Trajectory Following

Course Overview

Course Overview

- What is a Trajectory
- Geometry – Measuring where things are.
- Kinematics – Describing robot movement. Converting between robot movement and wheel movement.
- Odometry – Keeping track of where the robot is.
- Trajectory Creation – What are and how to create trajectories.
- Trajectory Execution – How to execute a trajectory

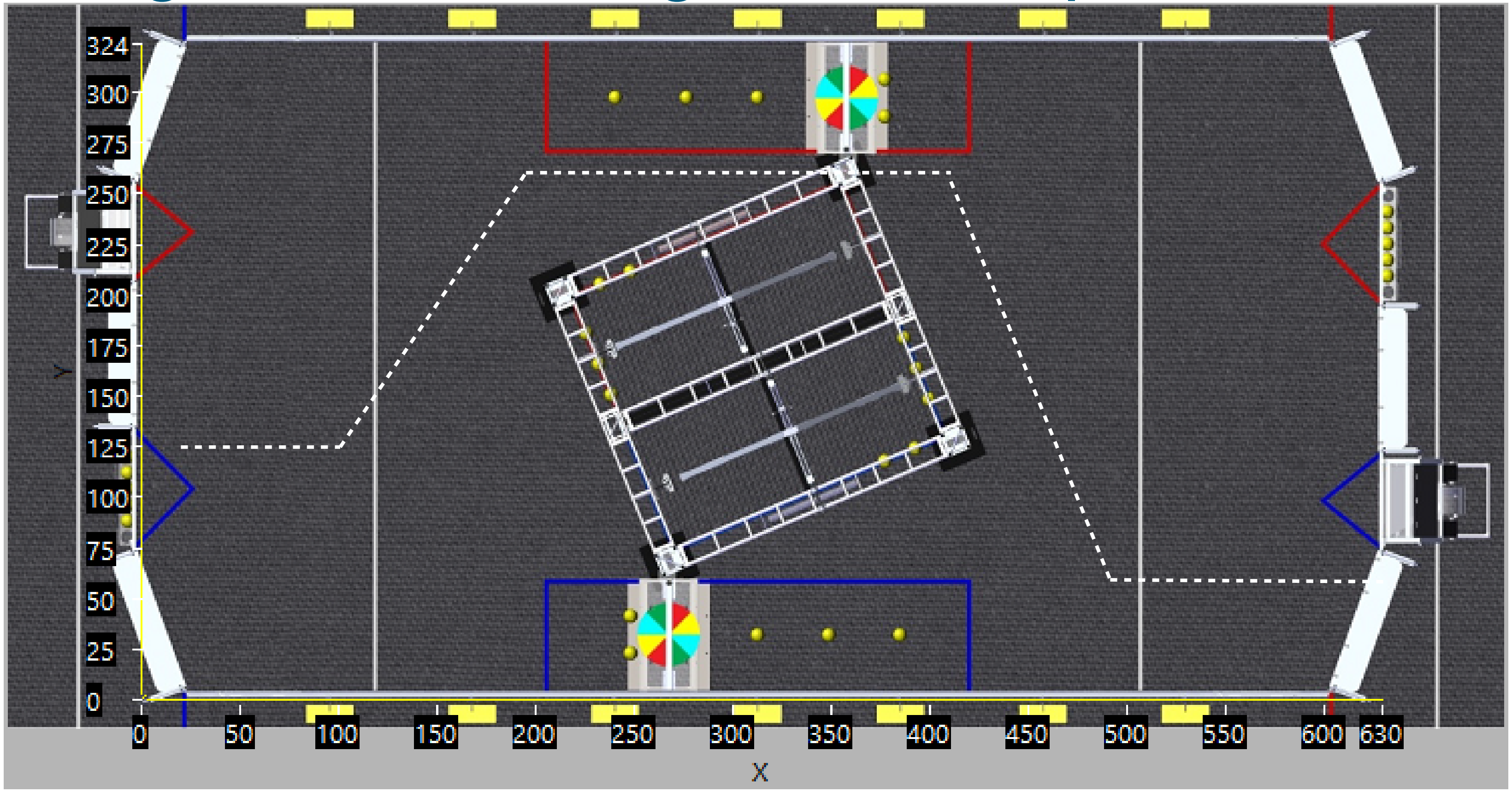
Trajectory Following

What is a Trajectory

What is a Trajectory

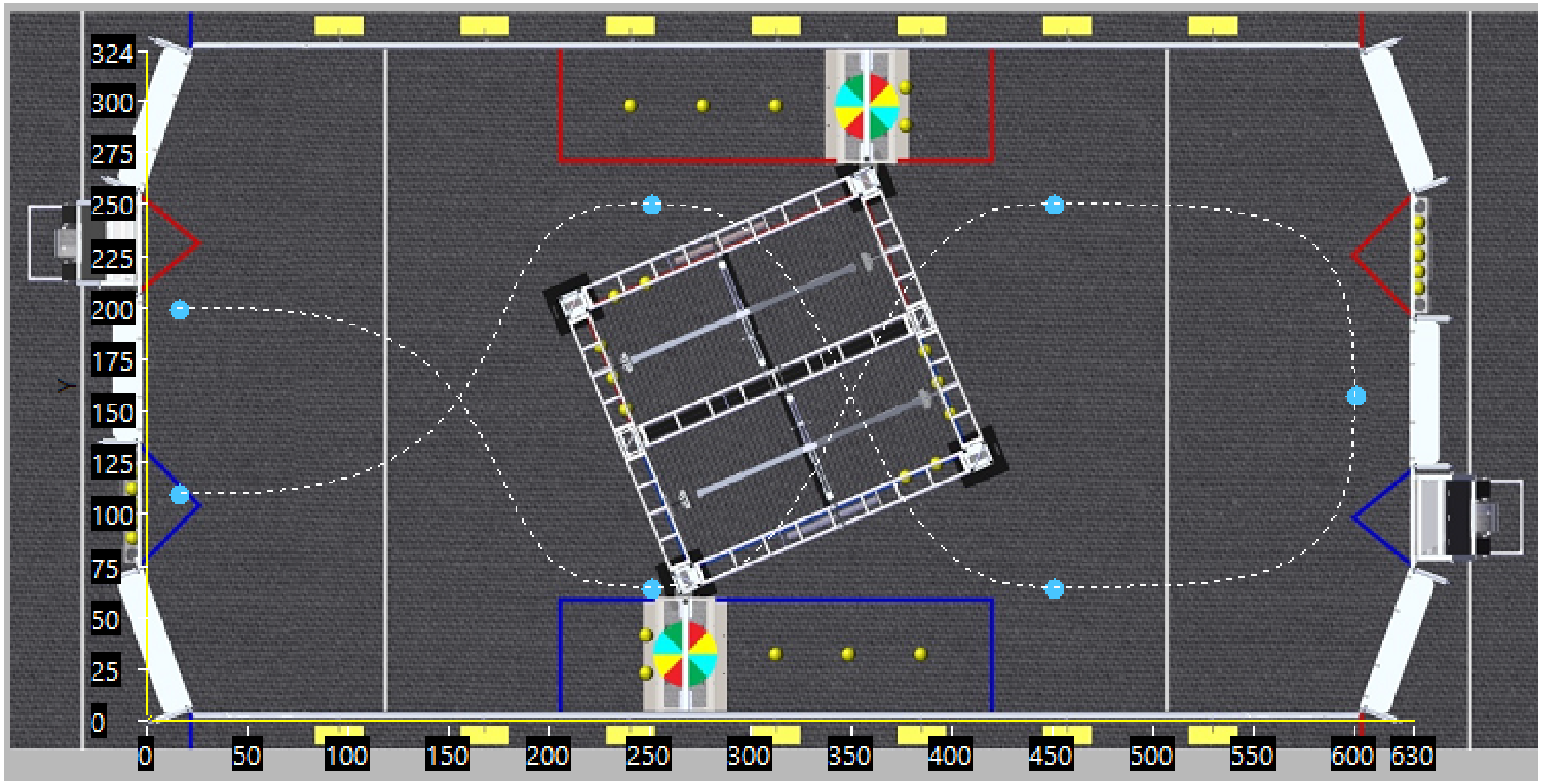
- **Methods to get from one place to another without driver intervention.**
  - Could be during autonomous or teleop competition phase
- **Method 1 - Straight, spin, straight, spin, straight**
  - Advantages - Software easy to implement. Easy to instrument. Easy to change. Easy to tune. Can be accurate
  - Disadvantages – Each move has to come to a stop before making the next move. Potentially slower execution.
- **Method 2 – Trajectory, or Path, Following**
  - Advantages - Potentially faster execution (no stopping along the way). Can follow more complex paths. Can be accurate.
  - Disadvantages – Software harder to implement.   More instrument coordination. Harder to change. Harder to tune.

Straight, Turn, Straight -- Example



- Series of straight and turn (spin) movements.
- Each movement, either go straight, or spin, starts at zero velocity and ends at zero velocity.
- Error of each movement can be minimized. These errors accumulate.

Trajectory Following - Example



- A single continuous movement.
- Velocity is constrained as needed by physical robot characteristics.
- Error of sensors accumulates over time.

Trajectory versus Straight / Turn

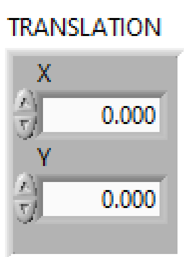
- **Sensors**
  - Both require encoders and gyro.
  - Both require sensors to be accurate.
  - Sensor errors accumulate over time. Shorter movements be more accurate.
- Straight / Turn – This method is potentially more accurate since shorter movements are moved, minimizing sensor (gyro) drift accumulating over time.
- Straight / Turn – Accuracy doesn't depend on accuracy of speed control.
- Trajectory – Can be much quicker.
- Trajectory – Requires accurate drive motor speed control.

