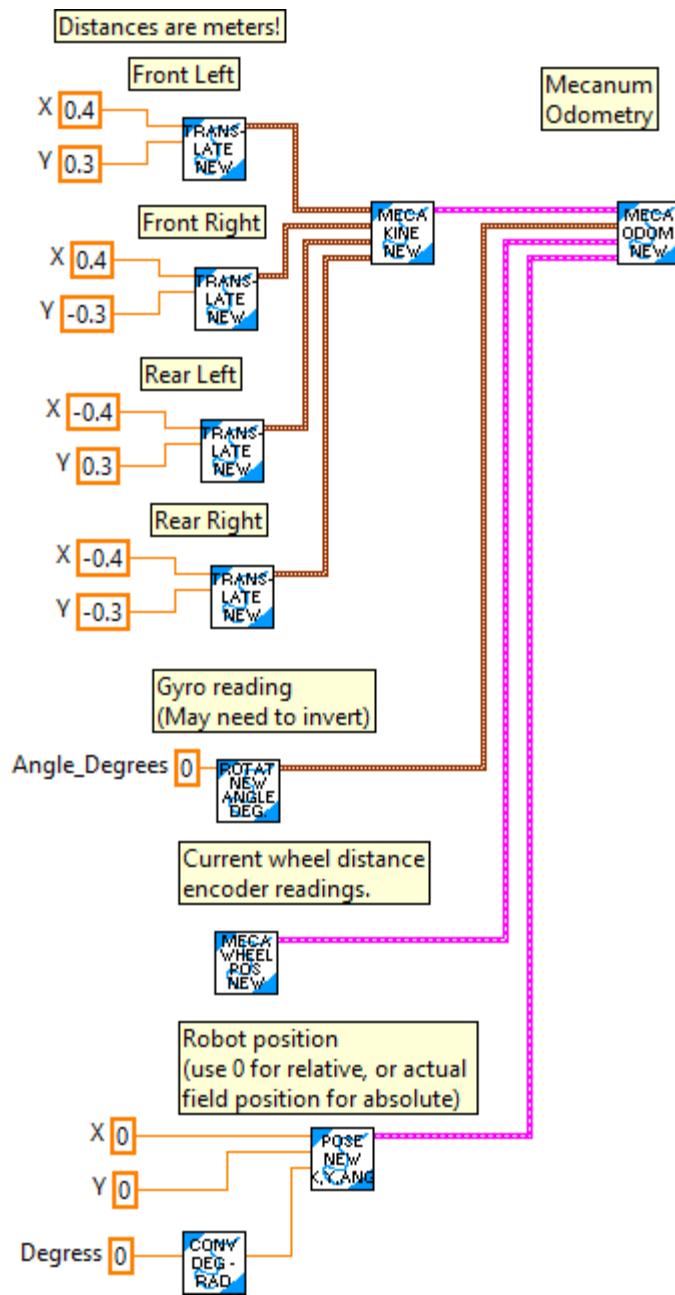
macro_MatBuilder_Create



macro_MatBuilder_Fill**macro_MathUtils_ClampInput****macro_MatrixHelper_CoerceSize**

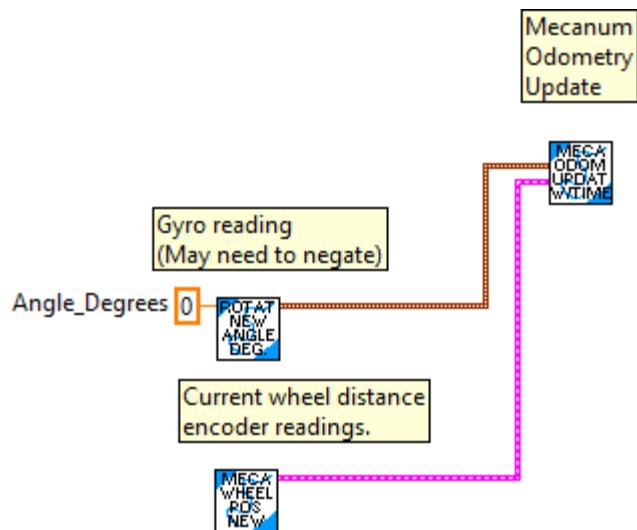
macro_MecaDriveKineNew

macro_MecaDriveOdomNew



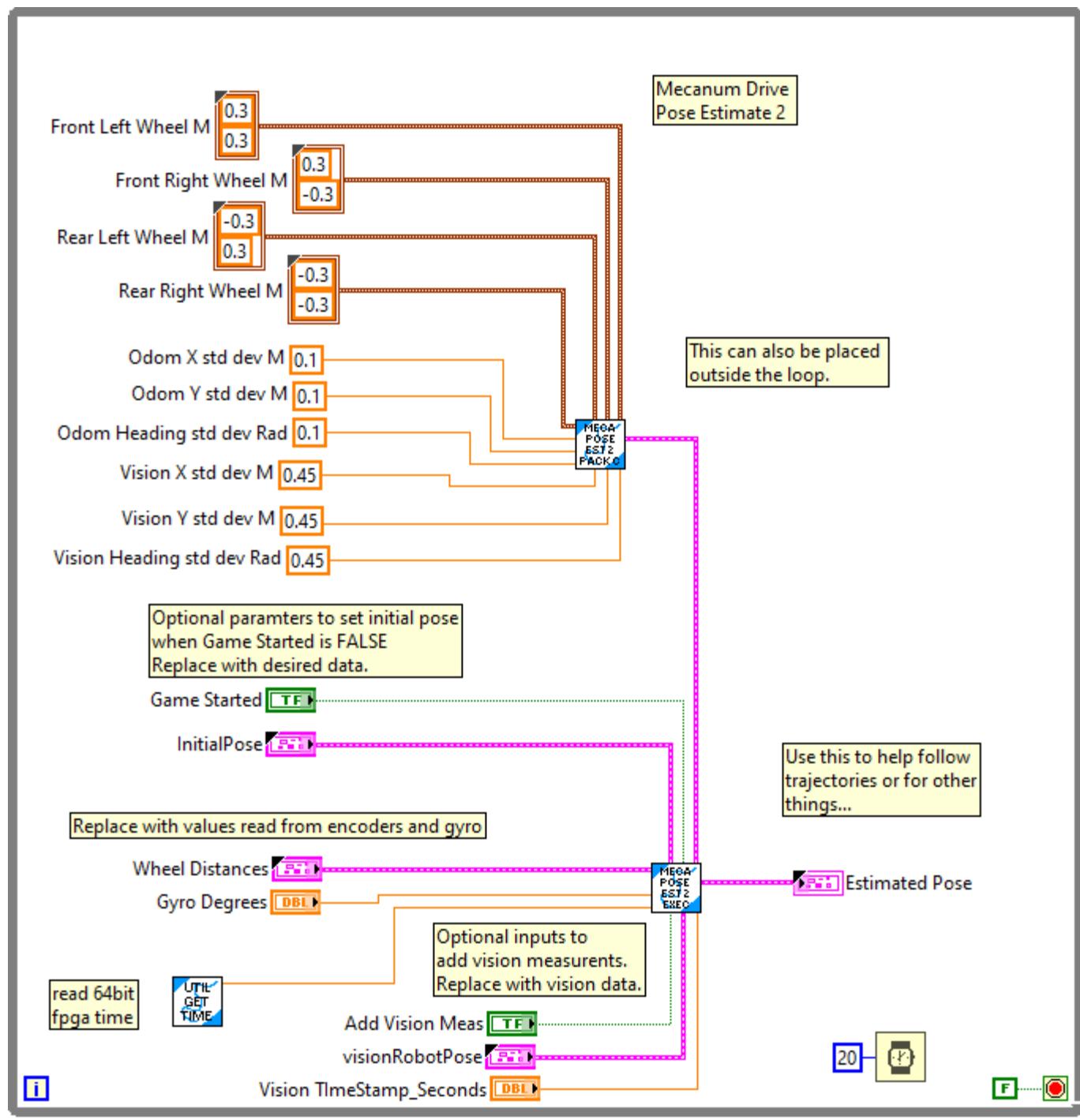
macro_MecaDriveOdomUpdate

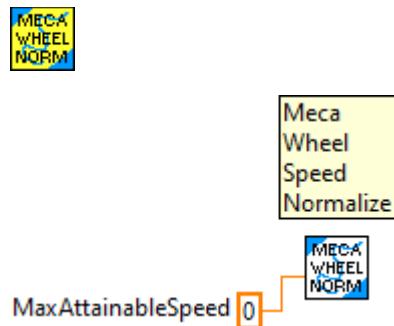
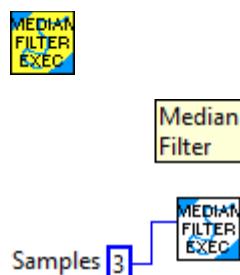
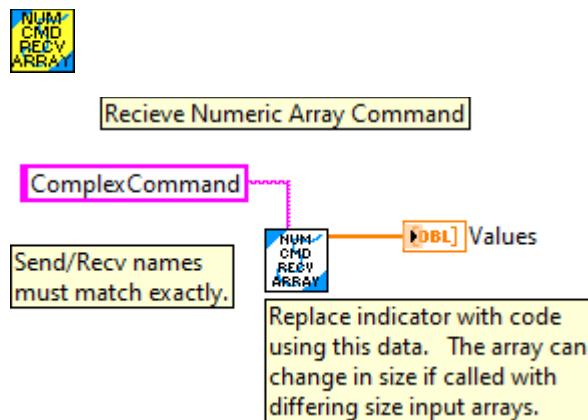




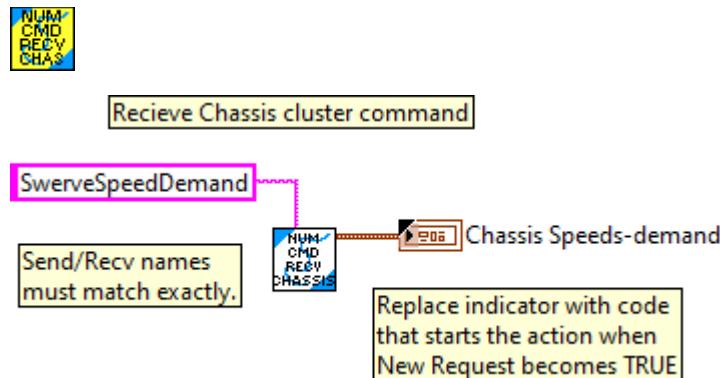
macro_MecaDrivePoseEst2_Execute



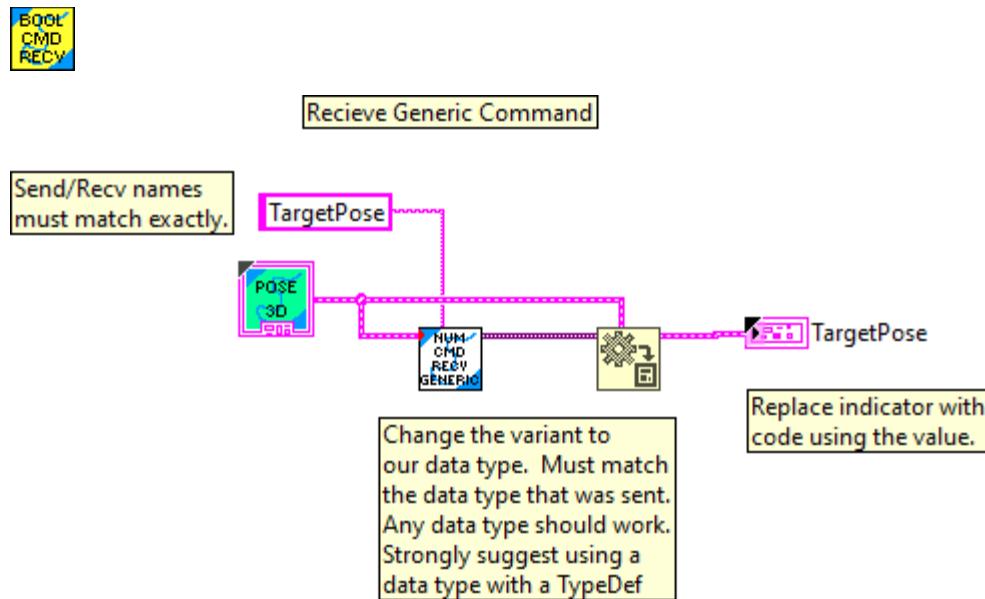


macro_MecaWheelSpeedNormalize**macro_MedianFilter_Execute****macro_NumCmd_Recv_Array**

macro_NumCmd_Recv_Chassis

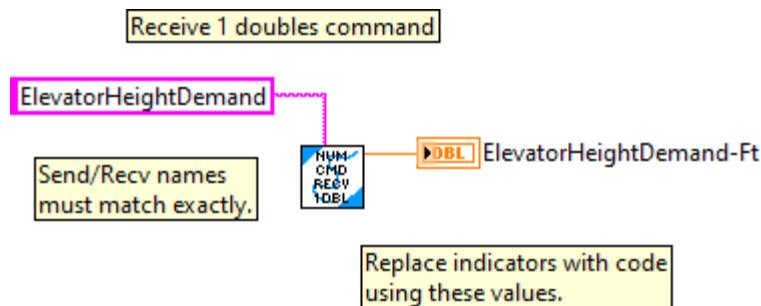
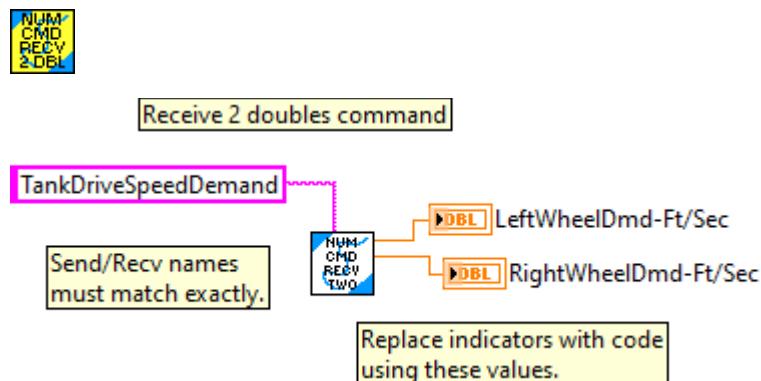
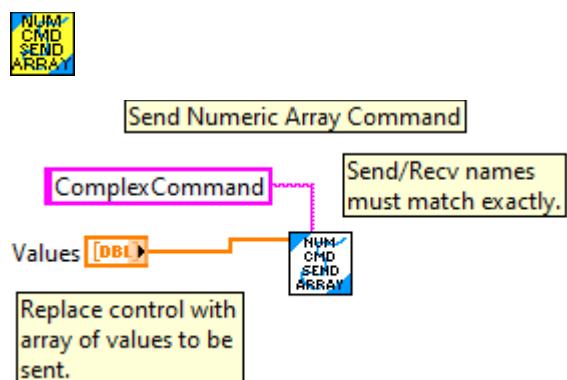


