

## RIT VEXU Core API

Generated by Doxygen 1.13.2

---

<b>1 Core</b>	<b>1</b>
1.1 Getting Started	2
1.2 Features	2
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 Class Documentation</b>	<b>8</b>
4.1 Async Class Reference	8
4.1.1 Detailed Description	9
4.2 AutoChooser Class Reference	9
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	9
4.2.3 Member Function Documentation	10
4.2.4 Member Data Documentation	10
4.3 BasicSolenoidSet Class Reference	10
4.3.1 Detailed Description	11
4.3.2 Constructor & Destructor Documentation	11
4.3.3 Member Function Documentation	11
4.4 BasicSpinCommand Class Reference	11
4.4.1 Detailed Description	12
4.4.2 Constructor & Destructor Documentation	12
4.4.3 Member Function Documentation	12
4.5 BasicStopCommand Class Reference	13
4.5.1 Detailed Description	13
4.5.2 Constructor & Destructor Documentation	13
4.5.3 Member Function Documentation	13
4.6 Branch Class Reference	14
4.6.1 Detailed Description	14
4.7 screen::ButtonWidget Class Reference	14
4.7.1 Detailed Description	14
4.7.2 Constructor & Destructor Documentation	14
4.7.3 Member Function Documentation	15
4.8 CommandController Class Reference	16
4.8.1 Detailed Description	16
4.8.2 Constructor & Destructor Documentation	16
4.8.3 Member Function Documentation	16
4.9 Condition Class Reference	18
4.9.1 Detailed Description	19
4.10 CustomEncoder Class Reference	19

4.10.1 Detailed Description	19
4.10.2 Constructor & Destructor Documentation	19
4.10.3 Member Function Documentation	19
4.11 DelayCommand Class Reference	21
4.11.1 Detailed Description	21
4.11.2 Constructor & Destructor Documentation	21
4.11.3 Member Function Documentation	22
4.12 DriveForwardCommand Class Reference	22
4.12.1 Detailed Description	22
4.12.2 Constructor & Destructor Documentation	22
4.12.3 Member Function Documentation	23
4.13 DriveStopCommand Class Reference	23
4.13.1 Detailed Description	23
4.13.2 Constructor & Destructor Documentation	23
4.13.3 Member Function Documentation	24
4.14 DriveToPointCommand Class Reference	24
4.14.1 Detailed Description	24
4.14.2 Constructor & Destructor Documentation	24
4.14.3 Member Function Documentation	25
4.15 AutoChooser::entry_t Struct Reference	26
4.15.1 Detailed Description	26
4.15.2 Member Data Documentation	26
4.16 ExponentialMovingAverage Class Reference	26
4.16.1 Detailed Description	27
4.16.2 Constructor & Destructor Documentation	27
4.16.3 Member Function Documentation	27
4.17 Feedback Class Reference	28
4.17.1 Detailed Description	29
4.17.2 Member Function Documentation	29
4.18 FeedForward Class Reference	30
4.18.1 Detailed Description	31
4.18.2 Constructor & Destructor Documentation	31
4.18.3 Member Function Documentation	31
4.19 FeedForward::ff_config_t Struct Reference	32
4.19.1 Detailed Description	32
4.19.2 Member Data Documentation	32
4.20 Filter Class Reference	33
4.20.1 Detailed Description	33
4.21 Flywheel Class Reference	33
4.21.1 Detailed Description	34
4.21.2 Constructor & Destructor Documentation	34
4.21.3 Member Function Documentation	35