Trajectory Following

Geometry

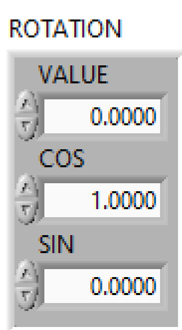
Location - Translation2D

- **Specified by X, and Y**
  - In relation to robot, X increases forward, Y increases to the left.
  - In relation to a field drawing X increases to the right, Y increases up.
  - Units are SI (meters)
- **Translation2D Data**



Orientation – Rotation2D

- **Which way the front of the robot is facing. This is also the direction of travel.**
  - Specified by angle of rotation
  - Values increase – counterclockwise. The normal gyro reading in previous years has been backwards from this!
  - Units are SI (radians). >> Sometimes degrees – read the documentation!
- **Rotation2D Data**
  - Value is angle. Sin and Cos are stored so they don't have to be recalculated.

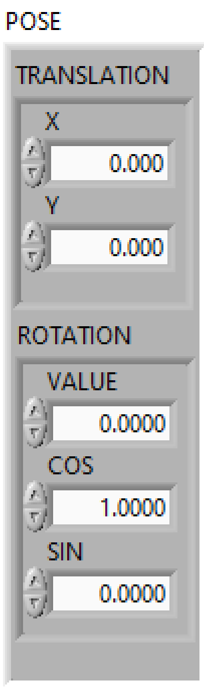


Robot – Orientation, Direction of Travel

- **Robot orientation and direction of travel**
  - These can be different for Mecanum and Swerve drive robots. (Enough said for now…but) The WPILIB doesn’t explicitly separate these things. Trajectories don’t contain an orientation component separate from the heading component. (The assumption is that the orientation should remain constant during travel.) (heading angle increases counter-clockwise) Future versions may change that.

Position – Pose2D

- **Complete position contains both Translation and Rotation components.**
- **Pose2D Data**



Absolute and Relative

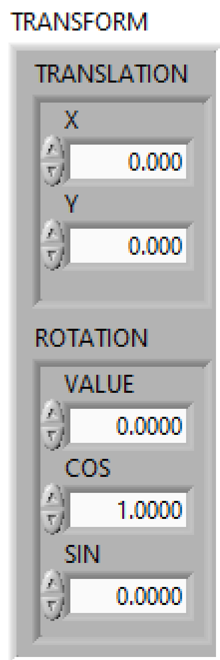
- **Location, Orientation, Heading, Position can be specified as absolute or relative.**
  - The absolute Location origin (0,0) can vary, but is usually the bottom left of the field (pathfinder and the trajectory library use this, Pathweaver uses the top left.) Whatever is used, be consistent.
  - The absolute Orientation and Heading angle origin (0), always both the same, point in the X direction.
  - Relative means the origin is relative to something else.   Generally the something else is the point inside the robot perimeter that the robot spins around around. (For Mecanum and Swerve robots the physical center of the robot is used.)
  - For robot relative, the orientation is 0 in the direction that the front of the robot faces.

Geometry Math

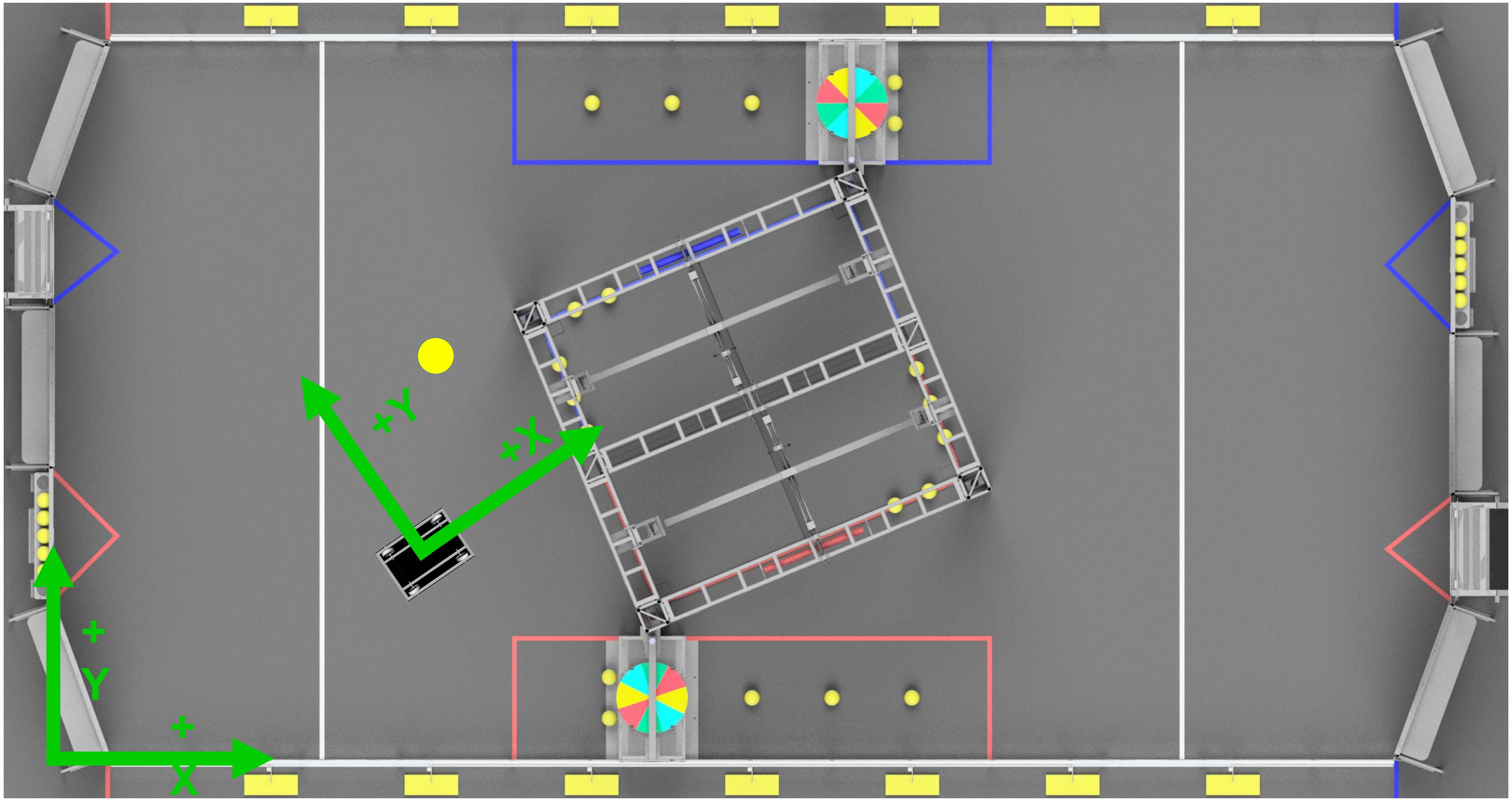
- **Translation2D**
  - Add, Subtract, Scalar multiply, Scalar divide, Rotate
- **Rotation2D**
  - Add, Subtract, Scalar multiply, Scalar divide
- **Transformation2D**
  - Operates on a Pose2D
  - Contains a Translation2D and Rotation2D
- **Twist2D**
  - Represents the change in distance along an arc.
  - Used internally in Odometry

Transformation 2D

- **Transformation2D**
  - Acts upon a Pose2D
  - Transforming a Pose2d by a Transform2d rotates the translation component of the transform by the rotation of the pose, and then adds the rotated translation component and the rotation component to the pose.
- **Transform2D Data**



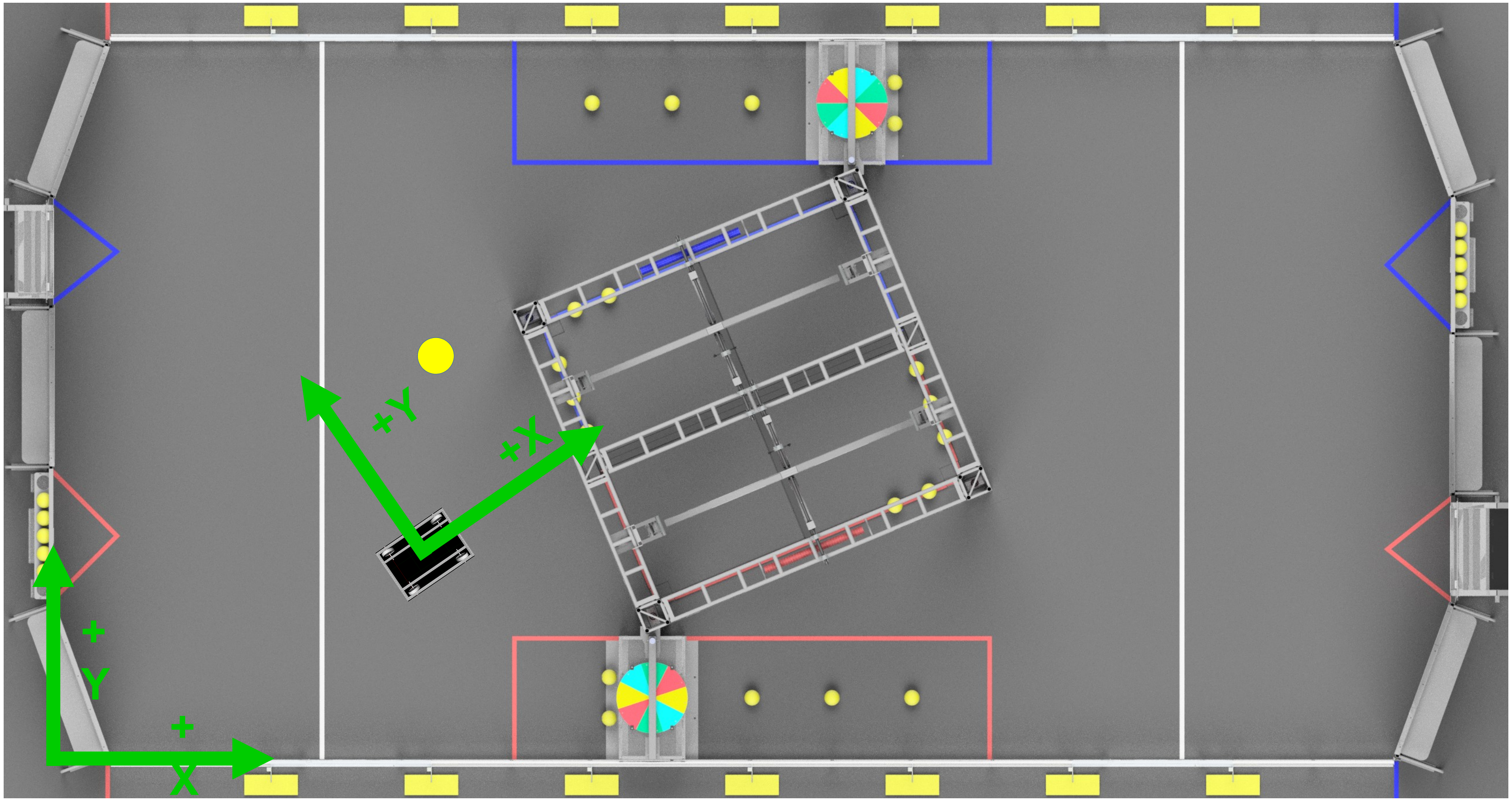
Absolute Geometry Example



- Absolute robot position (approx.) 15 ft, 10 ft, 40 deg. (X, Y, heading) [Pose2D]
- Absolute ball position (approx.) 15 ft, 18 ft. (X, Y) [Translation2D] (Ball has no rotation.)