macro_NumCmd_Recv_Generic

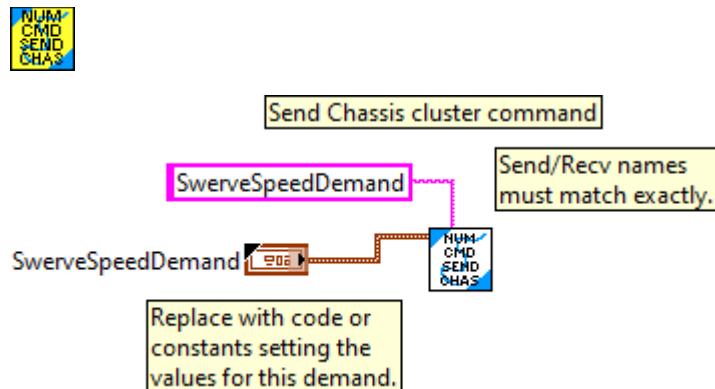


macro_NumCmd_Recv_OneDbl

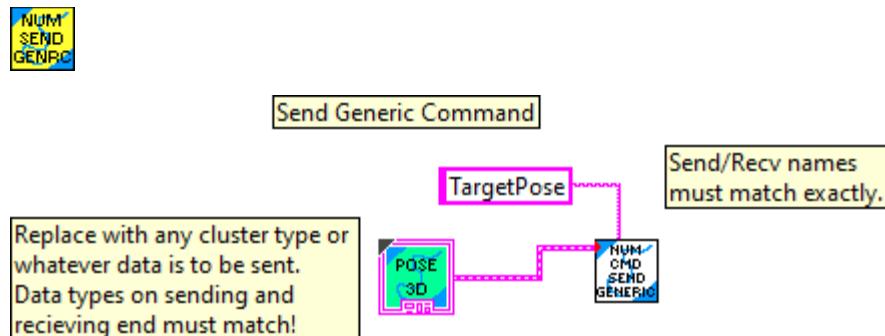


**macro_NumCmd_Recv_TwoDbl****macro_NumCmd_Send_Array**

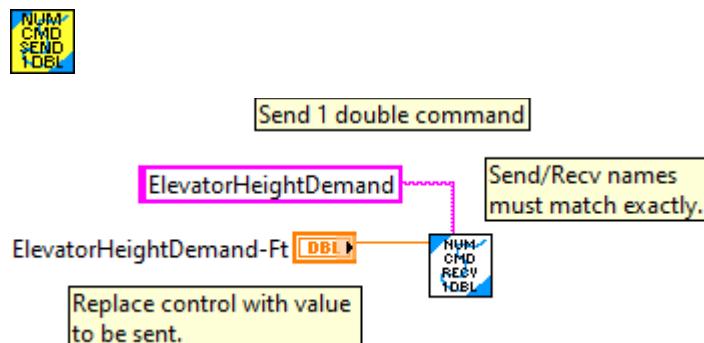
macro_NumCmd_Send_Chassis



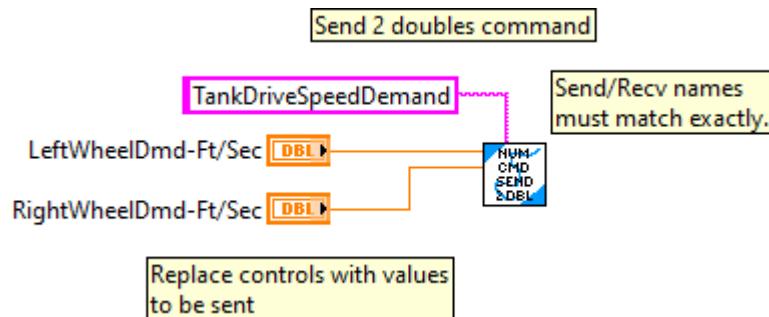
macro_NumCmd_Send_Generic



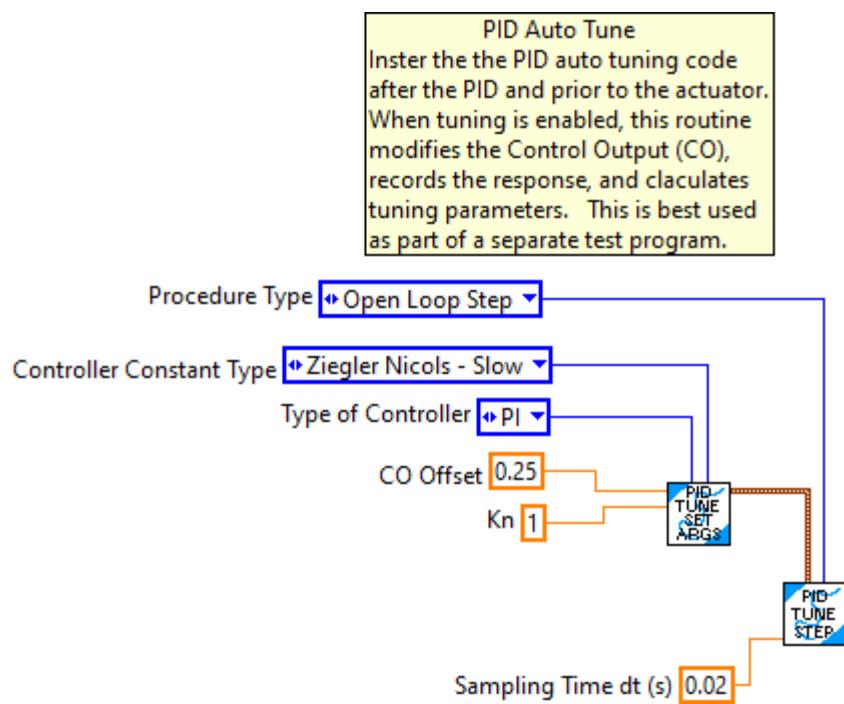
macro_NumCmd_Send_OneDbl



macro_NumCmd_Send_TwoDbl

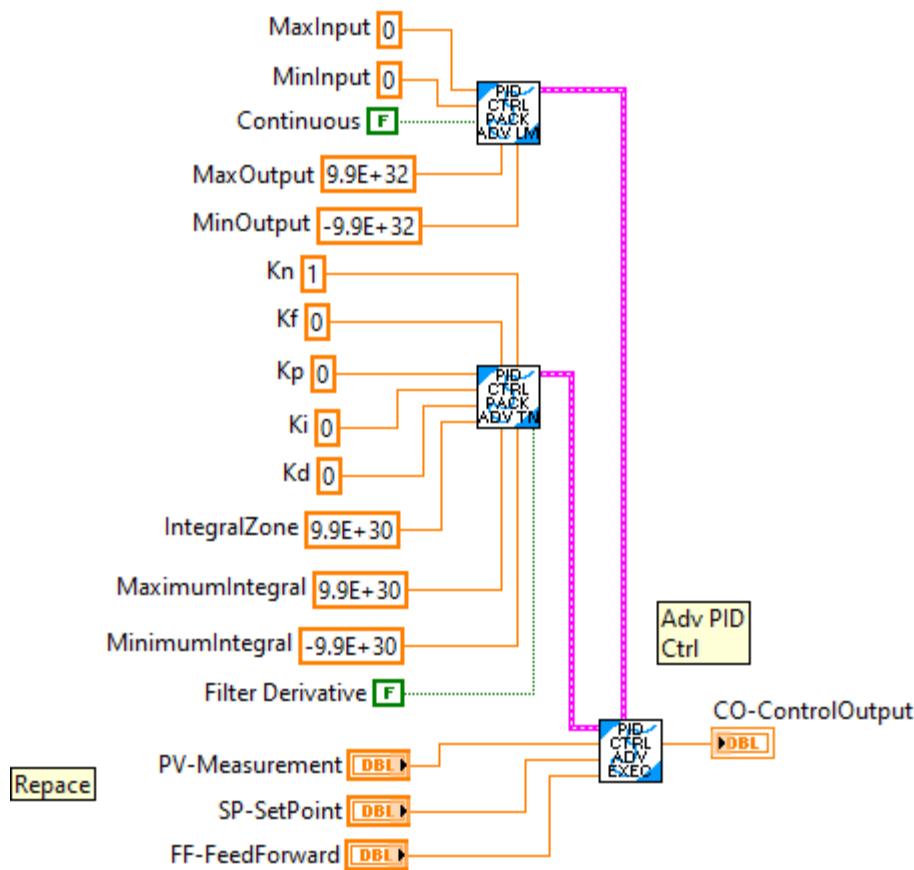


macro_PIDAutoTune



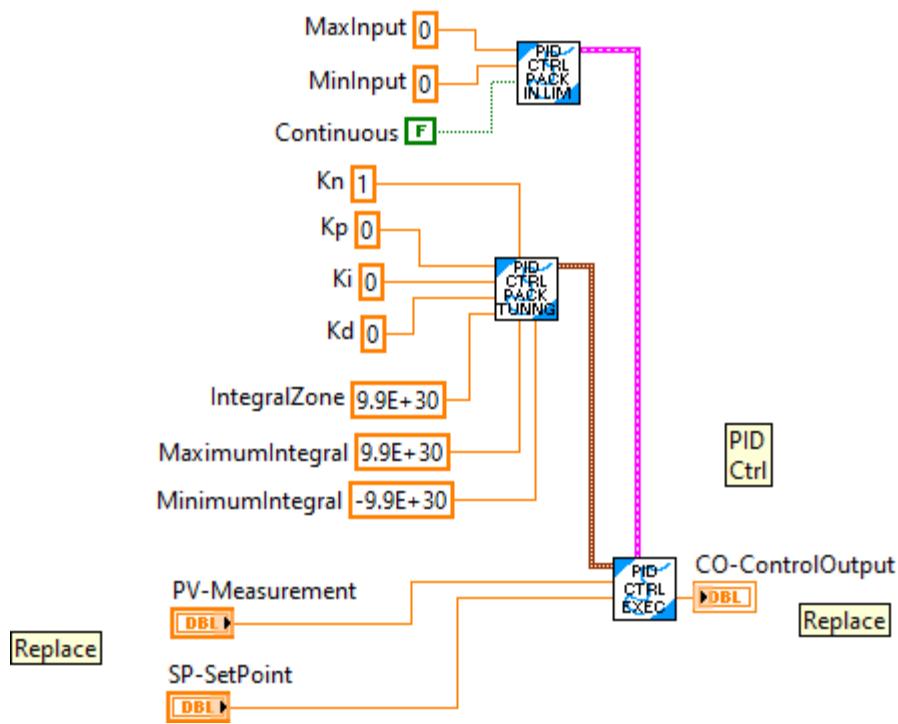
macro_PIDController_AdvExecute



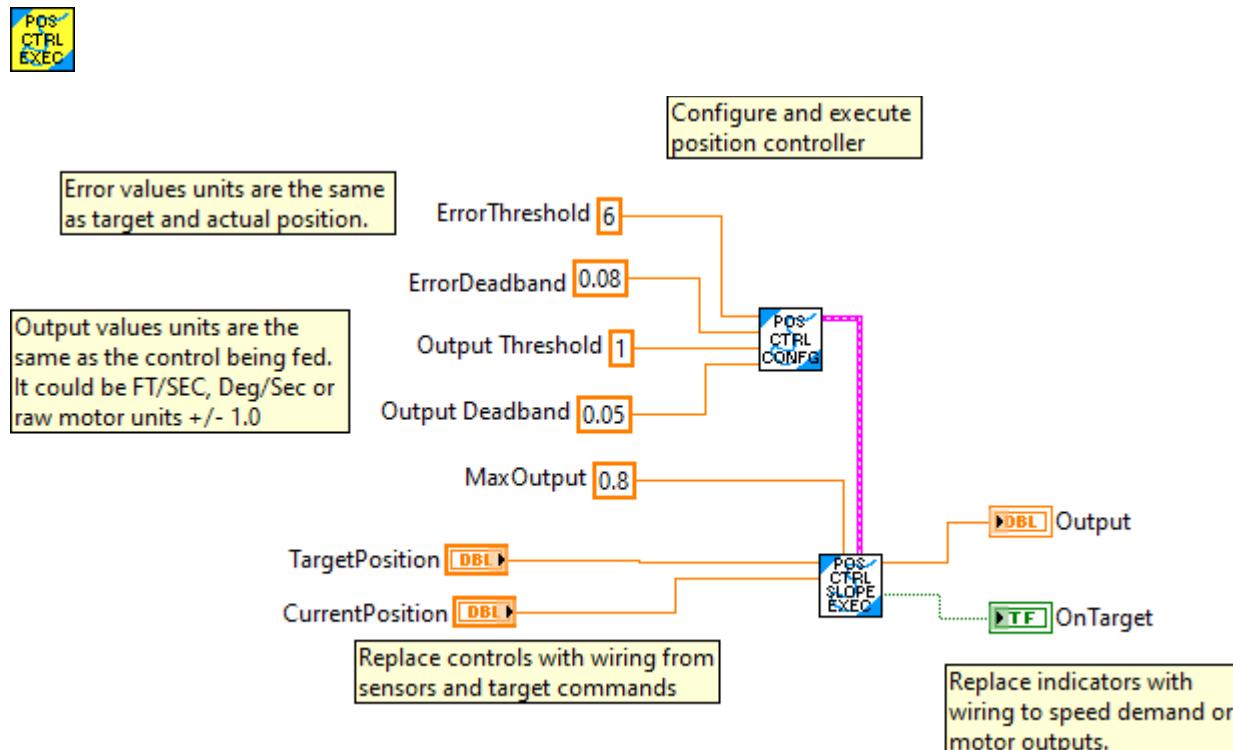


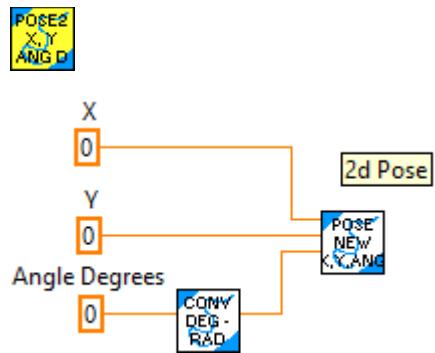
macro_PIDController_Execute

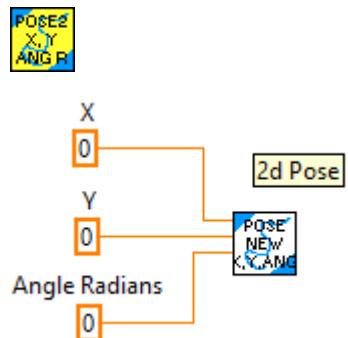




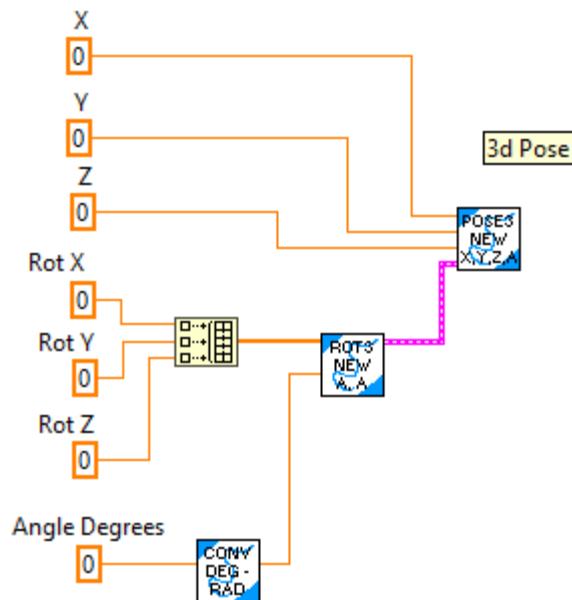
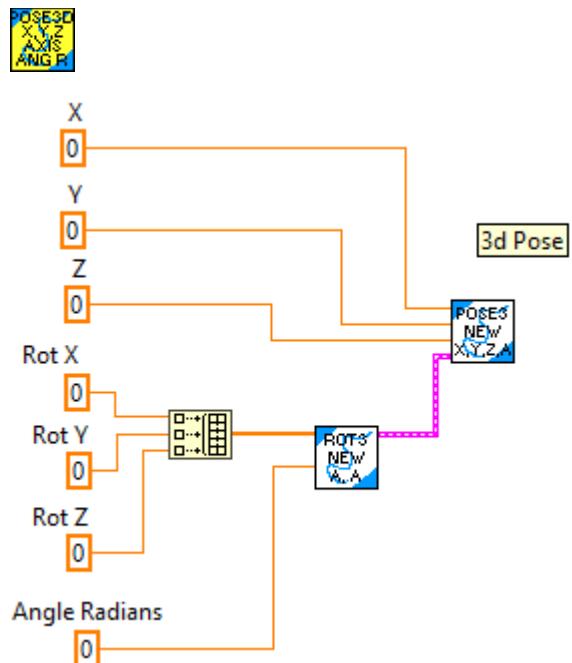
macro_PosCtrl_Execute