4.21.4 Friends And Related Symbol Documentation . . . . .	37
4.22 FlywheelStopCommand Class Reference . . . . .	37
4.22.1 Detailed Description . . . . .	37
4.22.2 Constructor & Destructor Documentation . . . . .	37
4.22.3 Member Function Documentation . . . . .	38
4.23 FlywheelStopMotorsCommand Class Reference . . . . .	38
4.23.1 Detailed Description . . . . .	38
4.23.2 Constructor & Destructor Documentation . . . . .	38
4.23.3 Member Function Documentation . . . . .	39
4.24 FlywheelStopNonTasksCommand Class Reference . . . . .	39
4.24.1 Detailed Description . . . . .	39
4.25 FunctionCommand Class Reference . . . . .	39
4.25.1 Detailed Description . . . . .	39
4.26 FunctionCondition Class Reference . . . . .	40
4.26.1 Detailed Description . . . . .	40
4.27 screen::FunctionPage Class Reference . . . . .	40
4.27.1 Detailed Description . . . . .	41
4.27.2 Constructor & Destructor Documentation . . . . .	41
4.27.3 Member Function Documentation . . . . .	41
4.28 GenericAuto Class Reference . . . . .	42
4.28.1 Detailed Description . . . . .	42
4.28.2 Member Function Documentation . . . . .	42
4.29 PurePursuit::hermite_point Struct Reference . . . . .	43
4.29.1 Detailed Description . . . . .	44
4.30 IfTimePassed Class Reference . . . . .	44
4.30.1 Detailed Description . . . . .	44
4.31 InOrder Class Reference . . . . .	44
4.31.1 Detailed Description . . . . .	44
4.32 Lift< T > Class Template Reference . . . . .	45
4.32.1 Detailed Description . . . . .	45
4.32.2 Constructor & Destructor Documentation . . . . .	45
4.32.3 Member Function Documentation . . . . .	46
4.33 Lift< T >::lift_cfg_t Struct Reference . . . . .	49
4.33.1 Detailed Description . . . . .	49
4.34 Logger Class Reference . . . . .	49
4.34.1 Detailed Description . . . . .	50
4.34.2 Constructor & Destructor Documentation . . . . .	50
4.34.3 Member Function Documentation . . . . .	50
4.35 MotionController::m_profile_cfg_t Struct Reference . . . . .	52
4.35.1 Detailed Description . . . . .	52
4.36 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage Class Reference .	53
4.36.1 Detailed Description . . . . .	53

4.36.2 Constructor & Destructor Documentation	53
4.36.3 Member Function Documentation	54
4.37 MecanumDrive Class Reference	54
4.37.1 Detailed Description	55
4.37.2 Constructor & Destructor Documentation	55
4.37.3 Member Function Documentation	55
4.38 MecanumDrive::mecanumdrive_config_t Struct Reference	57
4.38.1 Detailed Description	57
4.39 motion_t Struct Reference	57
4.39.1 Detailed Description	58
4.40 MotionController Class Reference	58
4.40.1 Detailed Description	59
4.40.2 Constructor & Destructor Documentation	59
4.40.3 Member Function Documentation	59
4.41 MovingAverage Class Reference	62
4.41.1 Detailed Description	62
4.41.2 Constructor & Destructor Documentation	62
4.41.3 Member Function Documentation	63
4.42 Odometry3Wheel Class Reference	64
4.42.1 Detailed Description	65
4.42.2 Constructor & Destructor Documentation	65
4.42.3 Member Function Documentation	65
4.43 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	66
4.43.1 Detailed Description	66
4.43.2 Member Data Documentation	66
4.44 OdometryBase Class Reference	67
4.44.1 Detailed Description	68
4.44.2 Constructor & Destructor Documentation	68
4.44.3 Member Function Documentation	68
4.44.4 Member Data Documentation	71
4.45 screen::OdometryPage Class Reference	72
4.45.1 Detailed Description	73
4.45.2 Constructor & Destructor Documentation	73
4.45.3 Member Function Documentation	73
4.46 OdometrySerial Class Reference	74
4.46.1 Detailed Description	75
4.46.2 Constructor & Destructor Documentation	75
4.46.3 Member Function Documentation	75
4.47 OdometryTank Class Reference	78
4.47.1 Detailed Description	79
4.47.2 Constructor & Destructor Documentation	79
4.47.3 Member Function Documentation	81