Note: Units here are feet and degrees. All the trajectory Trajectory Lib routines, use meters and radians. Conversion must be performed prior to calling them.

Relative Geometry Example



- Robot relative robot position 0 ft, 0 ft, 0 deg. (X, Y, heading) [Pose2D]
- Robot relative ball position (aprox.) 3 ft, 4 ft. (X, Y) [Translation2D] (Ball has no rotation.)

Note: Units here are feet and degrees. All the trajectory Trajectory Lib routines, use meters and radians. Conversion must be performed prior to calling them.

Exercises

- **Open Sample to do some exercises**
  - demo_geometry.vi

Geometry – Additional Information

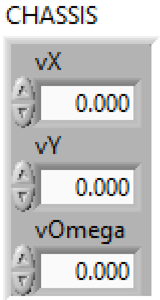
- https://docs.wpilib.org/en/stable/docs/software/advanced-controls/geometry/index.html

Trajectory Following

Kinematics

What is Kinematics

- **Kinematics – Describes robot movement**
  - Consists of X, Y, and rotational movement.
  - vX – Forward, backward movement (m/s)
  - vY – Side side movement (m/s)
  - vOmega (rotational) – How much the robot is turning. (radians/sec). Counterclockwise is plus.
    - Differential drive robots (like we use) always have have a Y movement of 0.0.
  - Contained in ChassisSpeed
    - Don't confuse with robot position, which uses Pose2D



Conversion to from wheel speeds

- **Need to convert desired ChassisSpeed to wheel speeds.**
  - Routines exist to do this for each different type of standard drive train – differential, swerve, mechanum.
- **Need to convert wheel speeds to ChassisSpeed**
  - Routines exist to do this for each different type of standard drive train – differential, swerve, mechanum.
- **These require physical robot characteristics**
  - Location of wheels, and type of robot.
- **Units are all SI (meters, radians)**
- **Values increase – forward, left, counterclockwise**

Kinematics – Additional Information

- https://docs.wpilib.org/en/stable/docs/software/kinematics-and-odometry/index.html

Trajectory Following

Odometry

Odometry – Measuring Robot's Position

- **Odometry – The process of measuring the robot's position.**
  - Can be an absolute field position, or relative to a particular starting point.
    - An absolute position needs an accurate starting position and heading [Pose2D].
  - Calculates updated position from robot sensors – wheel encoders, and gyro
  - The accuracy of the robot's position gets worse over time, even if the sensor measurements are calibrated really well. As such Odometry can generally only be used for short durations of time. This is generally enough to accurately execute a trajectory.

Odometry - Process

- **Set initial position**
  - Set to a known absolute position or set to 0,0,0 for relative movement.
- **Periodically call the Update routine**
  - Provide Gyro and drive encoder readings. Update routine will calculate new robot position.
- **Notes:**
  - There are different routines to update the position for each different type of robot drive.
  - Some documentation recommends resetting gyro and encoder readings. Be careful of this!!! If these readers are used elsewhere, those routines will likely break.
  - Multiple instances of odometry may be running at the same time. For example one for position since start of match and one for position since start of trajectory execution.

Odometry - Issues

- **Gyro Sensor Drift**
  - Gyro readings drift over time. Can only use for a limited time. Good gyro calibration required.
- **Drive wheel encoder calibration**
  - Encoders need to be carefully calibrated so each is accurate.
- **Robot “skid”**
  - Robot wheels skid while turning and potentially while accelerating and decelerating.
- **Initial robot placement**
  - Initial position and rotation of robot is very important. Rotational errors get worse as distance increases!

Odometry – Additional Information

- https://docs.wpilib.org/en/stable/docs/software/kinematics-and-odometry/index.html

Trajectory Following

Trajectory Basics and Creation

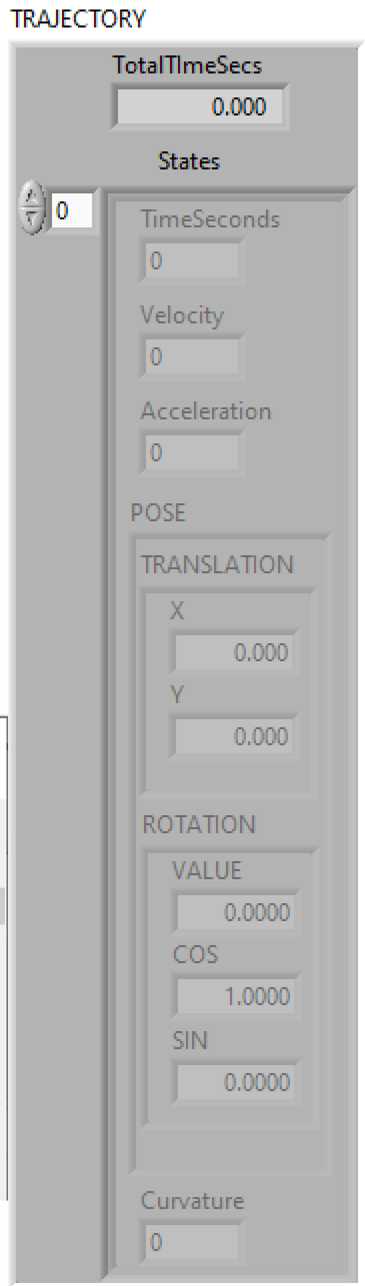
Trajectory - Basics

- **A Trajectory is a list of relative times, expected positions, and desired robot chassis speed**
- **Sample, file based, trajectory listing**
- **Time interval isn't constant. Use interpolation to obtain a sample at a specific time.**

2021-sample-challenge-quintic.csv - Notepad

File Edit Format View Help

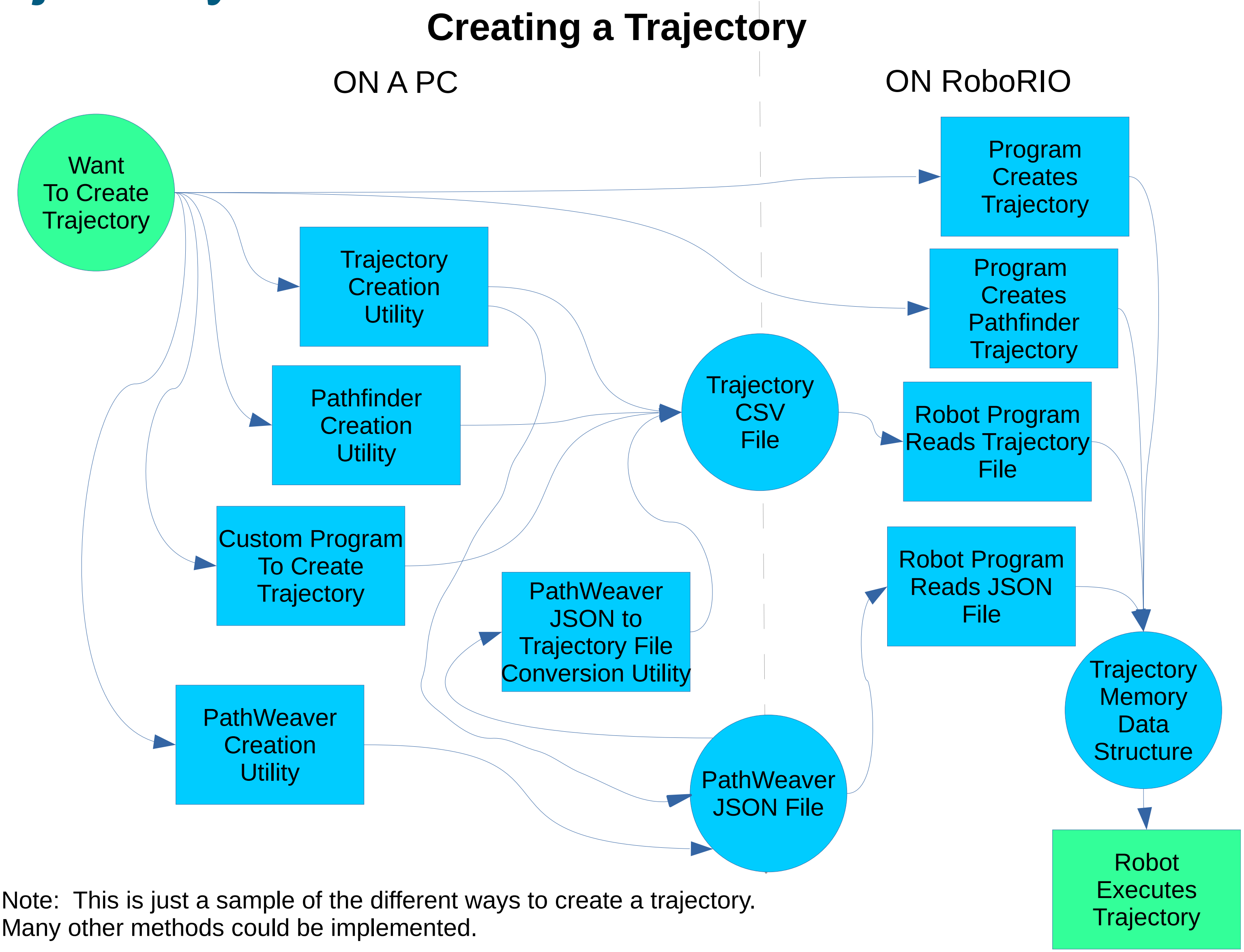
```
#-----------------------------------------------------------------------------------------------
#-- Time,        Velocity,    Accel,       X pos,       Y pos,        Heading,   Curvature, Comment
#-- Seconds,     m/s,         m/s^2,       m,           m,            radians,   radians/meter,
   0.00000,     0.00000,     0.63500,     0.00000,     0.00000,      0.00000,     0.00000,Segment:  1
   0.45081,     0.28626,     0.63500,     0.06453,    -0.00002,     -0.00075,    -0.02304,Segment:  2
   0.63738,     0.40474,     0.63500,     0.12899,    -0.00013,     -0.00295,    -0.04519,Segment:  3
   0.78032,     0.49550,     0.63500,     0.19332,    -0.00043,     -0.00655,    -0.06664,Segment:  4
   0.90053,     0.57184,     0.63500,     0.25748,    -0.00100,     -0.01150,    -0.08754,Segment:  5
   1.00613,     0.63889,     0.63500,     0.32139,    -0.00193,     -0.01775,    -0.10804,Segment:  6
   1.10124,     0.69929,     0.63500,     0.38502,    -0.00329,     -0.02527,    -0.12825,Segment:  7
   1.26902,     0.80582,     0.63500,     0.51120,    -0.00761,     -0.04398,    -0.16814,Segment:  8
   1.41528,     0.89870,     0.63500,     0.63567,    -0.01449,     -0.06741,    -0.20773,Segment:  9
   1.54593,     0.98167,     0.63500,     0.75810,    -0.02443,     -0.09535,    -0.24724,Segment:  10
   1.66450,     1.05696,     0.63500,     0.87822,    -0.03783,     -0.12760,    -0.28658,Segment:  11
   1.77336,     1.12608,     0.31827,     0.99579,    -0.05504,     -0.16396,    -0.32533,Segment:  12
```