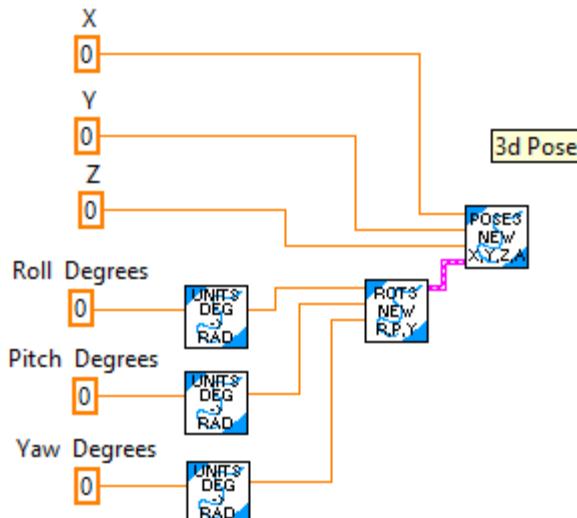


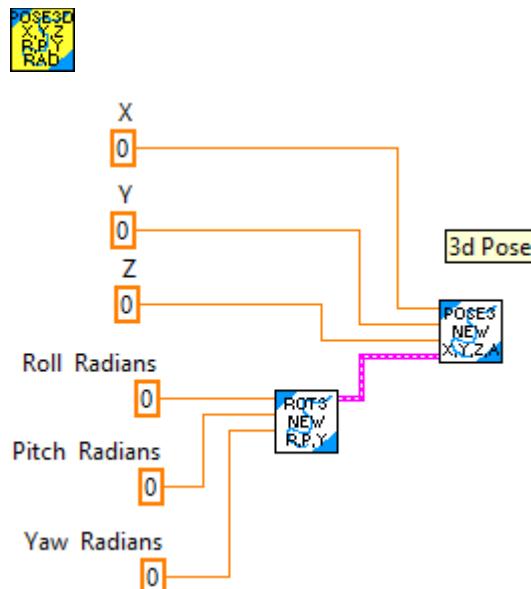
macro_Pose2d_XYAngleDeg

macro_Pose2d_XYAngleRad

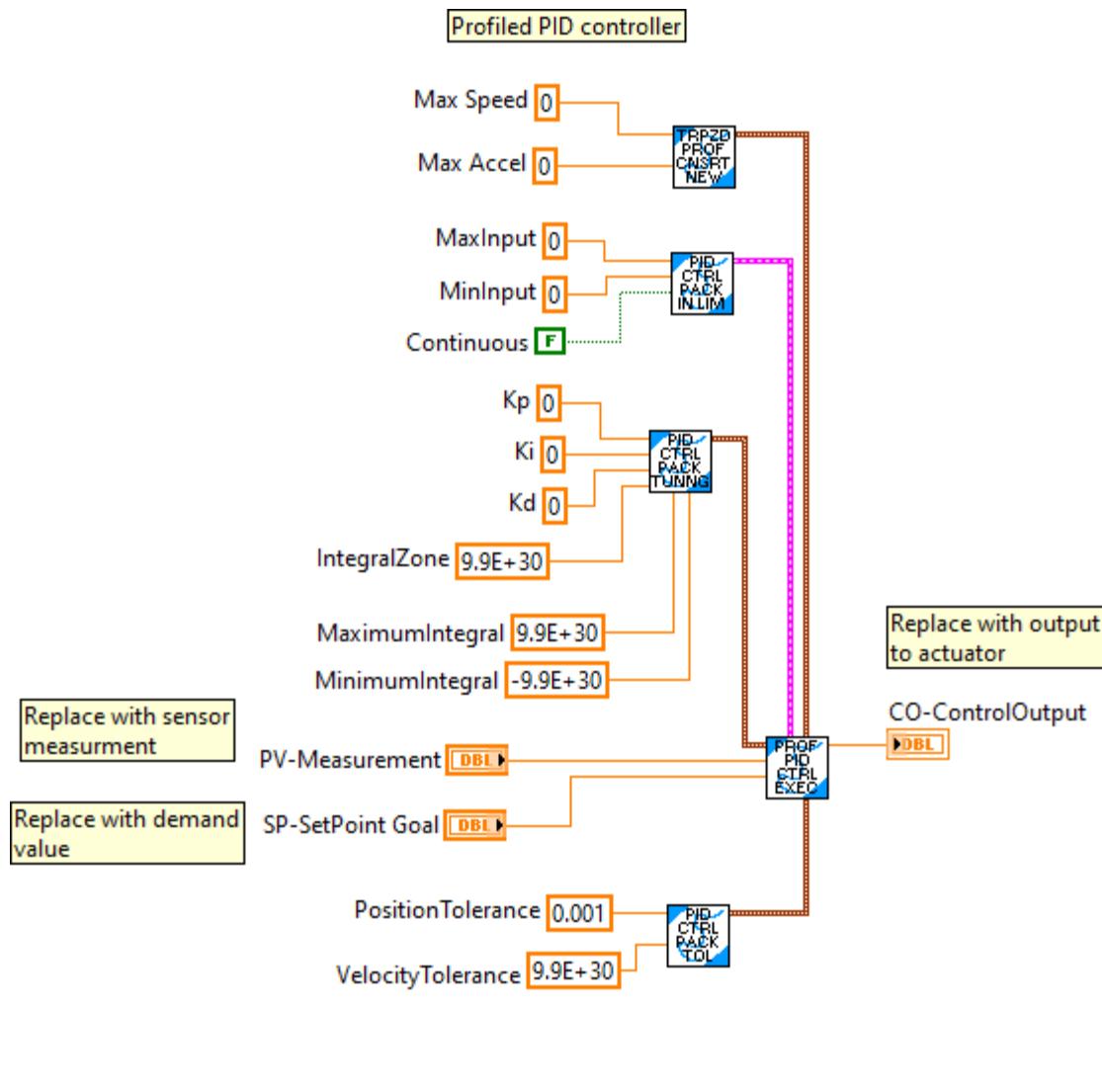
macro_Pose3d_XYZ_Axis_AngleDeg

**macro_Pose3d_XYZ_Axis_AngleRad****macro_Pose3d_XYZ_RollPitchYaw_Deg**

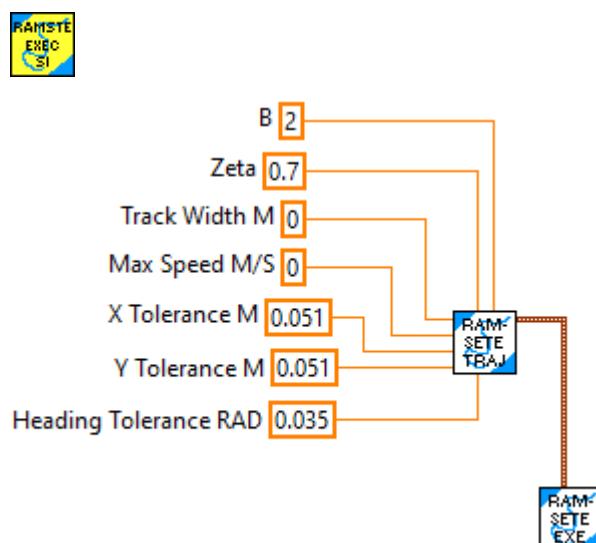


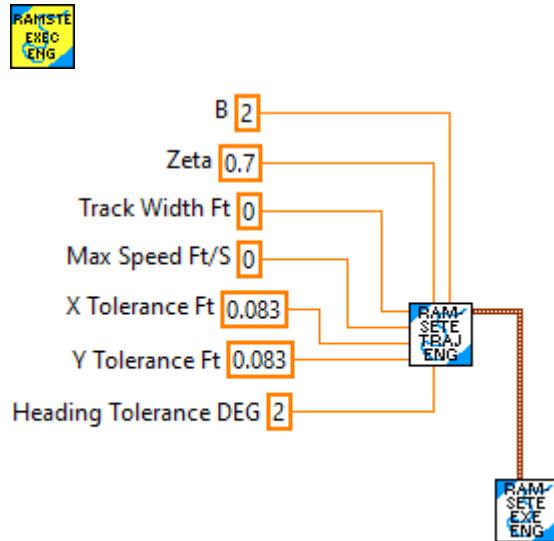
macro_Pose3d_XYZ_RollPitchYaw_Rad

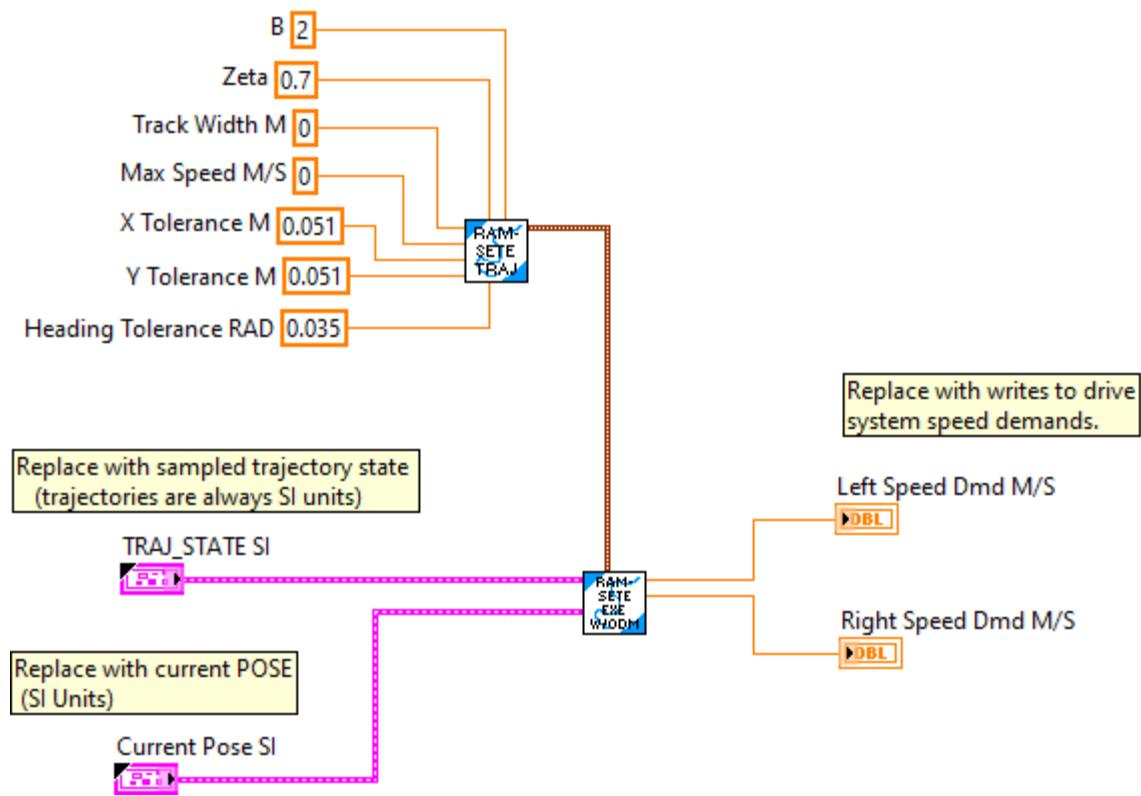
macro_ProfiledPIDController_Execute



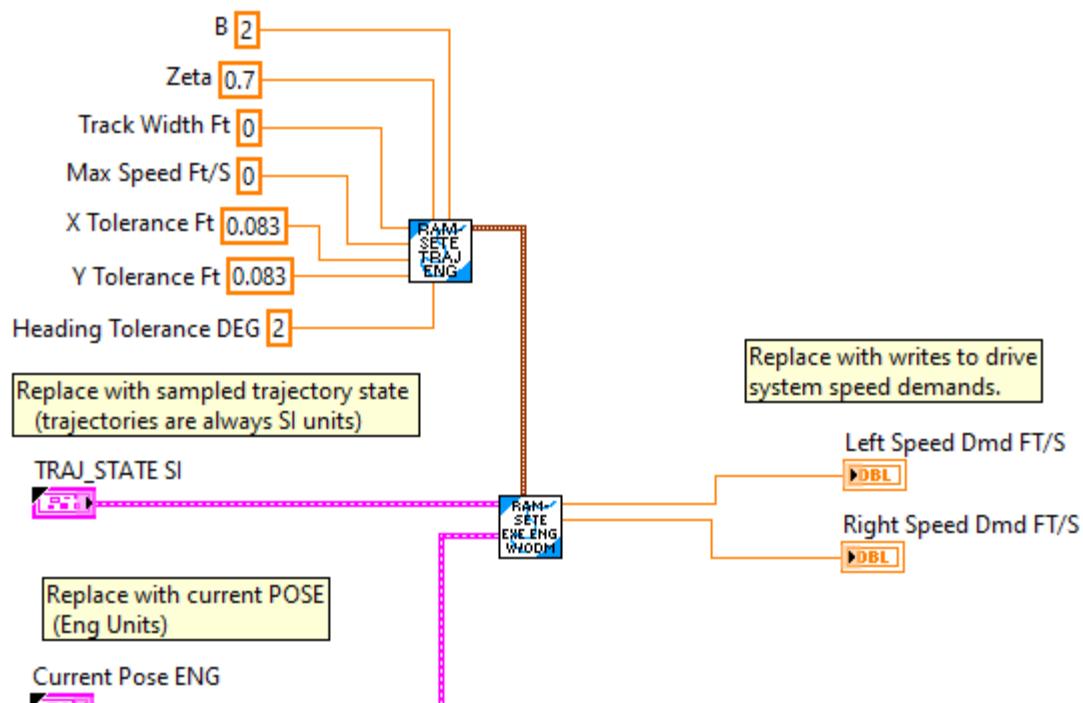
macro_Ramsete_Execute



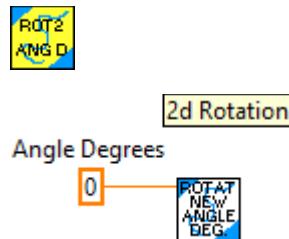
macro_Ramsete_Execute_ENG**macro_Ramsete_Execute_External_Odometry**



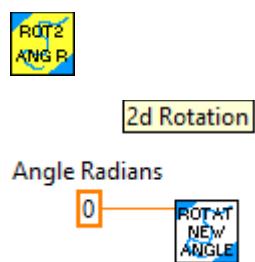
macro_Ramsete_Execute_External_Odometry_ENG

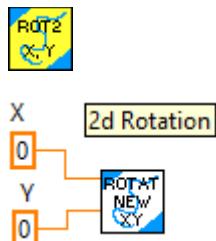



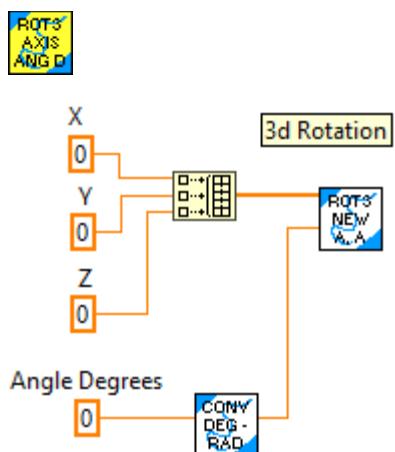