Trajectory - Creation

- **Trajectories can be created.**
  - Manually (not recommended)
  - Programatically (C++, Java, LabVIEW)
  - Using a tool (PathFinder, PathWeaver, Trajectory Creation)
    - PathFinder was the original tool. Less used recently.
    - PathWeaver creates JSON file. 2020 version didn't use constraints, but has variable weights.
    - Trajectory Creation creates CSV file.

Trajectory Creation / Use - Flowchart



Note: This is just a sample of the different ways to create a trajectory.
Many other methods could be implemented.

Trajectory - Defined

- **Trajectories are defined by “waypoints”.**
  - Starting and ending positions. (Pose2D)
  - Positions along the desired path. (Pose2D)
  - Optionally weights are defined for each point. This determines how “straight” the path goes through a particular point.
- **Trajectories are calculated from waypoints using splines.**
  - Cubic Hermite Splines. (These don’t use interior waypoint headings, just X, Y)
  - Quintic Hermite Spines
  - It is worth trying both

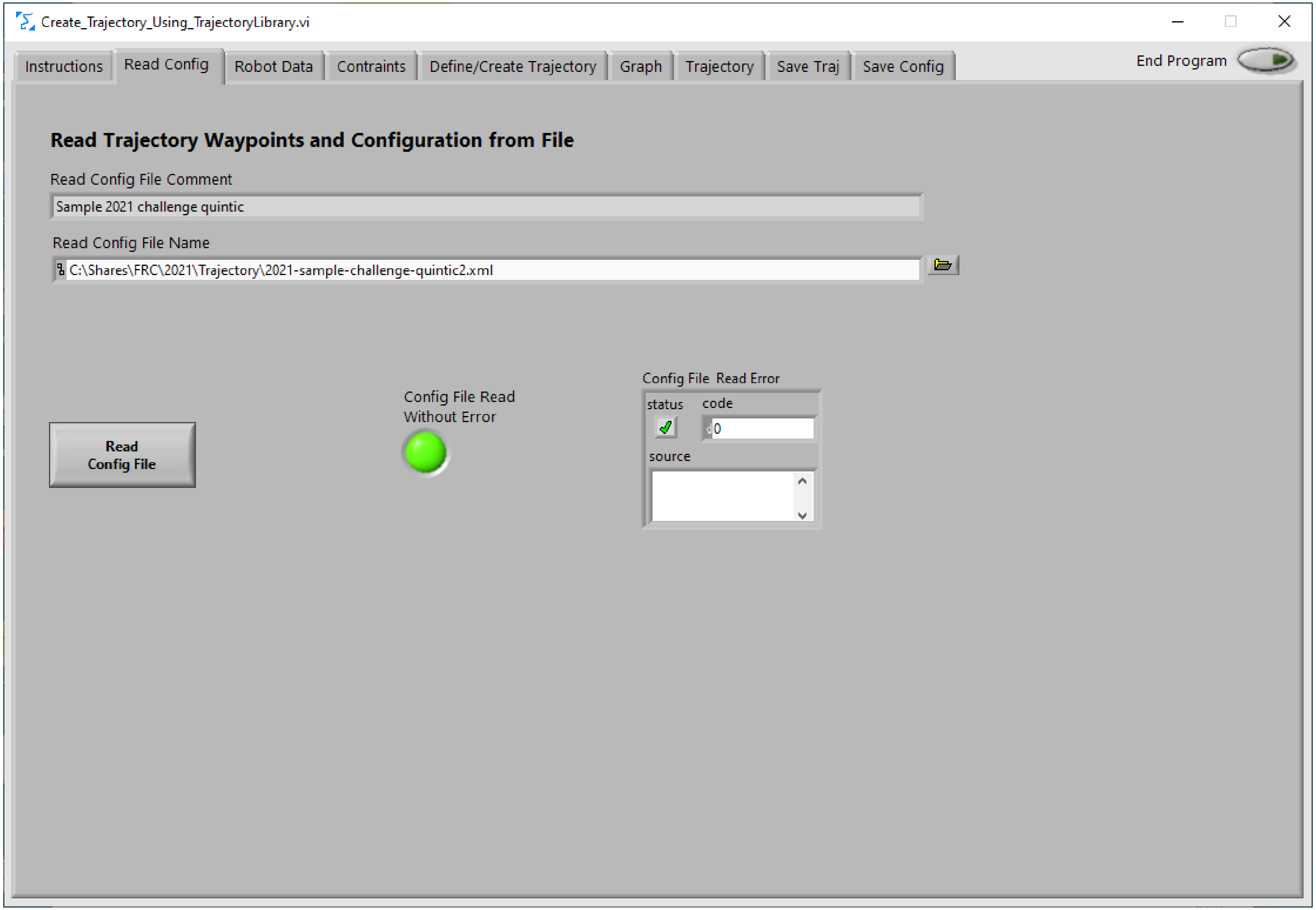
Trajectory – Physical Data / Constraints

- **Trajectory creation requires robot physical data**
  - Differential drive wheelbase
- **Constraints define robot physical limitations**
  - Max speed, max acceleration, max centripital acceleration, etc.
  - Can be estimated or obtained through robot characterization testing.

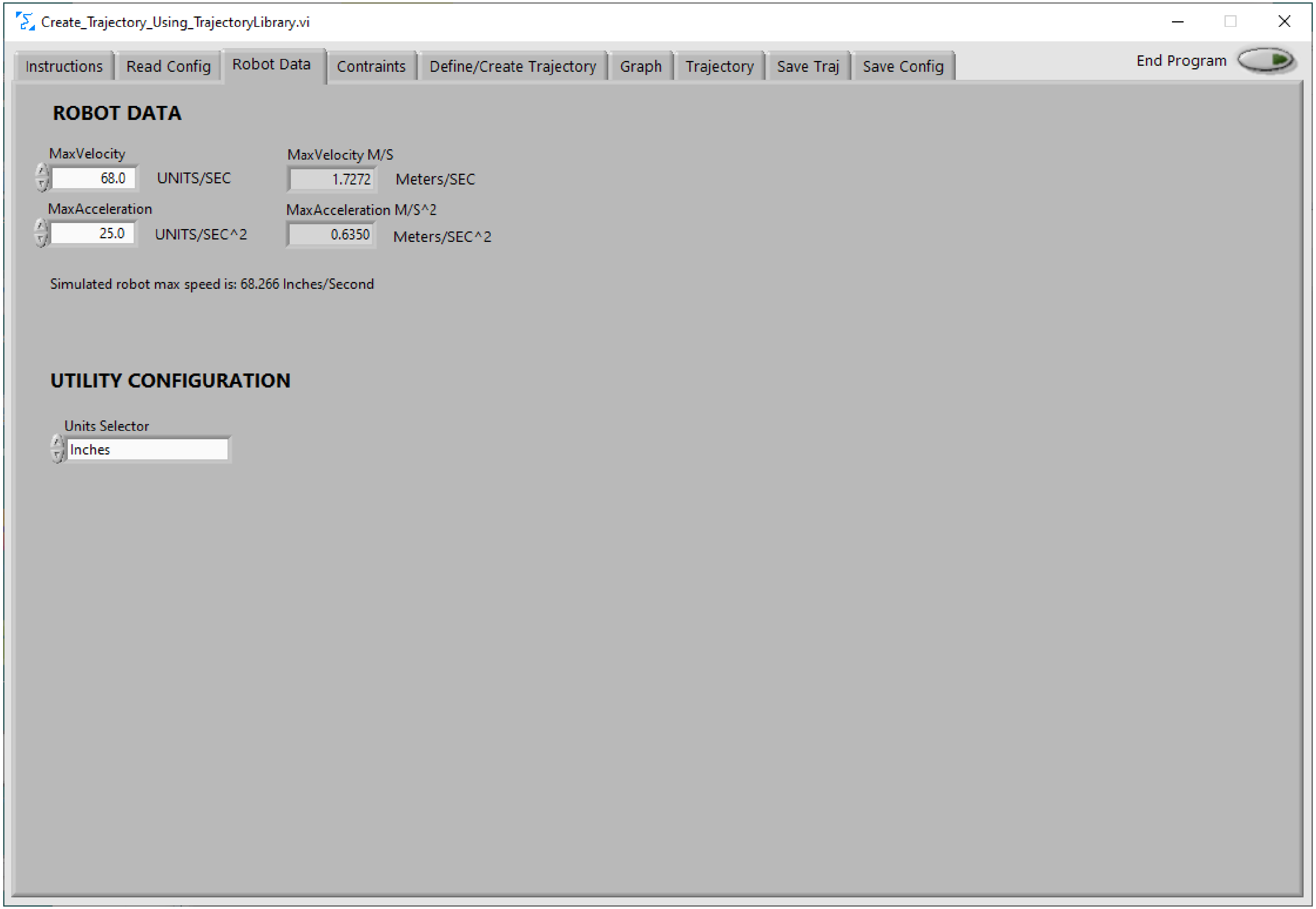
Trajectory Creation Demo

- **Demo trajectory creation utility. Other utilities exist:**
  - PathWeaver
  - PathFinder
  - Some teams made their own.

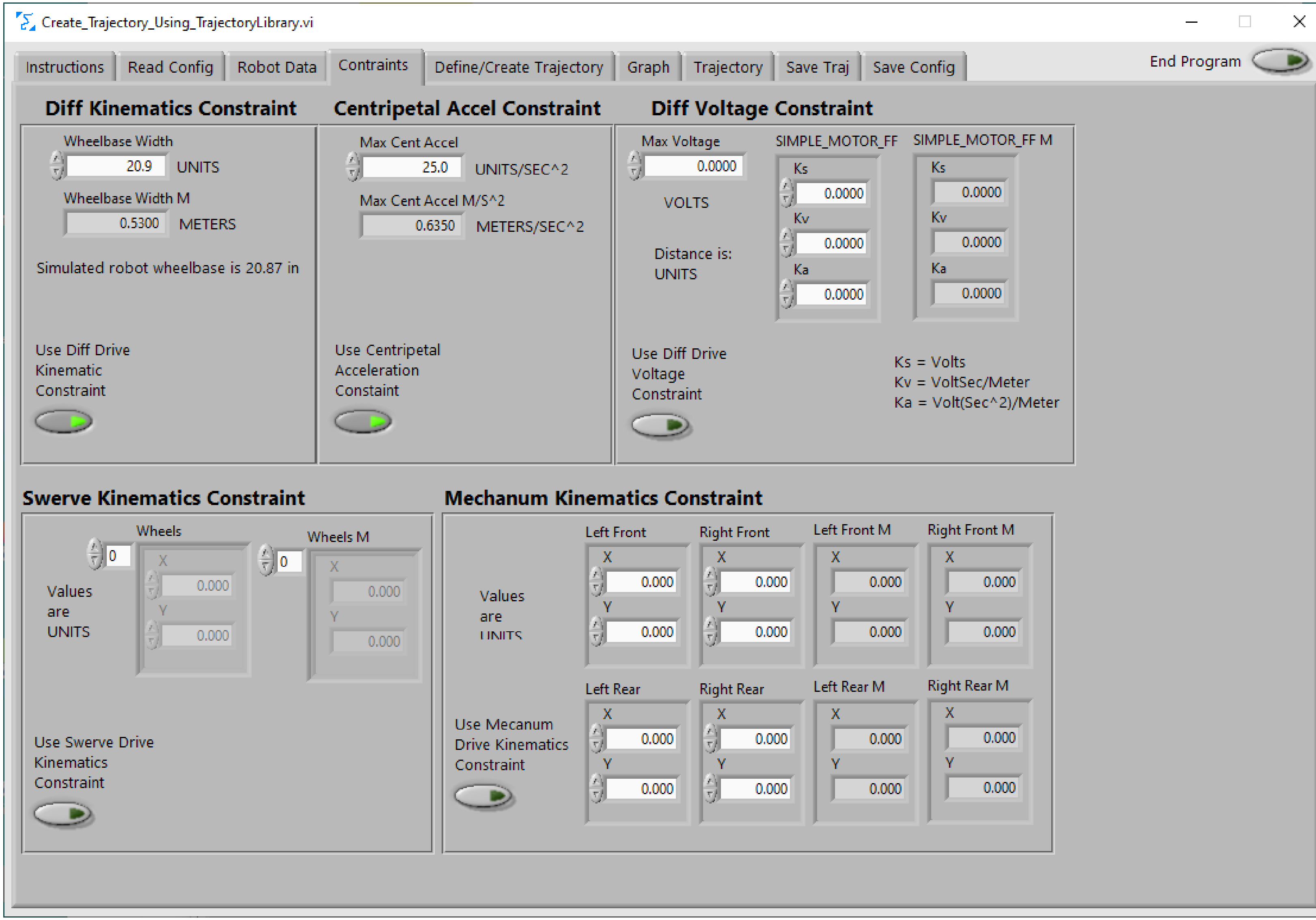
Trajectory Creation – Read configuration