macro_Rotation2d_AngleDeg

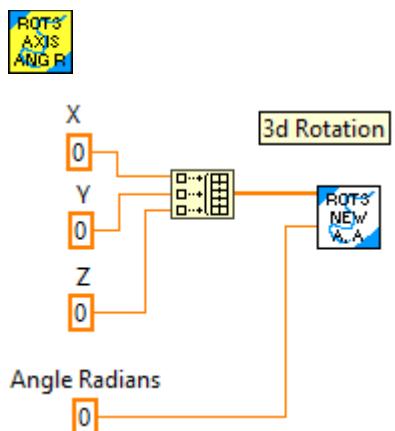


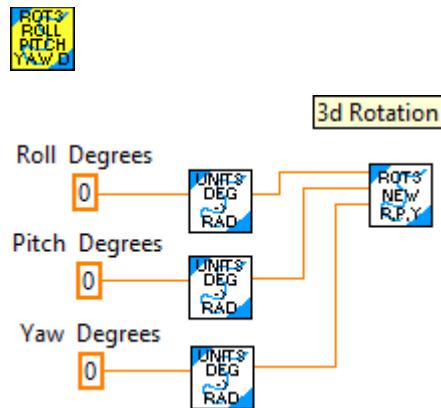
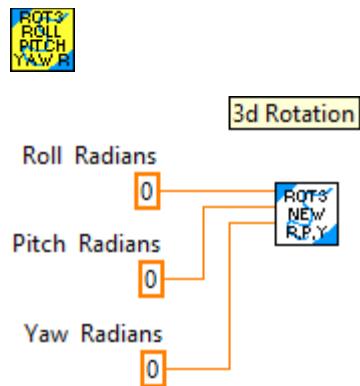
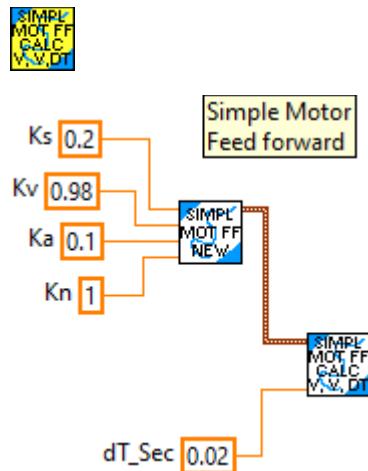
macro_Rotation2d_AngleRad

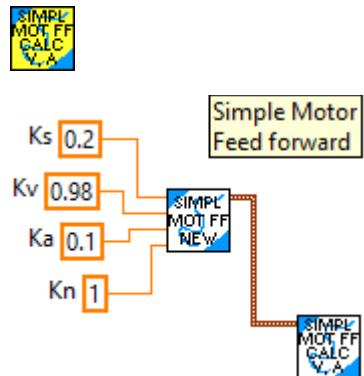


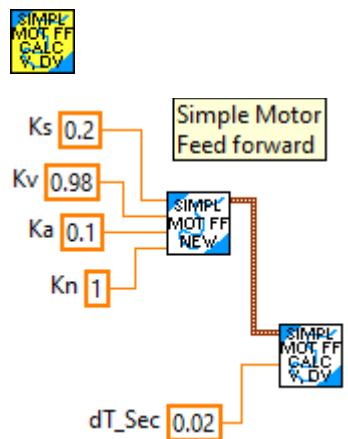
macro_Rotation2d_X_Y

macro_Rotation3d_Axis_AngleDeg

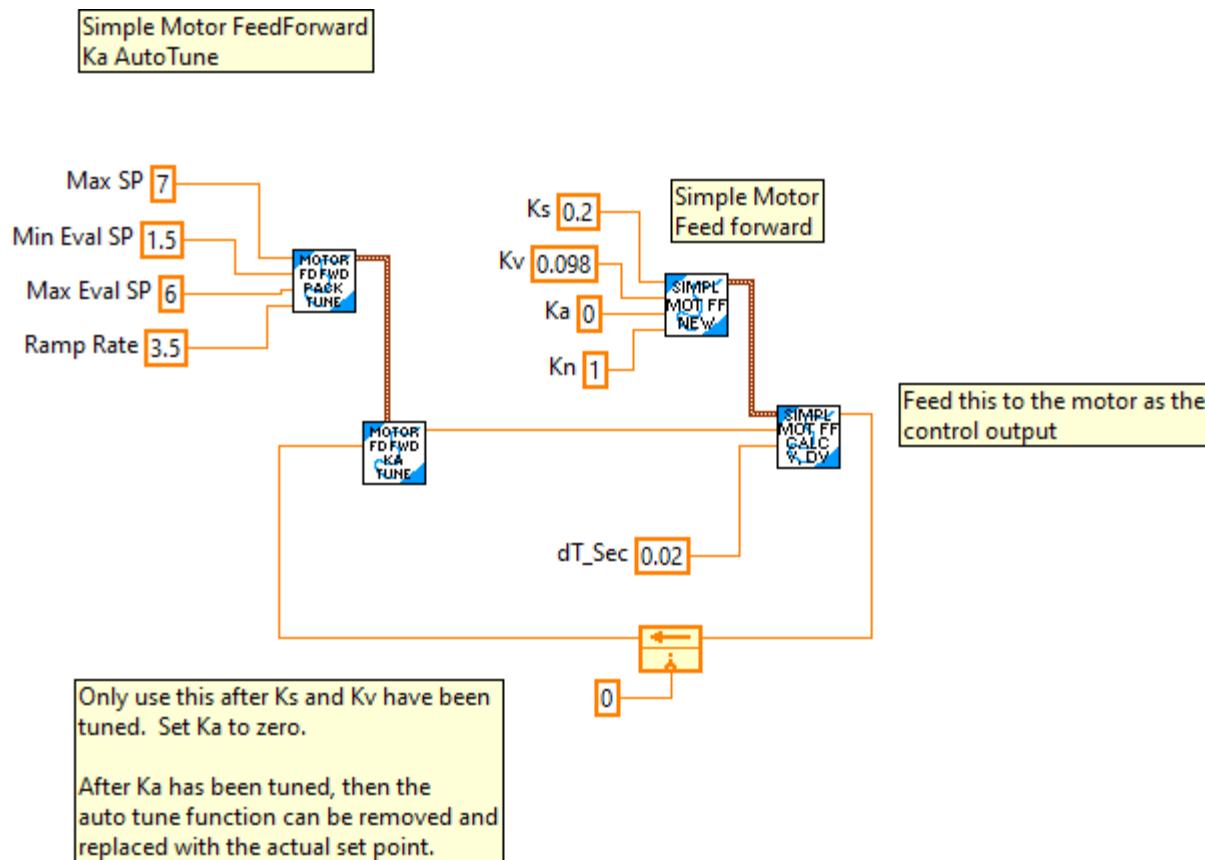
macro_Rotation3d_Axis_AngleRad

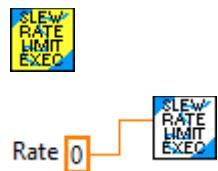
macro_Rotation3d_RollPitchYaw_Deg**macro_Rotation3d_RollPitchYaw_Rad****macro_SimpleMotorFF_Calc_NextV**

macro_SimpleMotorFF_Calculate

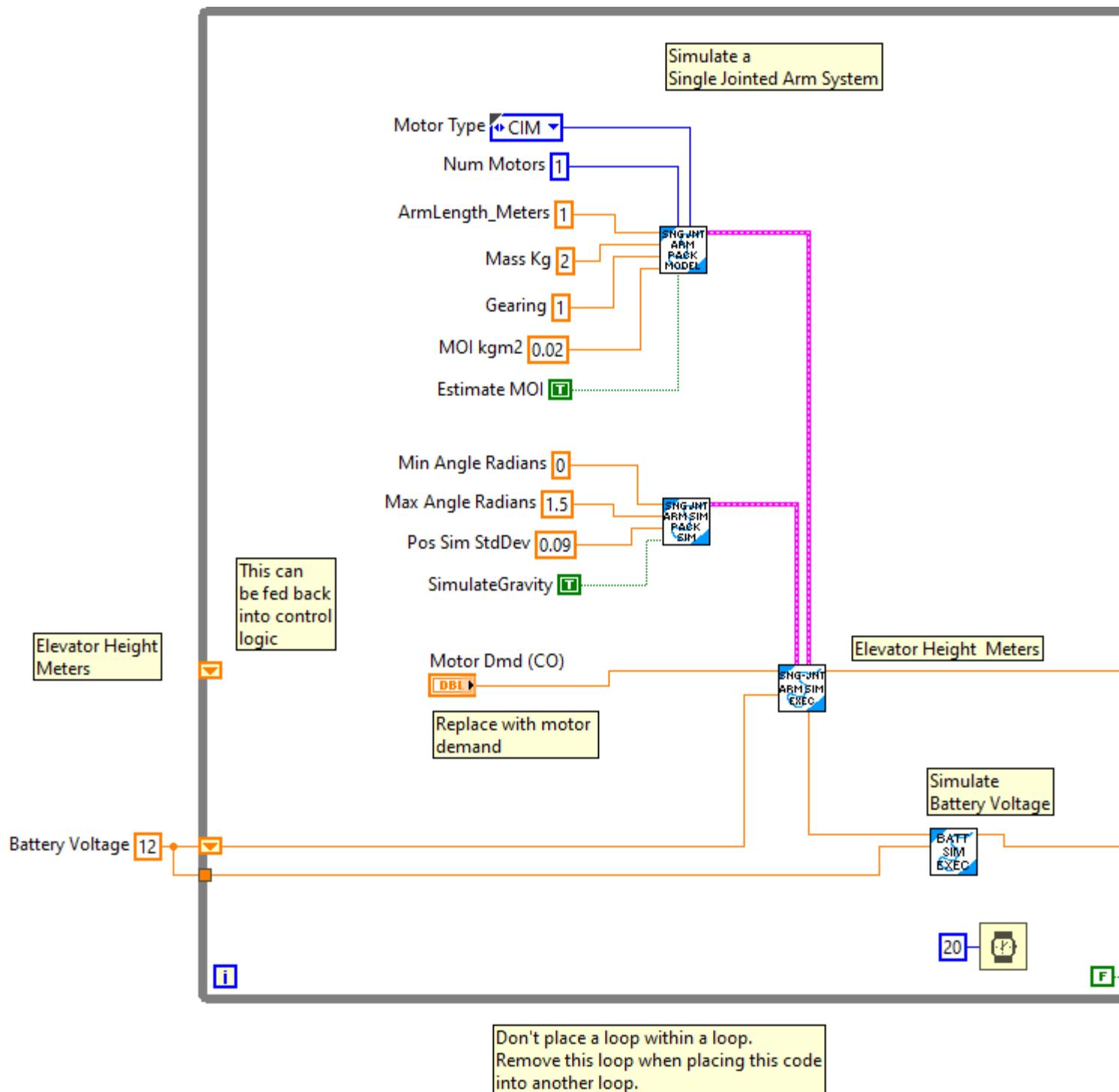
macro_SimpleMotorFF_Calculate_CalcAccel

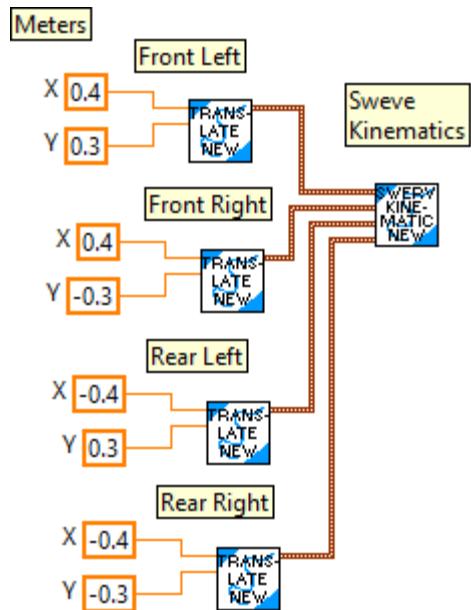
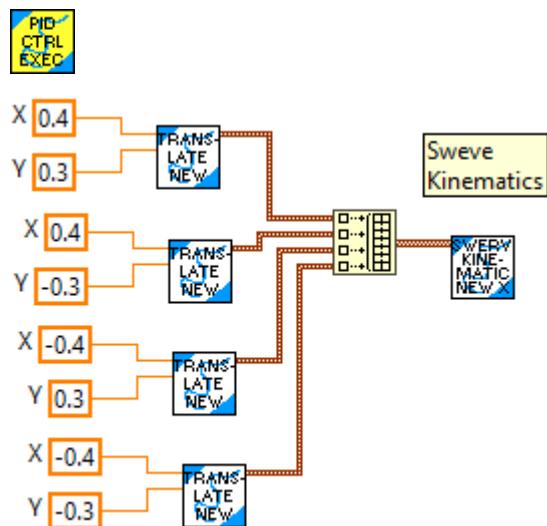
macro_SimpleMotorFF_Ka_AutoTune