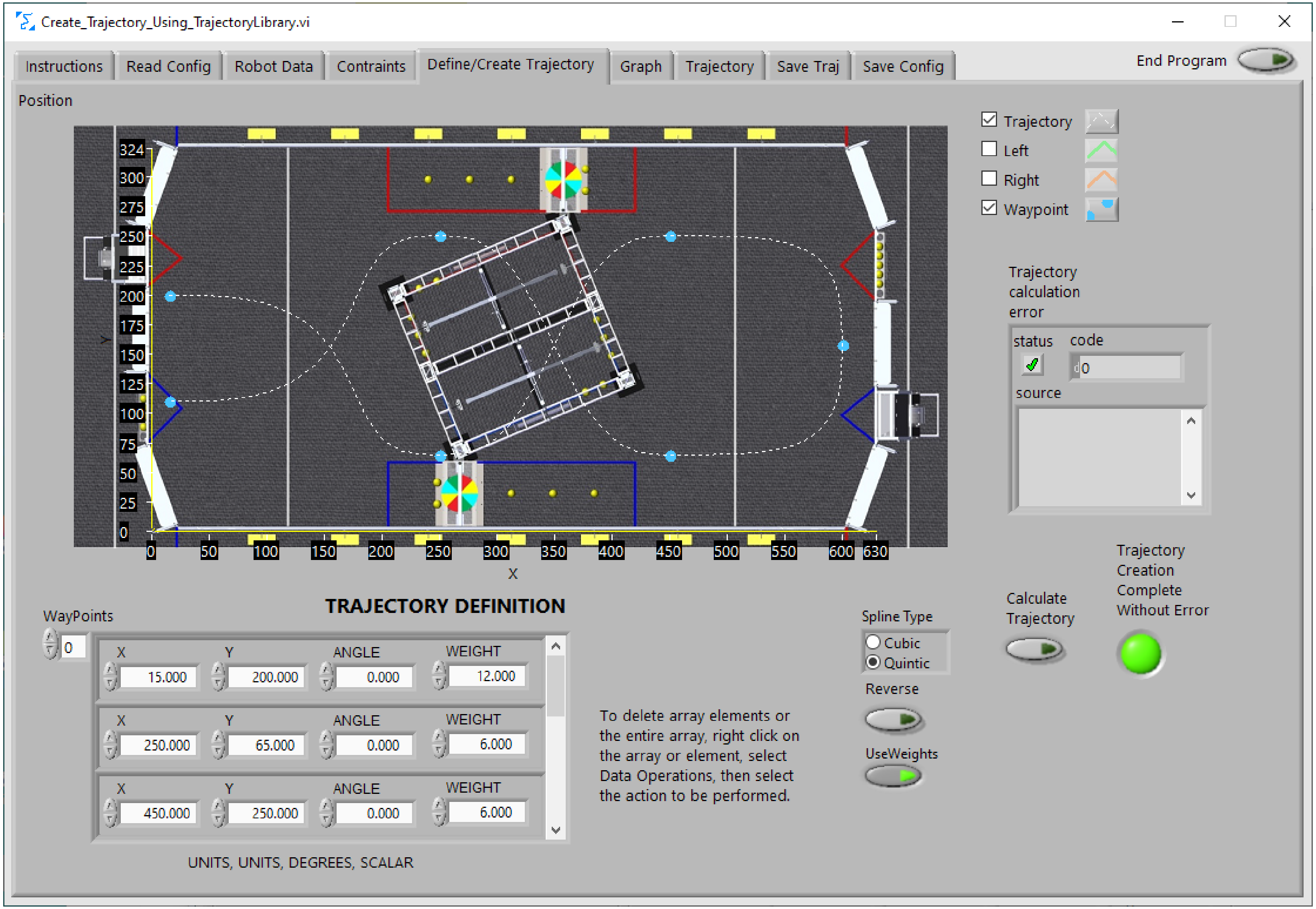
Trajectory Creation – Robot Data



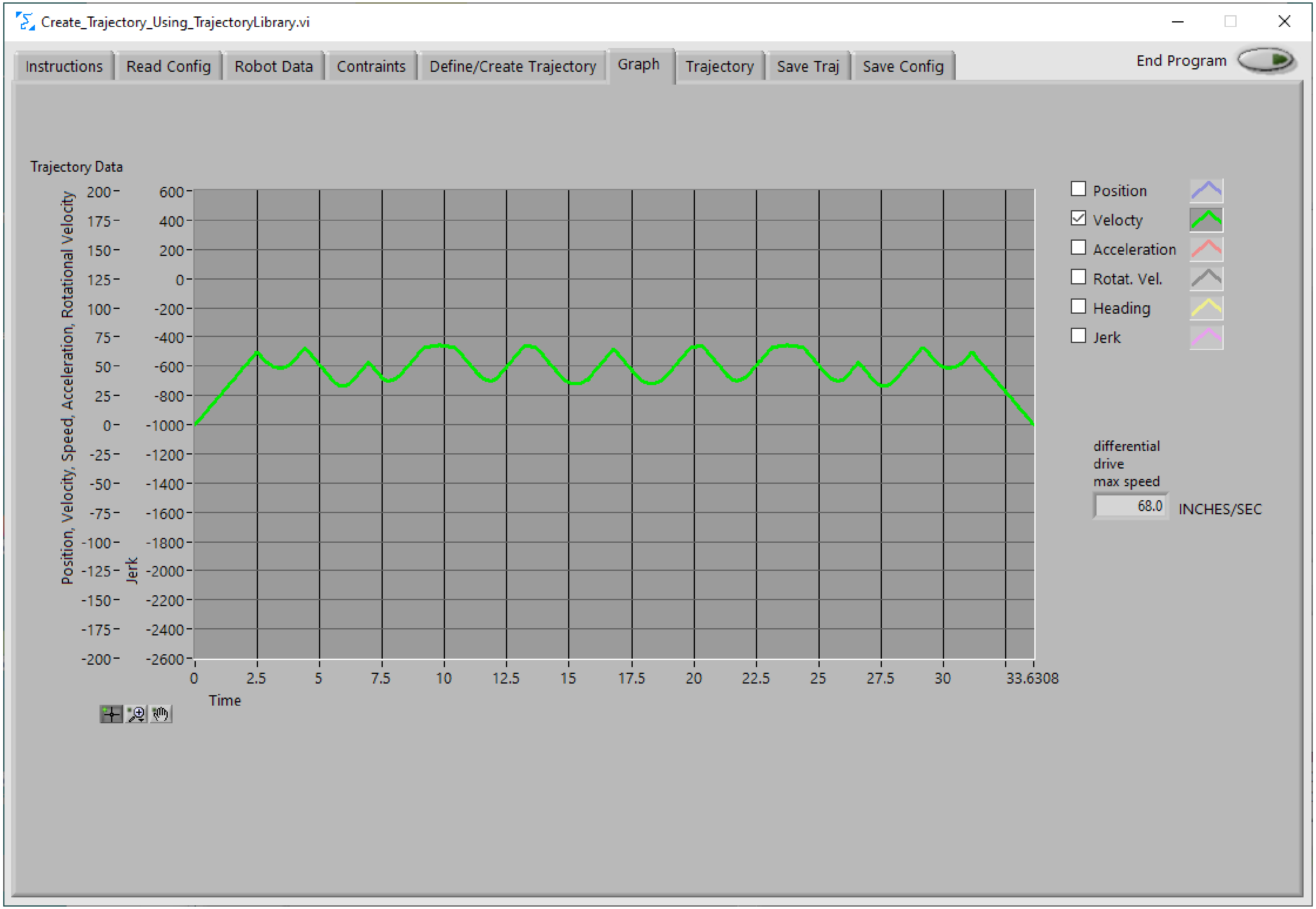
Trajectory Creation – Define Constraints



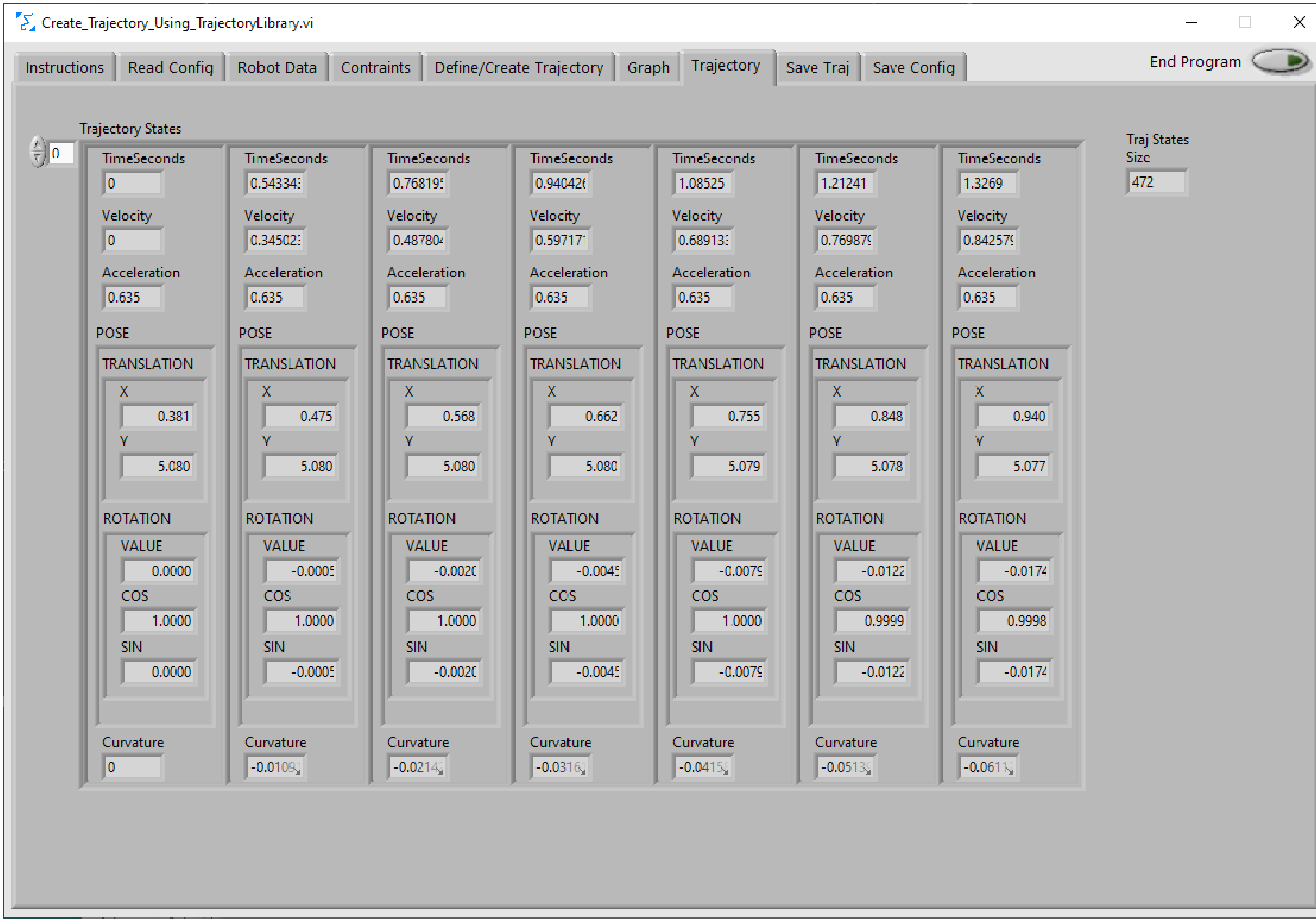
Trajectory Creation – Set Waypoints, Create



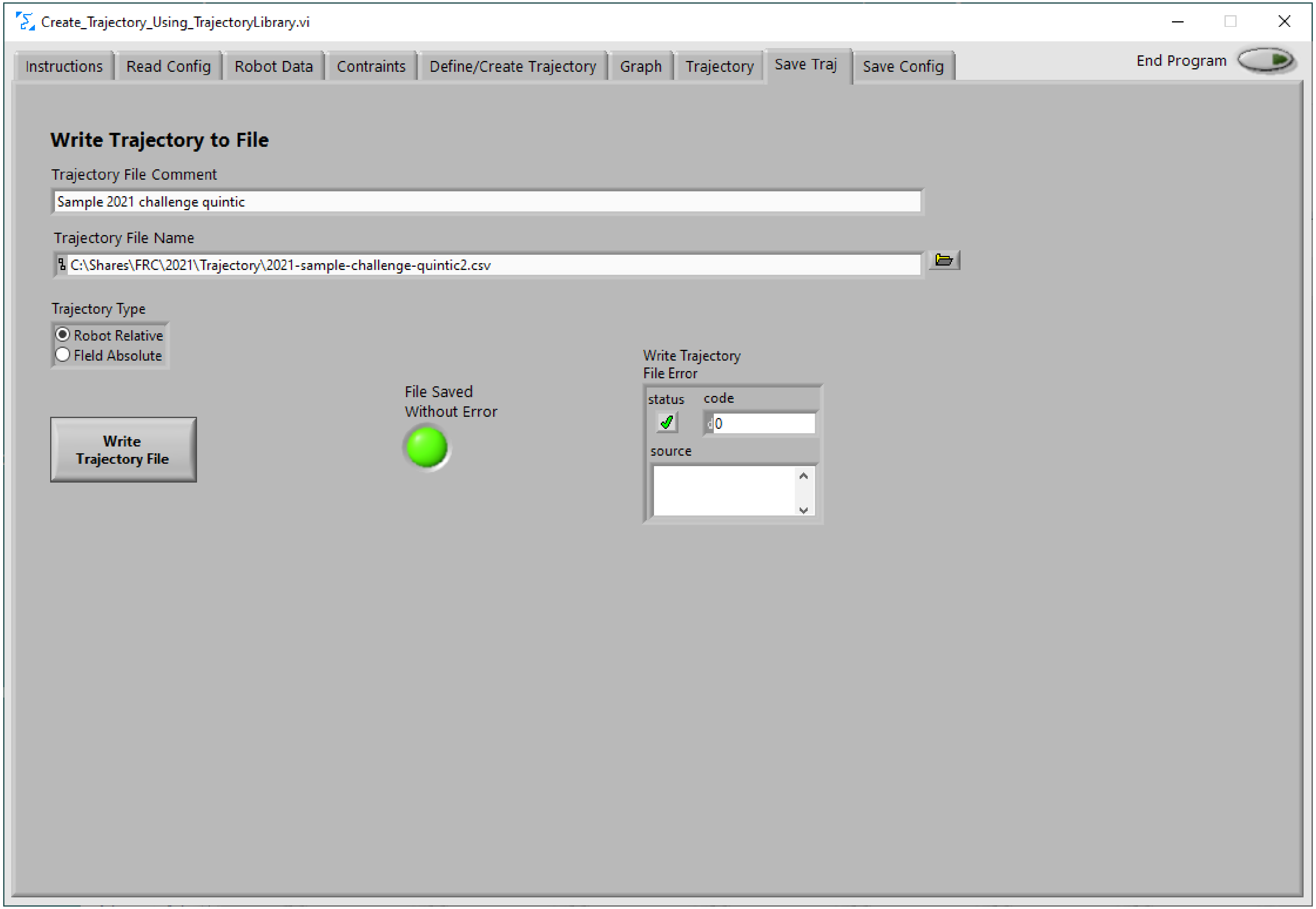
Trajectory Creation – Review Trajectory



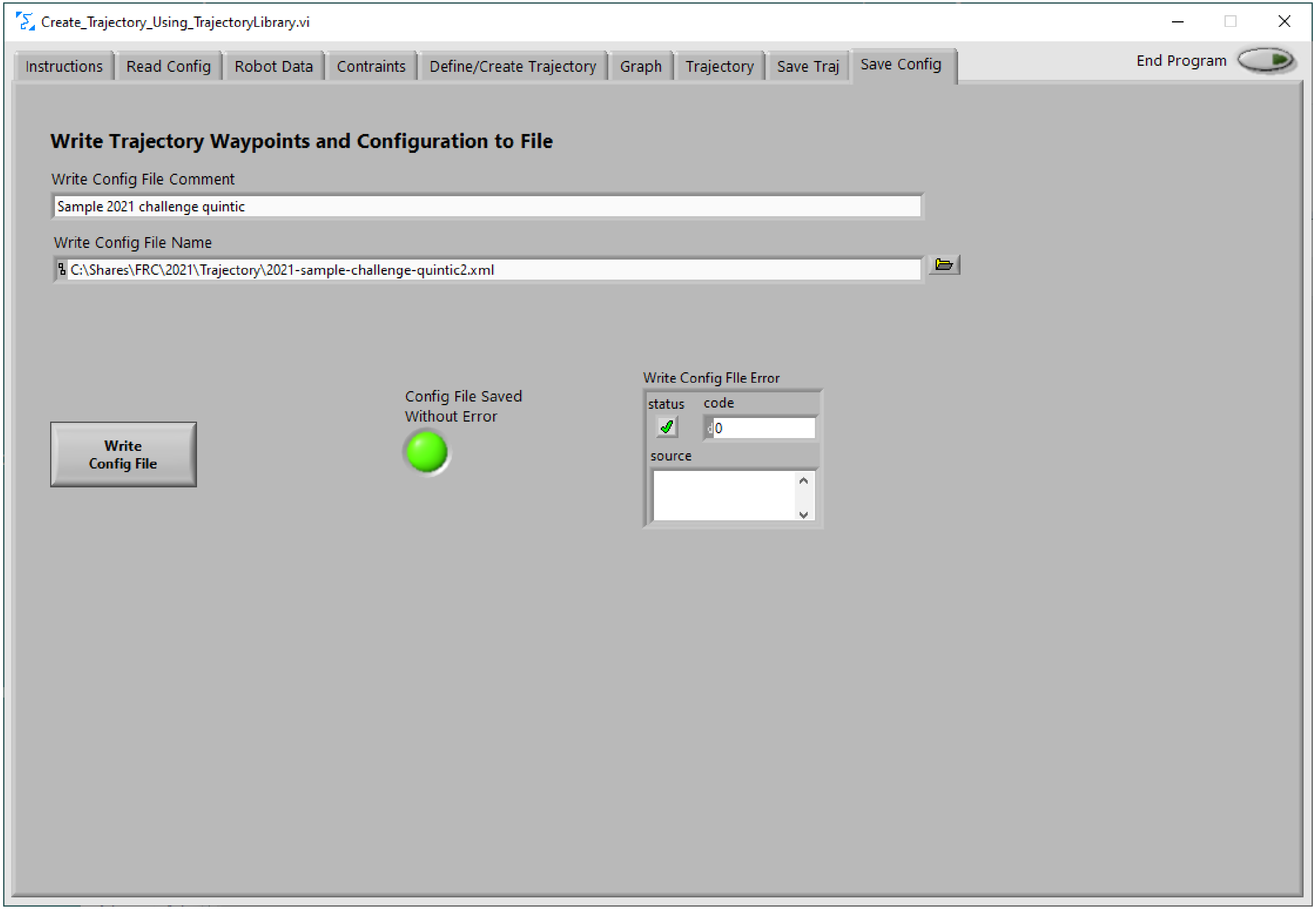
Trajectory Creation – Review Trajectory



Trajectory Creation – Write Trajectory



Trajectory Creation – Write configuration



Trajectory Creation Process

- **Define waypoints (X, Y, Rotation, and weight)**
- **Create smooth path between waypoints using splines**
  - There are many curvefitting techniques. Splines are simple and fast.
  - This does NOT include timing or robot speed information.
- **Create trajectory by calculating desired robot speed along path ensuring constraints are not violated.**
  - Sample times, velocities, acceleration, and curvature are added.
  - This is a multi-pass process.
- **Populate memory data structure and/or write to a file.**

Robot Constraint Characterization

- **Robot testing may require trajectory revisions**
  - Constraints and robot parameters may need to be adjusted until trajectory execution is optimized.
- **Hints**
  - May need to artificially change wheelbase dimension to accommodate “skid” while turning.
    - Note: Must use physical wheel base when converting between robot and chassis speeds.
  - Begin with slow maximum speed and revise to faster speeds.
  - Disable closed loop, ramsete, control to find best trajectory constraints.
  - Do testing to find maximum speed, accleration, and turning speed.
    - There is a characterization program. Read about it. The data will have to be translated to whatever tool is used to create trajectories.

Trajectory Creation – Additional Information

- https://docs.wpilib.org/en/stable/docs/software/wpilib-tools/pathweaver/introduction.html

Trajectory Following

Trajectory Execution

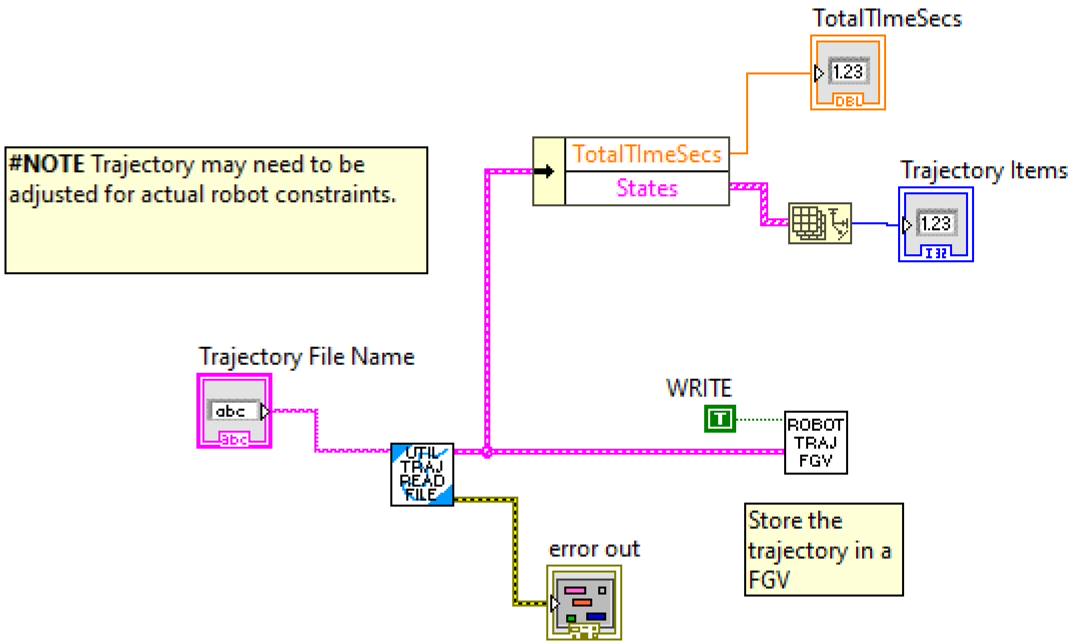
Controlling Trajectory Execution - Ramsete

- **Controlling Differential Drive Trajectory Execution**
  - Uses Ramsete controller.
  - Used as a front end to drive speed control, not instead of speed control.
- **Inputs**
  - Trajectory sample for the time offset.
  - Current Odometry
- **Outputs**
  - Desired chassis speeds. These need to be “normalized” and converted to wheel speed demands.
- **Starting Trajectory Execution**
  - Something initiates the execution of a trajectory. This could be an autonomous action or a joystick button pushed by a driver. When initiated, the Odometry data is reset and the relative trajectory time is set to zero.