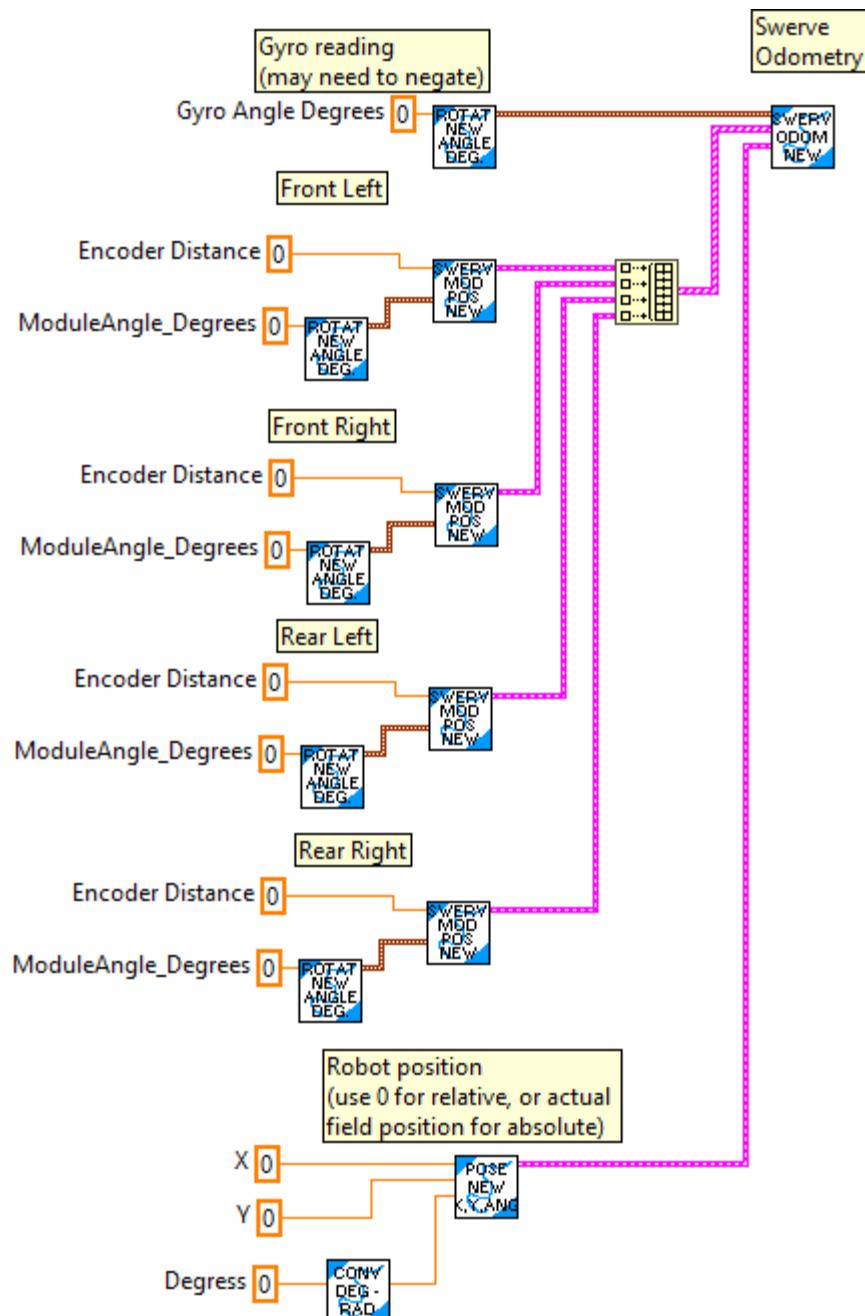


macro_SlewRateLimter_Execute

macro_SngJntArm_Sim_Execute

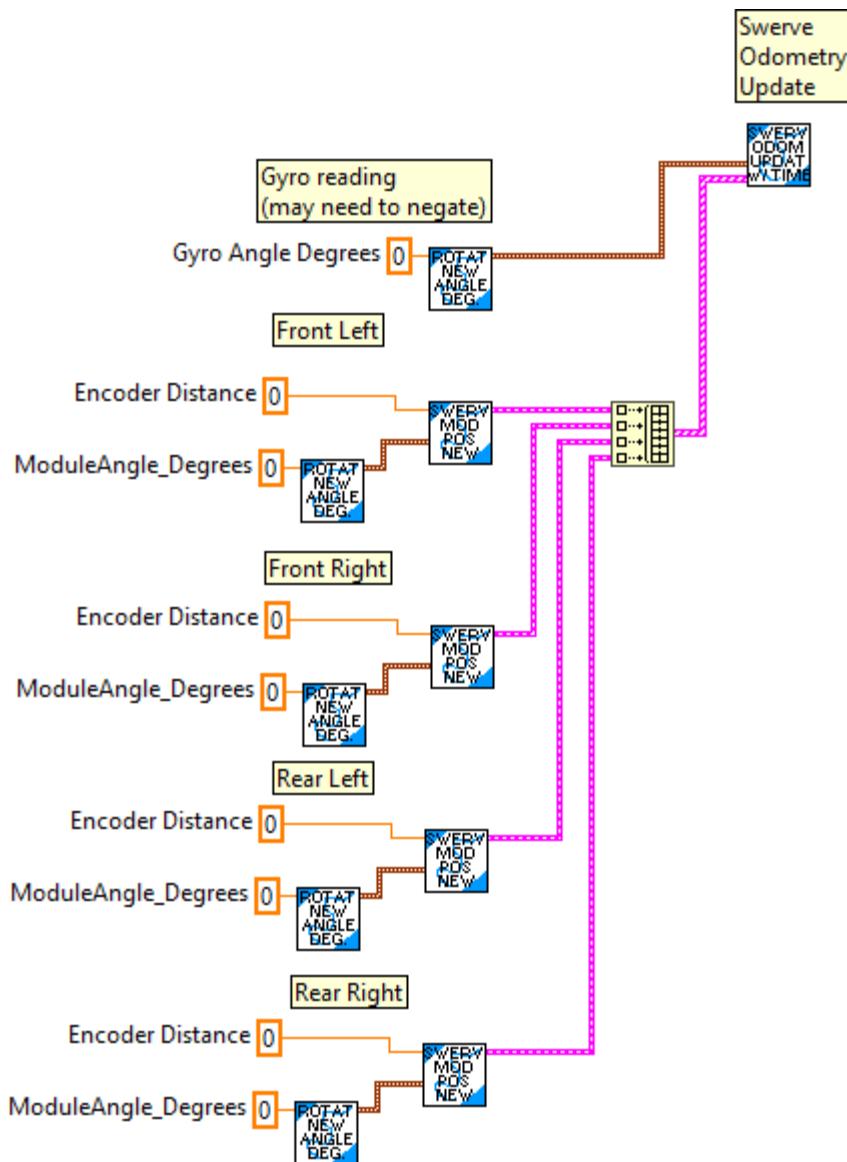
**macro_SwerveDriveKineNew4**

**macro_SwerveDriveKinematicsX****macro_SwerveDriveOdomNew4**



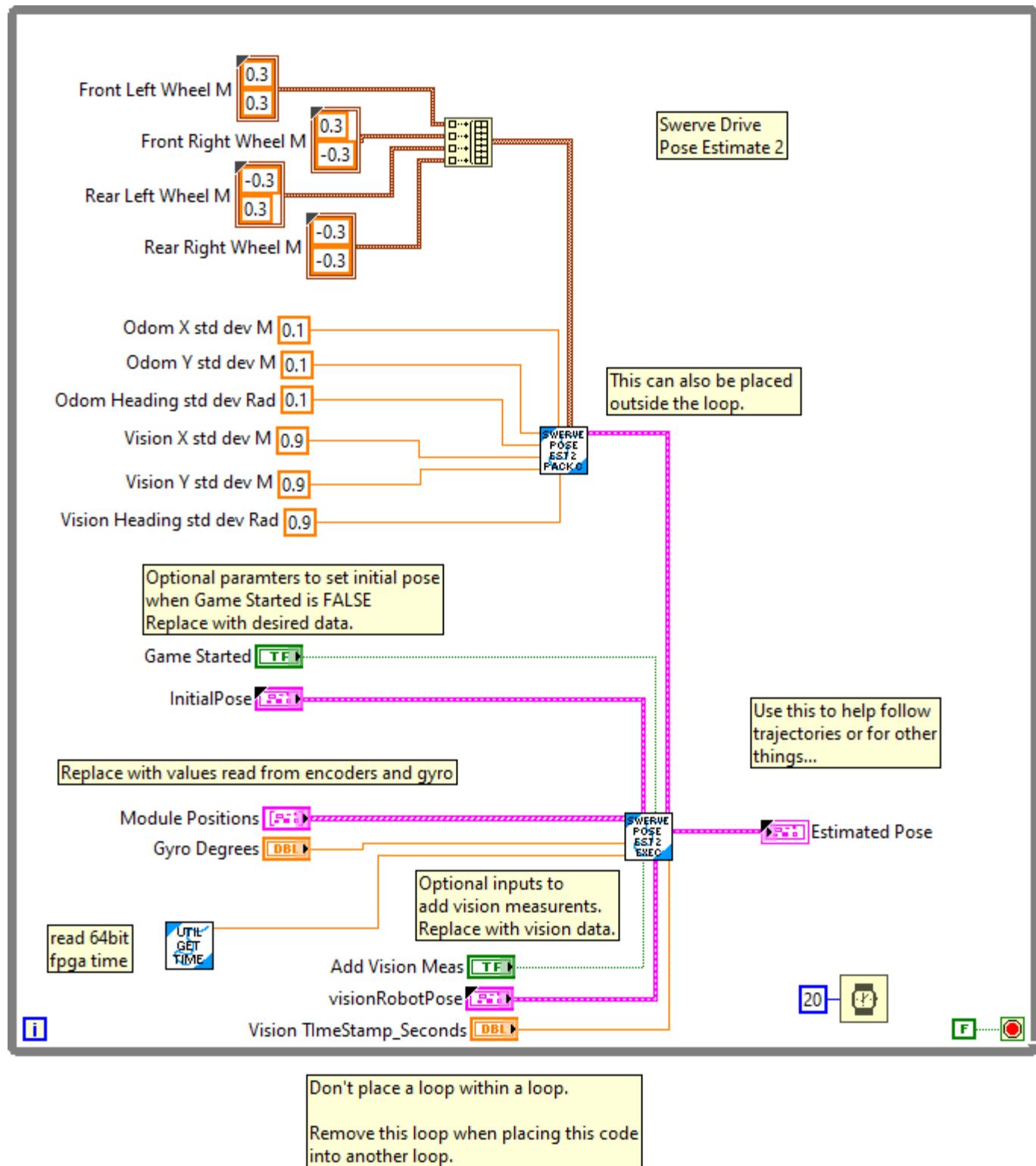
macro_SwerveDriveOdomUpdate4

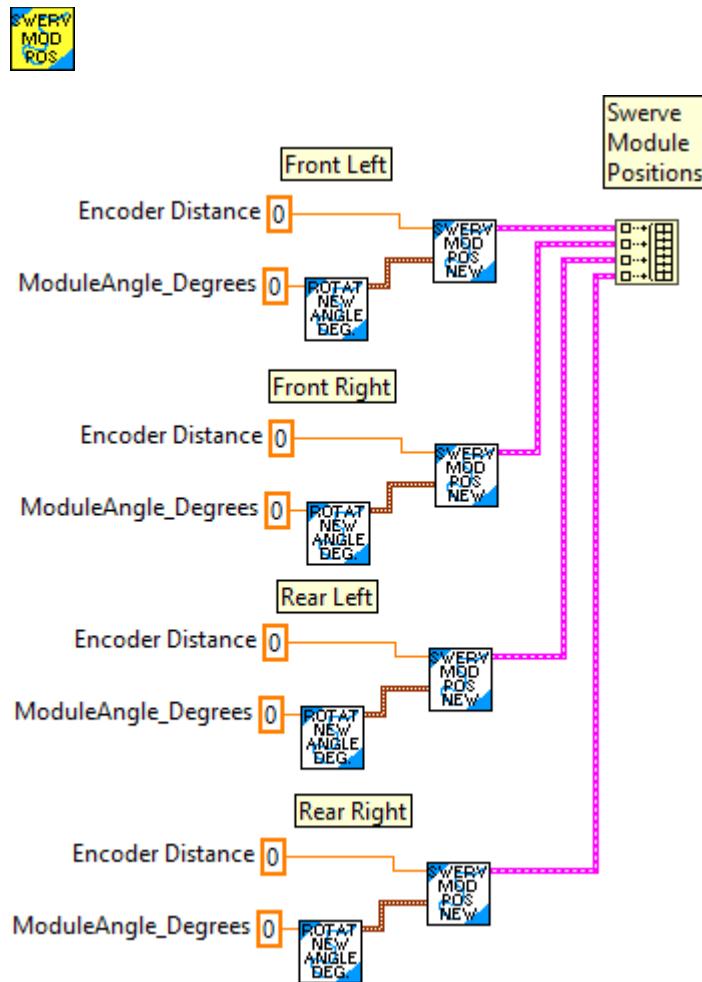


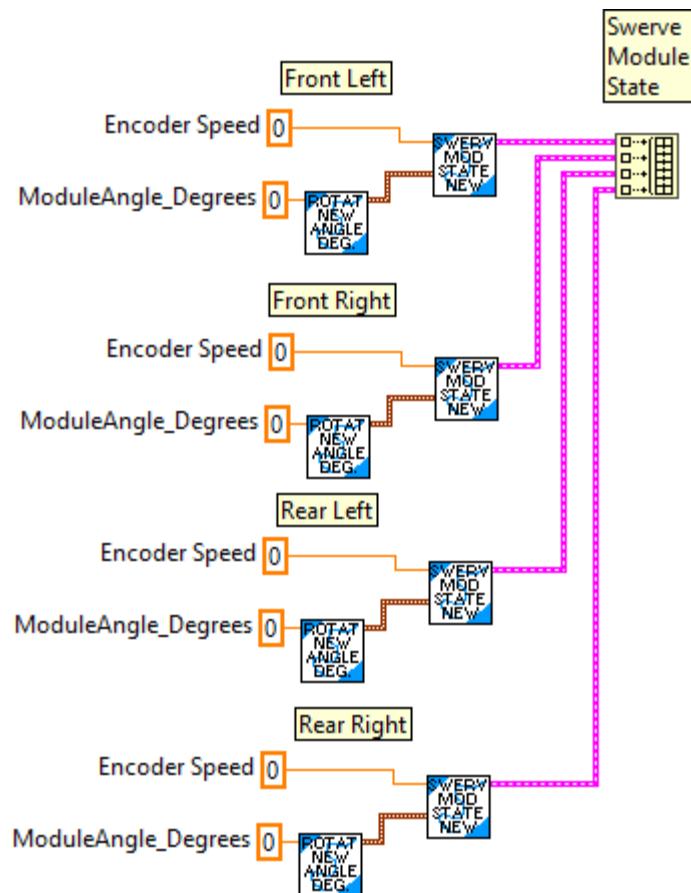


macro_SwerveDrivePoseEst2_Execute

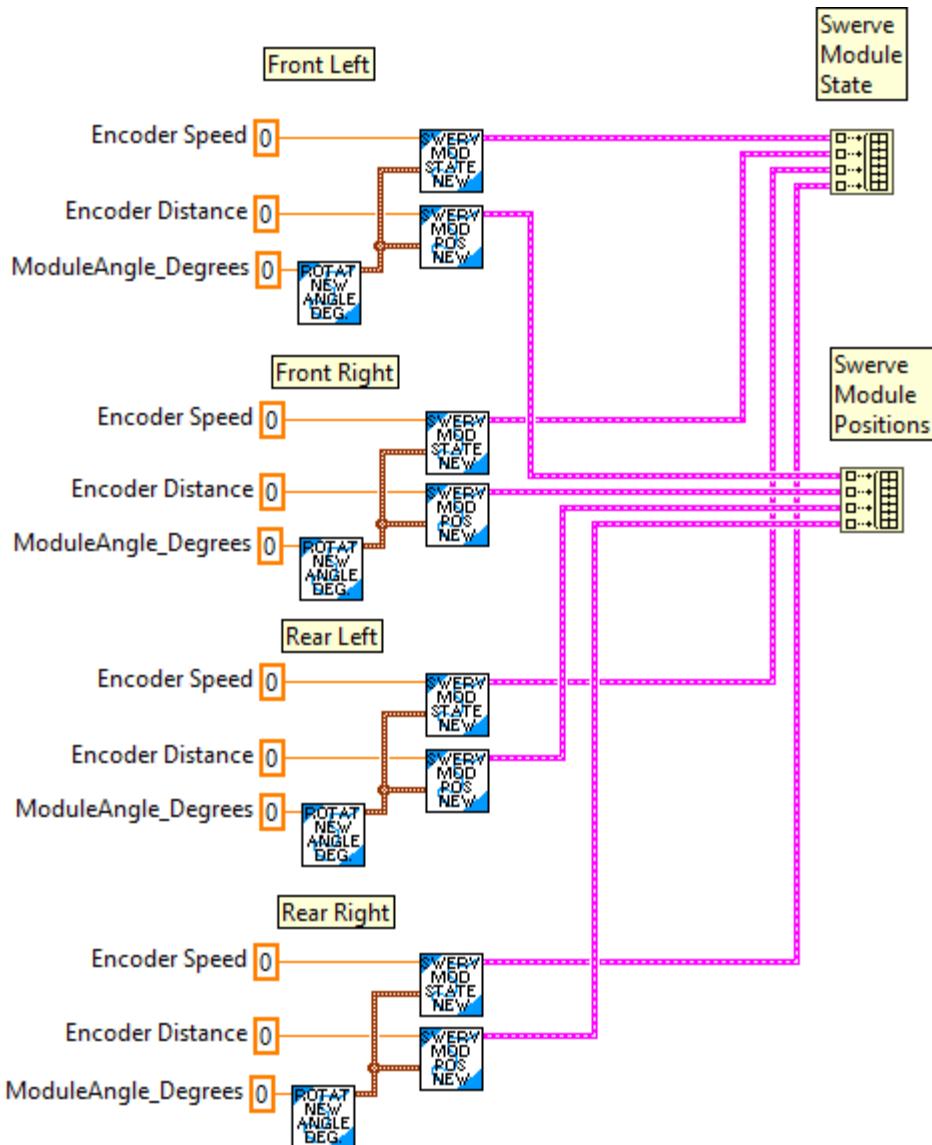




macro_SwerveModPosNew4**macro_SwerveModStateNew4**

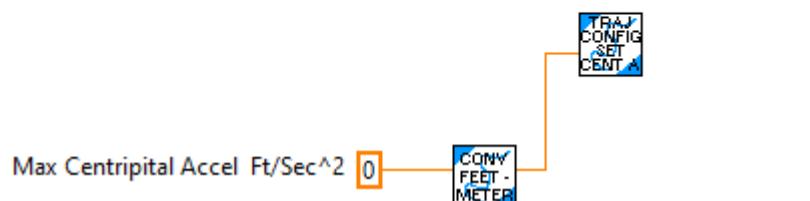


macro_SwerveModStatePosNew4

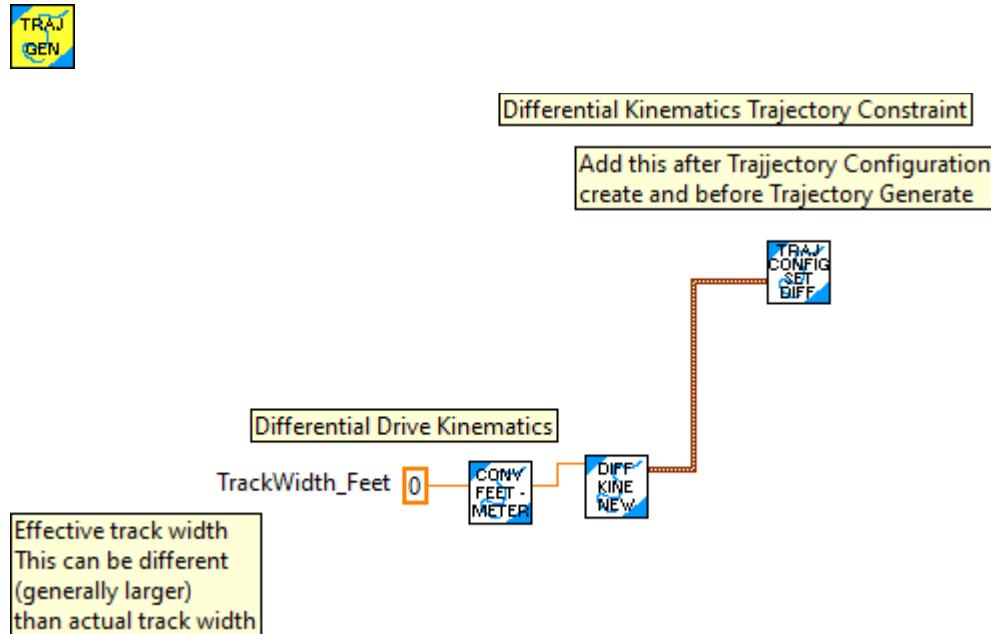
**macro_TrajectoryConstraint_CentripAccel**

Centripital Acceleration Trajectory Constraint

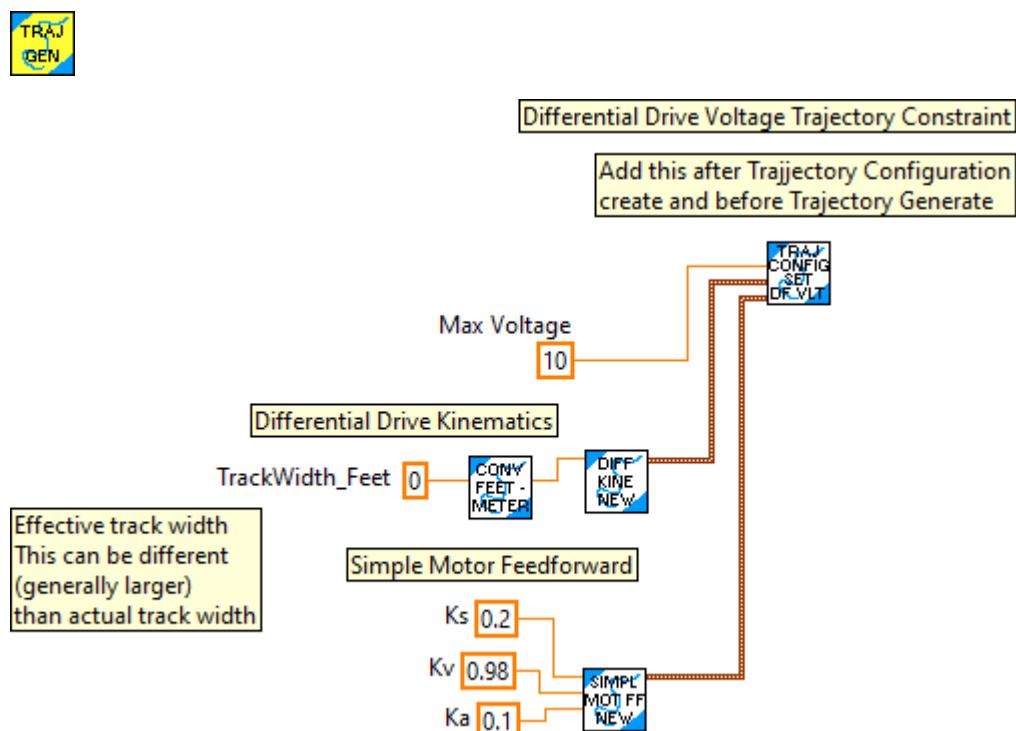
Add this after Trajectory Configuration
create and before Trajectory Generate