Controlling Trajectory Execution - Procedure

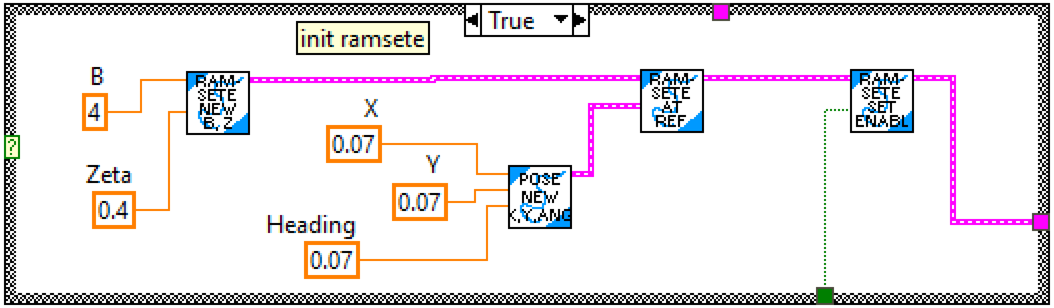
- **Each execution cycle until the trajectory time has elapsed, the robot control system:**
  - Gets the current elapsed time of the trajectory
  - Updates the odometry to calculate a current robot position.
  - Obtains the trajectory sample for the current elapsed trajectory time. This contains the desired robot position, desired linear and rotation robot velocities.
  - For differential robot’s this data is fed to a ramsete control routine which calculates the desired robot Chassis Speeds. Wheel speeds are calculated from Chassis Speeds. These are normalized (to prevent any speed from being larger than the maximum allowed speed.)
  - The normalized Wheel Speed demands are sent to the drive logic, which controls drive wheel speed. This should be done the same as wheel speed is controlled when not executing a trajectory.
- **Notes**
  - Make certain units are converted as needed.
  - Write Odometry, Trajectory sample, Trajectory error, and speed demands to network tables for diagnosis and tracking.

Sample – Read Trajectory File



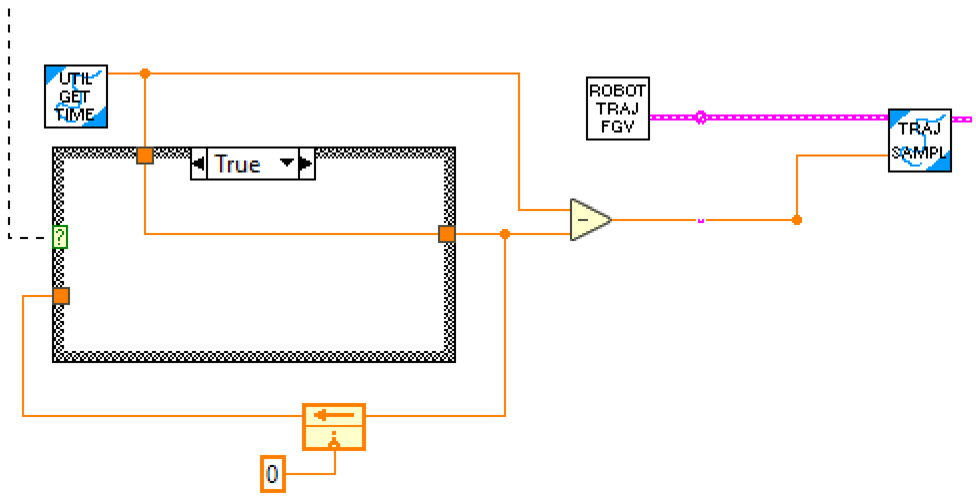
- Read trajectory file. Done once, when robot powered on.
- Populate memory data structure
- Report total time and trajectory states for reporting to dashboard via network tables.

Sample – Initialization Trajectory Execution



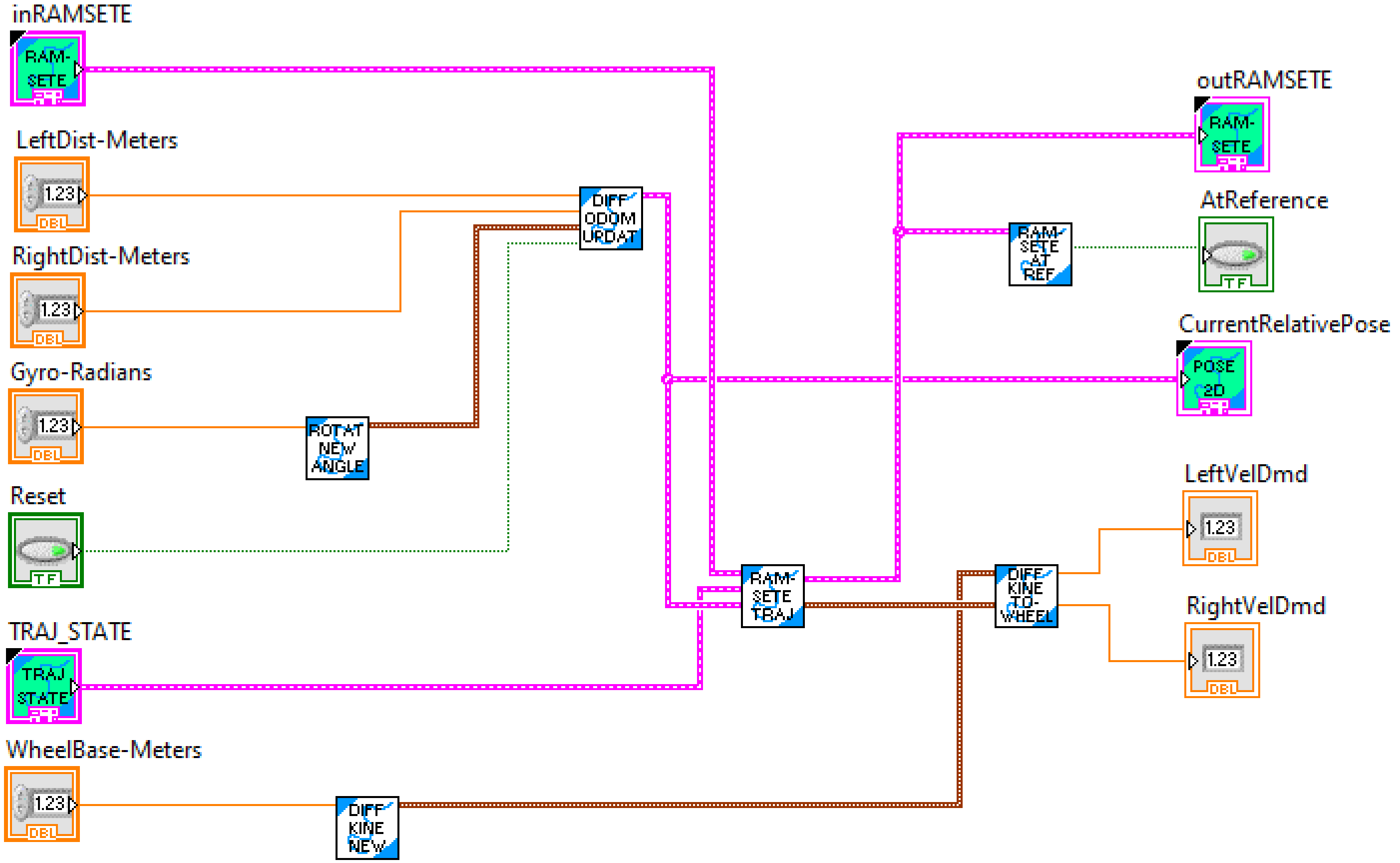
- Done once either at robot power up or before starting trajectory.
- Set Ramsete tuning
- Set allowed error to report “on target”.
- Enable ramsete control. During debugging this could be controlled from dashboard input.

Sample – Get trajectory sample



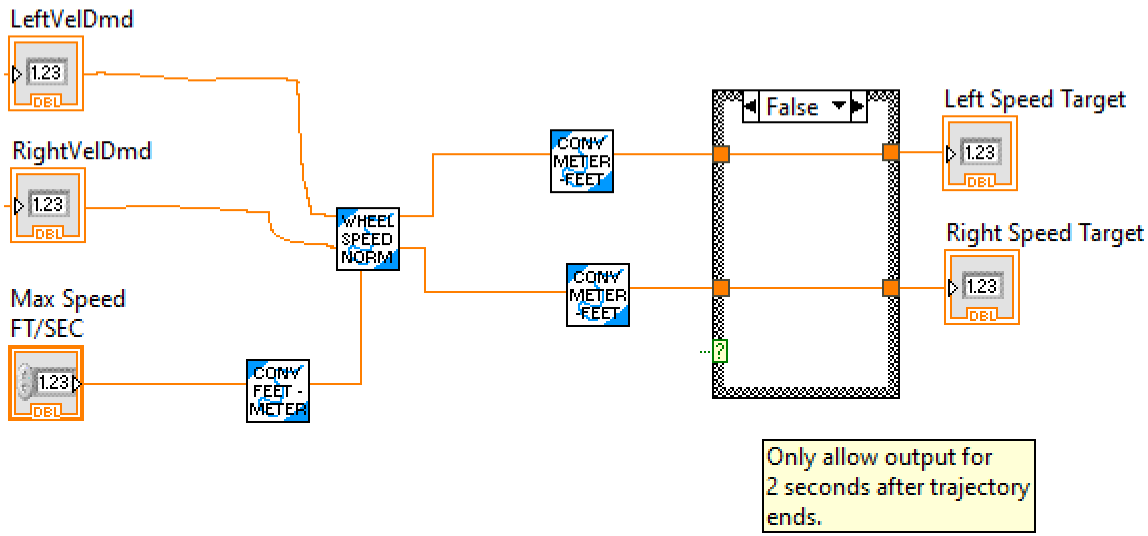
- Get time and calculate time offset since start of trajectory execution
- Get trajectory sample for the current time offset

Sample – Calc desired wheel speeds



- Update robot position (Odometry) using encoder distances, and gyro reading
- Using trajectory state and current position execute ramsete to calc desired speed. Then convert chassis speed to wheel speeds

Sample – Normalize wheel speeds



- Normalize wheel speeds. (Ensure speed doesn't exceed maximum.)
- Convert to units used by speed controller.
- If we are X seconds past trajectory execution force wheel demands to 0.

Final demo and questions

Additional Information

- Kinematics / Odometry:
- https://docs.wpilib.org/en/latest/docs/software/kinematics-and-odometry/index.html

- Trajectory Tutorial:
- https://docs.wpilib.org/en/latest/docs/software/examples-tutorials/trajectory-tutorial/index.html

- LabVIEW Trajectory Library and Samples:
- https://www.chiefdelphi.com/t/labview-trajectory-library-wpilib-port-v1-3/383263

- Team 254 Motion Planning Presentation
- https://www.youtube.com/watch?v=8319J1BEHwM