macro_TrajectoryConstraint_DiffKine



macro_TrajectoryConstraint_DiffVolt

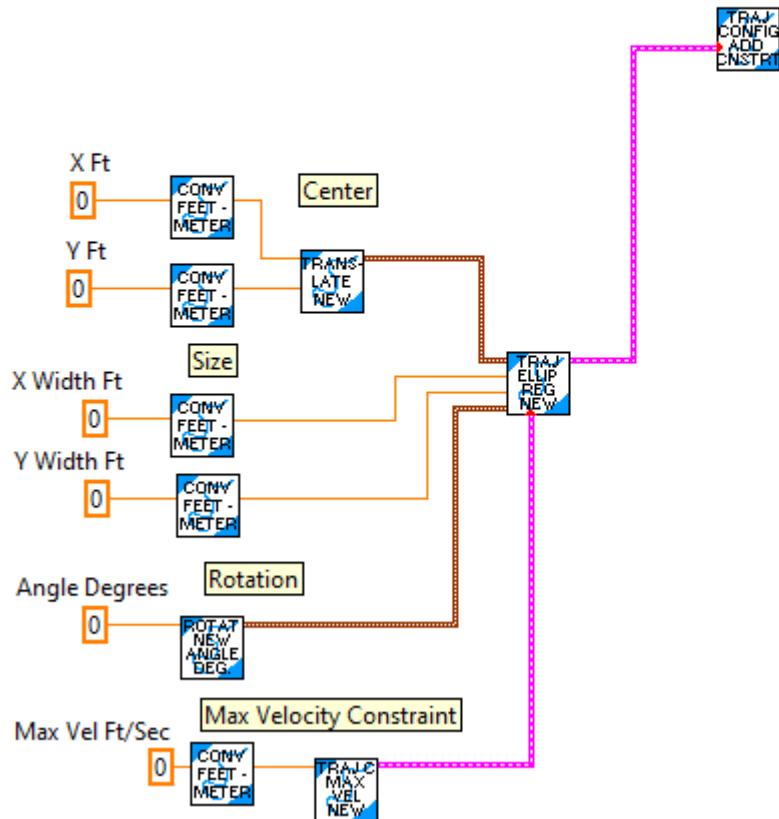


macro_TrajectoryConstraint_EllipRegion



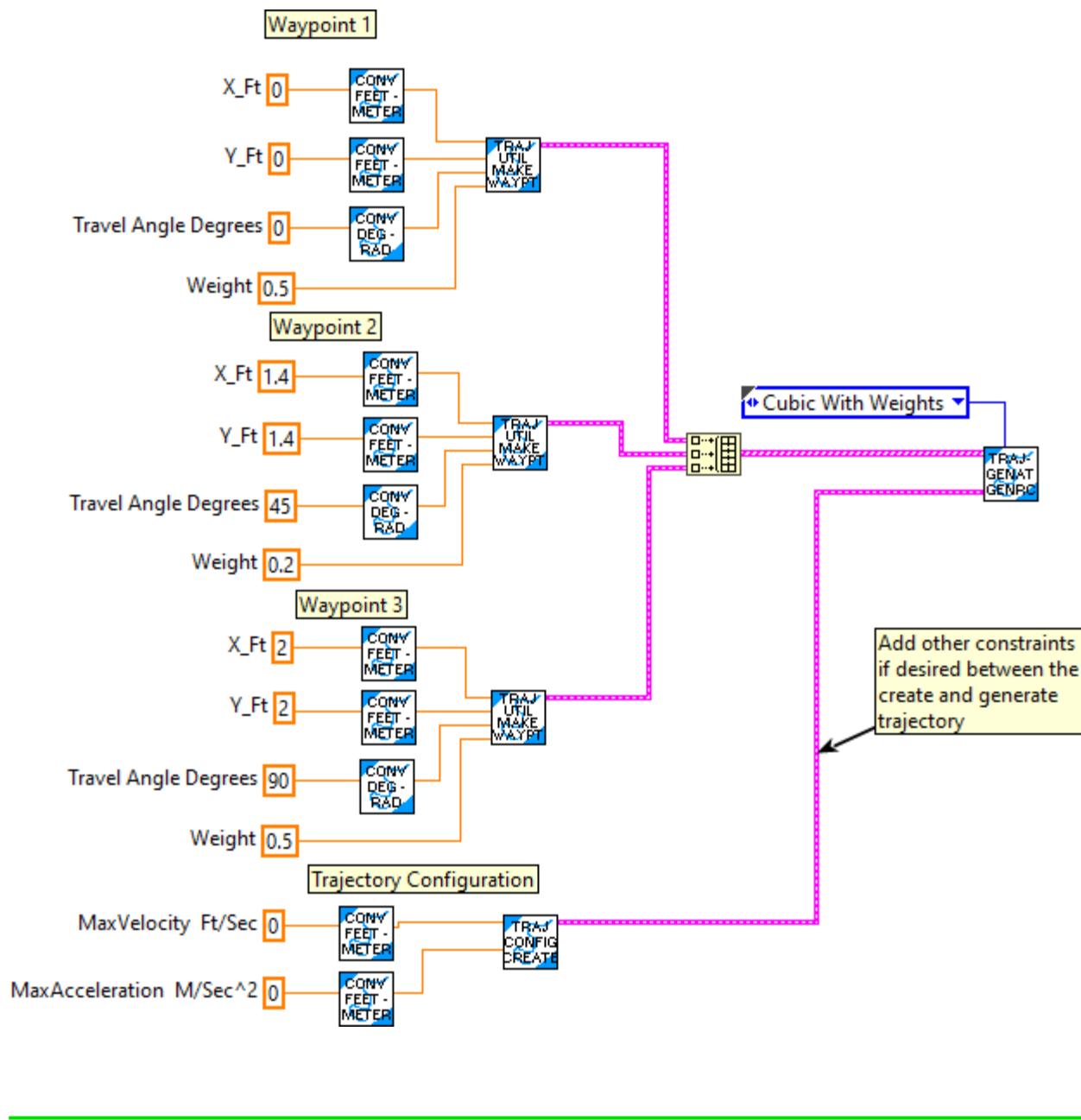
Elliptical Region Constraint -- Using Max Velocity Constraint

Add this after Trajectory Configuration
create and before Trajectory Generate



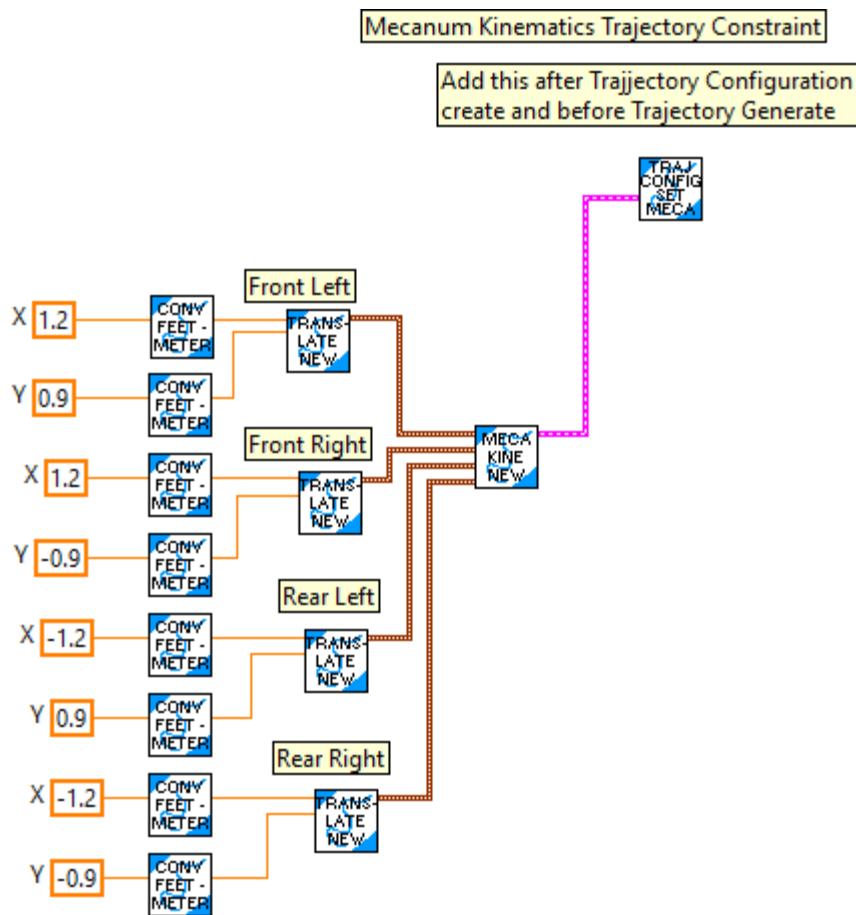
macro_TrajectoryConstraint_MaxVel





macro_TrajectoryConstraint_MecaKine



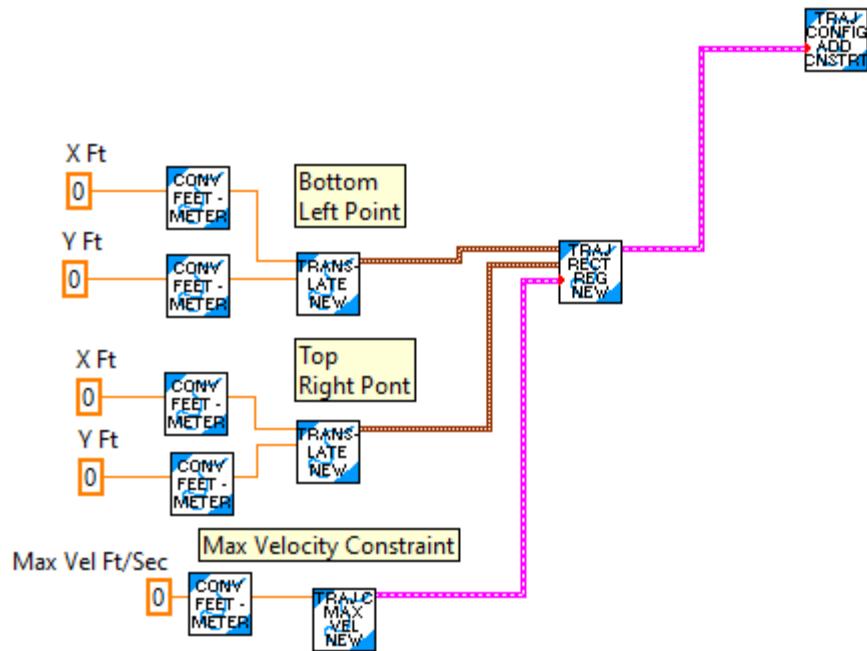


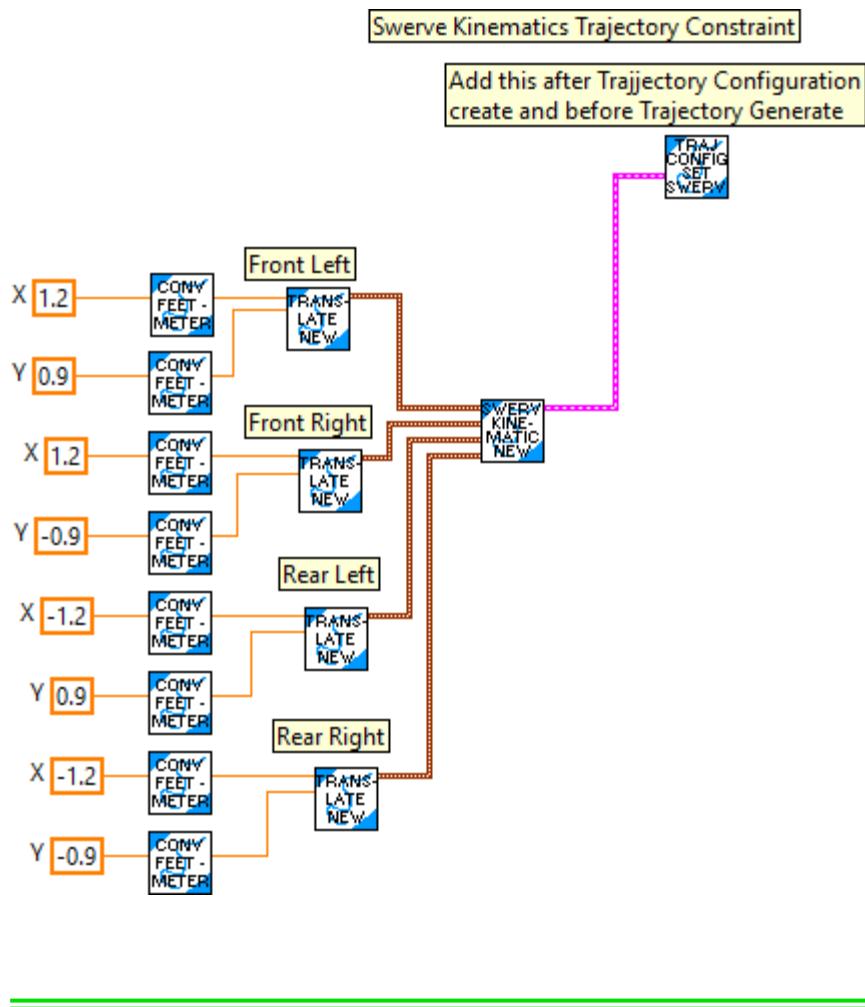
macro_TrajectoryConstraint_RectRegion



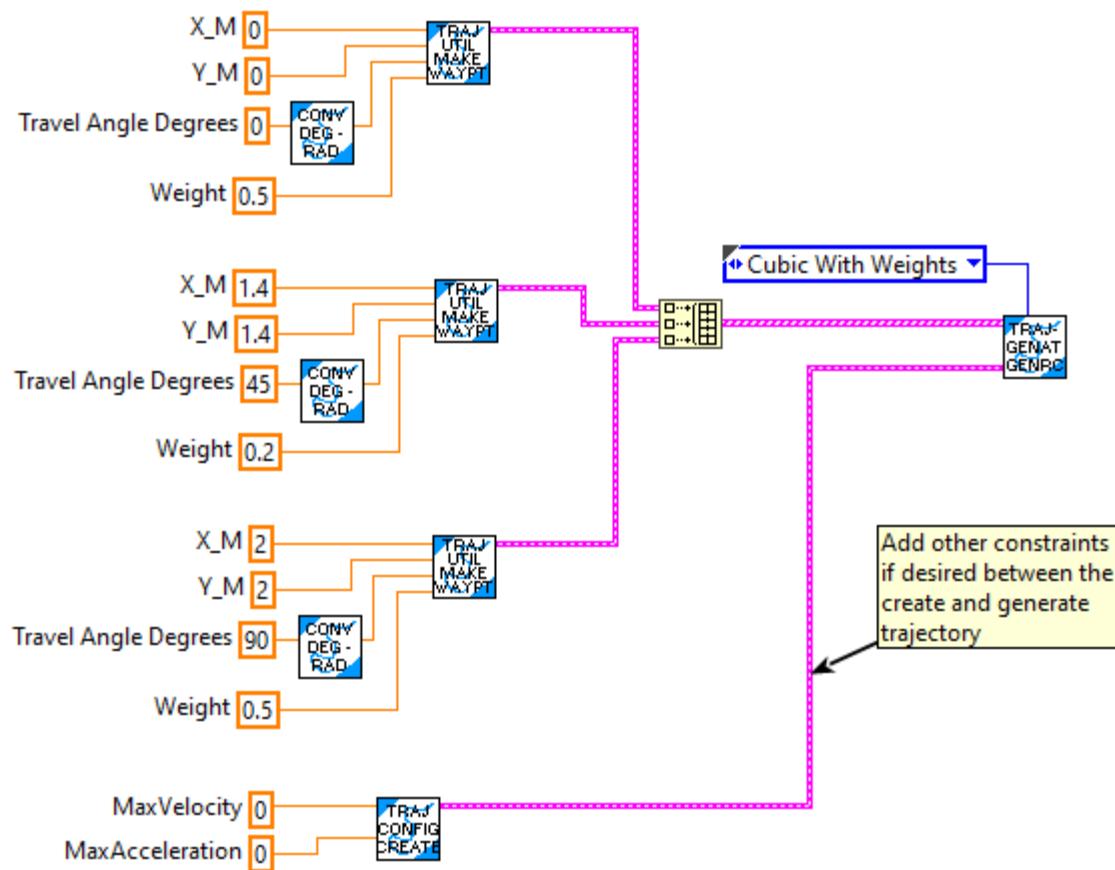
Rectangular Region Constraint -- Using Max Velocity Constraint

Add this after Trajectory Configuration
create and before Trajectory Generate

**macro_TrajectoryConstraint_SwerveKine**

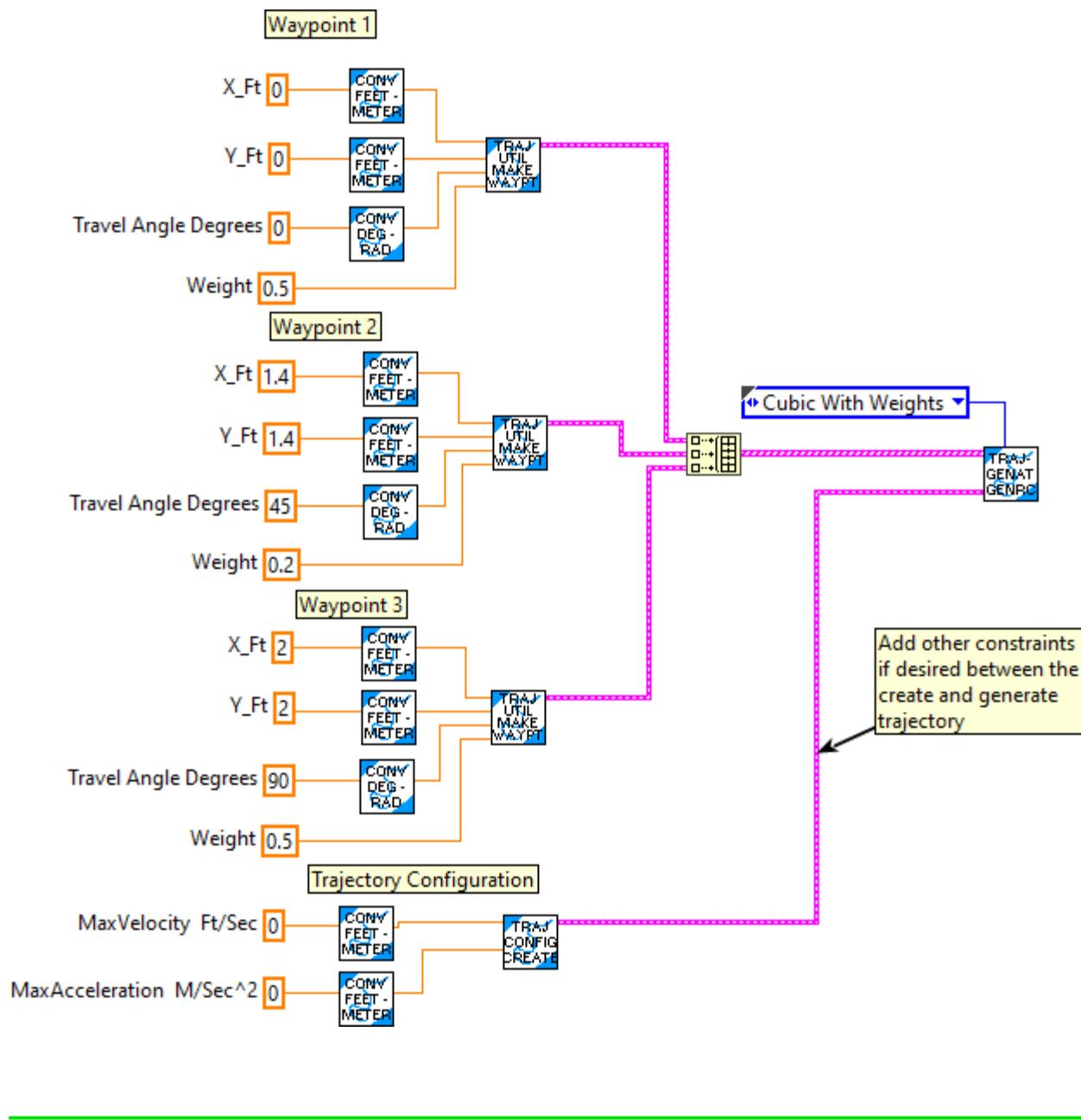


macro_TrajectoryGenerate



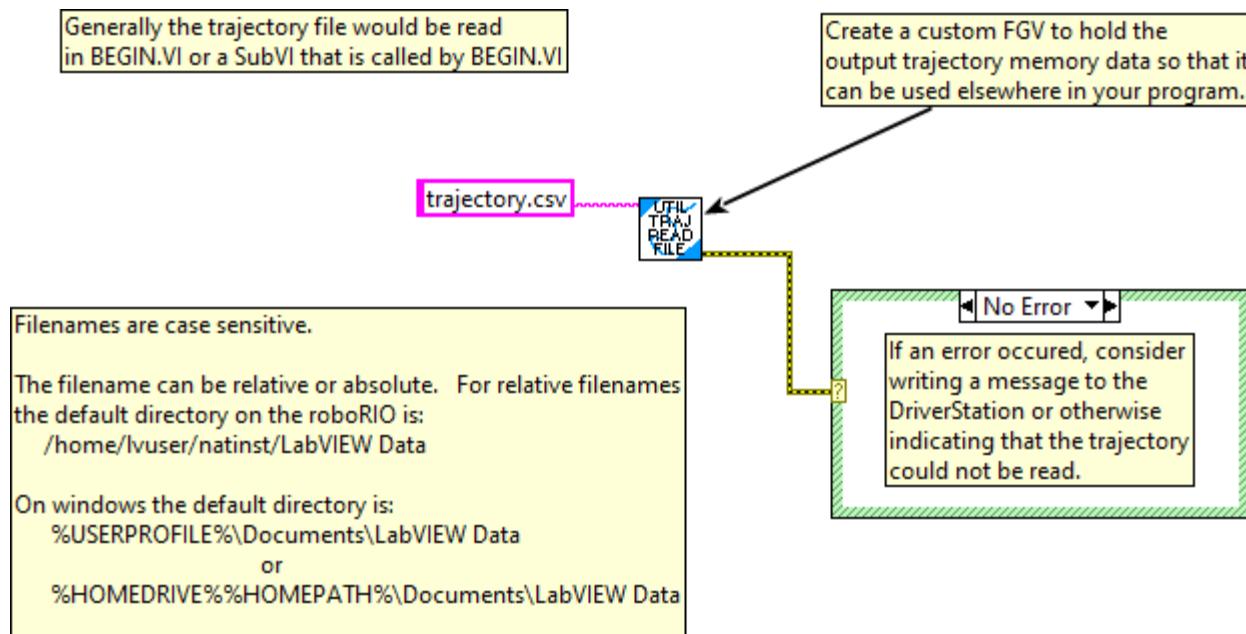
macro_TrajectoryGenerateENG



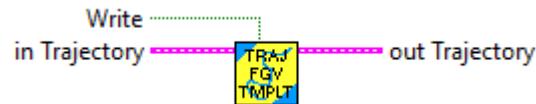


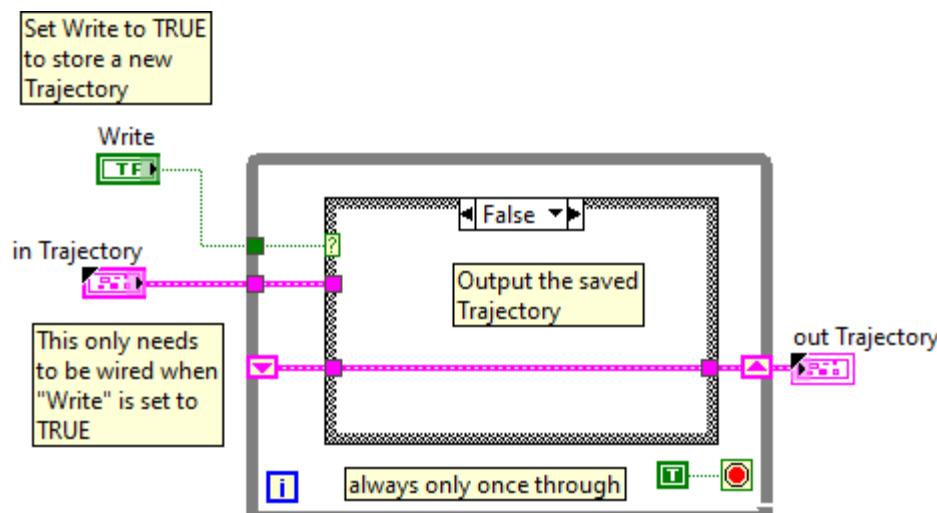
macro_TrajectoryReadFile





macro_Trajectory_FGV_Template

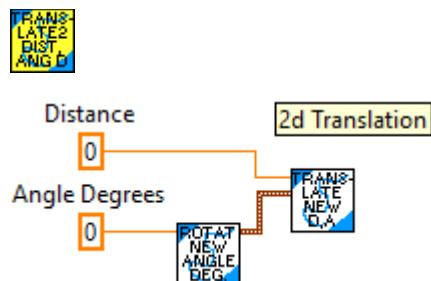




To use this template:

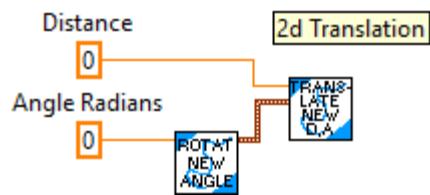
- 1) Create a new VI in your project and copy this code into the new VI.
- 2) Set the "Write" control and the "in Trajectory" as input terminals that are set to "recommended".
- 3) Set the "out Trajectory" as an output terminal.
- 4) Set Execution properties of the new VI to "Non-Reentrant". This will make this VI and the data it contains, global wherever it is used.

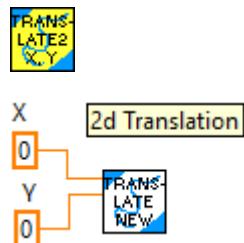
macro_Translation2d_Dist_AngleDeg

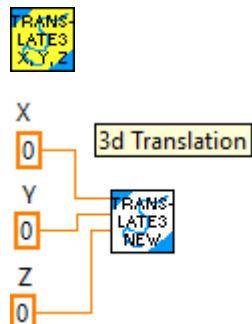


macro_Translation2d_Dist_AngleRad

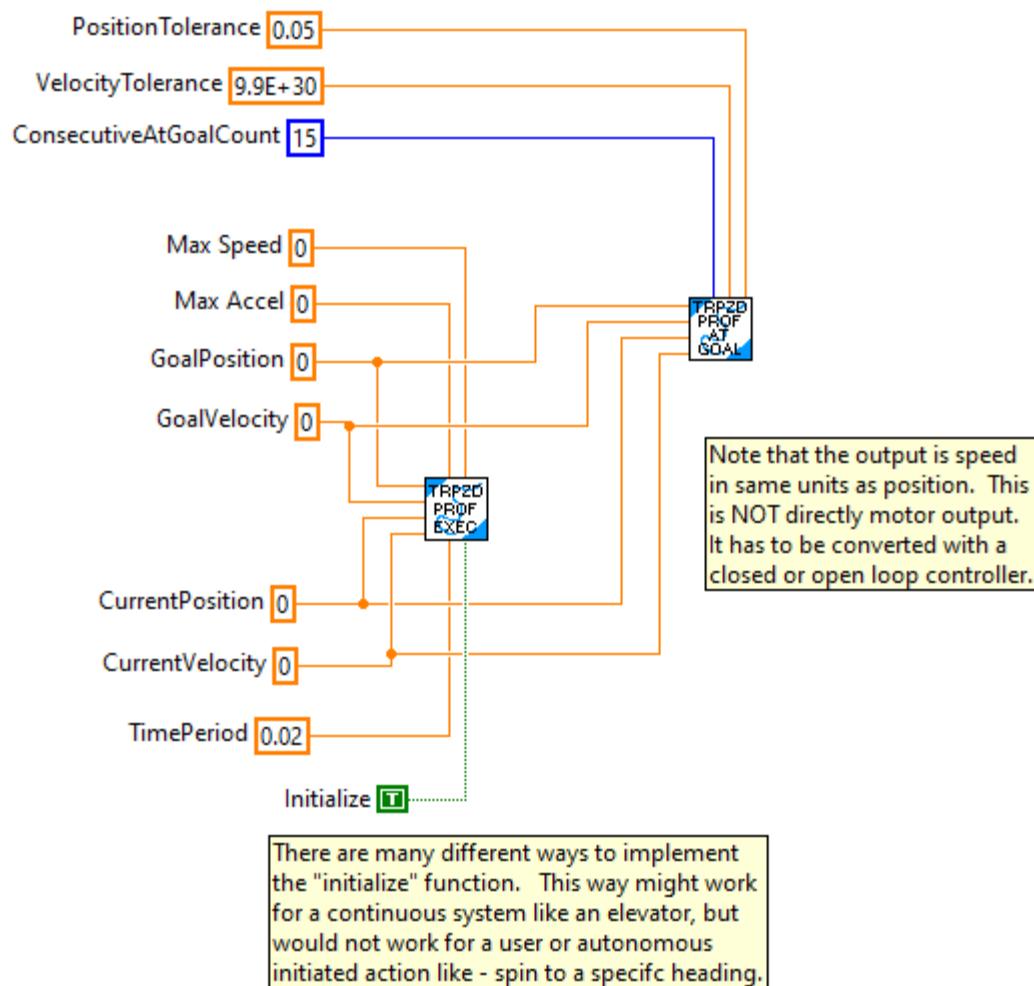




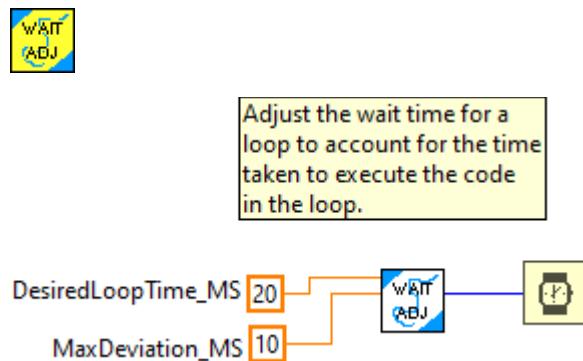
macro_Translation2d_X_Y

macro_Translation3d_X_Y_Z

macro_TrapezoidProfile_Execute

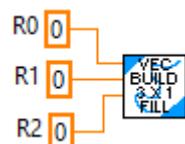


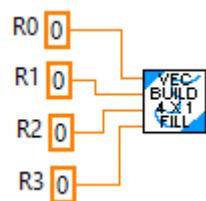
macro_WaitAdjust

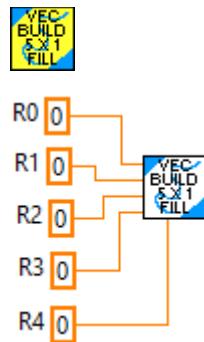
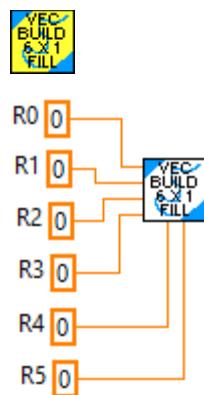
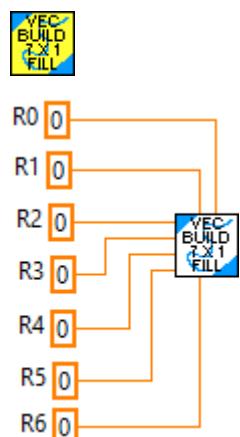


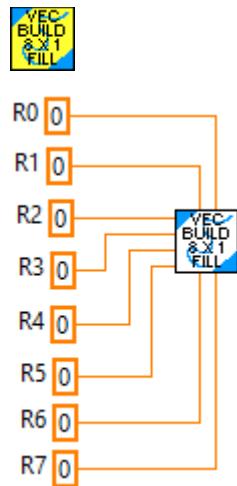
macro_vecBuilder1x1Fill

macro_vecBuilder2x1Fill

macro_vecBuilder3x1Fill

macro_vecBuilder4x1Fill

macro_vecBuilder5x1Fill**macro_vecBuilder6x1Fill****macro_vecBuilder7x1Fill**

macro_vecBuilder8x1Fill**macro_vecBuilder_ArrayBy1Fill**