4.48 OdomSetPosition Class Reference	81
4.48.1 Detailed Description	81
4.48.2 Constructor & Destructor Documentation	81
4.48.3 Member Function Documentation	82
4.49 screen::Page Class Reference	82
4.49.1 Detailed Description	83
4.49.2 Member Function Documentation	83
4.50 Parallel Class Reference	83
4.50.1 Detailed Description	84
4.51 PurePursuit::Path Class Reference	84
4.51.1 Detailed Description	84
4.51.2 Constructor & Destructor Documentation	84
4.51.3 Member Function Documentation	84
4.52 PID Class Reference	85
4.52.1 Detailed Description	86
4.52.2 Member Enumeration Documentation	86
4.52.3 Constructor & Destructor Documentation	86
4.52.4 Member Function Documentation	87
4.52.5 Member Data Documentation	89
4.53 PID::pid_config_t Struct Reference	90
4.53.1 Detailed Description	90
4.53.2 Member Data Documentation	90
4.54 screen::PIDPage Class Reference	91
4.54.1 Detailed Description	91
4.54.2 Constructor & Destructor Documentation	91
4.54.3 Member Function Documentation	92
4.55 Pose2d Class Reference	92
4.55.1 Detailed Description	93
4.55.2 Constructor & Destructor Documentation	93
4.55.3 Member Function Documentation	95
4.55.4 Friends And Related Symbol Documentation	99
4.56 PurePursuitCommand Class Reference	99
4.56.1 Detailed Description	99
4.56.2 Constructor & Destructor Documentation	99
4.56.3 Member Function Documentation	100
4.57 Rect Struct Reference	100
4.57.1 Detailed Description	100
4.58 robot_specs_t Struct Reference	100
4.58.1 Detailed Description	101
4.58.2 Member Data Documentation	101
4.59 Rotation2d Class Reference	101
4.59.1 Detailed Description	102

4.59.2 Constructor & Destructor Documentation	102
4.59.3 Member Function Documentation	103
4.59.4 Friends And Related Symbol Documentation	108
4.60 screen::ScreenData Struct Reference	109
4.60.1 Detailed Description	109
4.61 Serializer Class Reference	109
4.61.1 Detailed Description	109
4.61.2 Constructor & Destructor Documentation	110
4.61.3 Member Function Documentation	111
4.62 screen::SliderWidget Class Reference	113
4.62.1 Detailed Description	114
4.62.2 Constructor & Destructor Documentation	114
4.62.3 Member Function Documentation	114
4.63 SpinRPMCommand Class Reference	115
4.63.1 Detailed Description	115
4.63.2 Constructor & Destructor Documentation	115
4.63.3 Member Function Documentation	116
4.64 PurePursuit::spline Struct Reference	116
4.64.1 Detailed Description	116
4.65 StateMachine< System, IDType, Message, delay_ms, do_log >::State Struct Reference	116
4.65.1 Detailed Description	116
4.66 StateMachine< System, IDType, Message, delay_ms, do_log > Class Template Reference	117
4.66.1 Detailed Description	117
4.66.2 Constructor & Destructor Documentation	118
4.66.3 Member Function Documentation	118
4.67 screen::StatsPage Class Reference	119
4.67.1 Detailed Description	119
4.67.2 Constructor & Destructor Documentation	119
4.67.3 Member Function Documentation	119
4.68 TakeBackHalf Class Reference	120
4.68.1 Detailed Description	120
4.68.2 Member Function Documentation	121
4.69 TankDrive Class Reference	122
4.69.1 Detailed Description	123
4.69.2 Member Enumeration Documentation	123
4.69.3 Constructor & Destructor Documentation	123
4.69.4 Member Function Documentation	124
4.70 tracking_wheel_cfg_t Struct Reference	132
4.70.1 Detailed Description	132
4.70.2 Member Data Documentation	133
4.71 Transform2d Class Reference	133
4.71.1 Detailed Description	134

4.71.2 Constructor & Destructor Documentation . . . . .	134
4.71.3 Member Function Documentation . . . . .	137
4.71.4 Friends And Related Symbol Documentation . . . . .	139
4.72 Translation2d Class Reference . . . . .	139
4.72.1 Detailed Description . . . . .	140
4.72.2 Constructor & Destructor Documentation . . . . .	140
4.72.3 Member Function Documentation . . . . .	141
4.72.4 Friends And Related Symbol Documentation . . . . .	146
4.73 TrapezoidProfile Class Reference . . . . .	146
4.73.1 Detailed Description . . . . .	147
4.73.2 Constructor & Destructor Documentation . . . . .	147
4.73.3 Member Function Documentation . . . . .	147
4.74 TurnDegreesCommand Class Reference . . . . .	149
4.74.1 Detailed Description . . . . .	149
4.74.2 Constructor & Destructor Documentation . . . . .	149
4.74.3 Member Function Documentation . . . . .	150
4.75 TurnToHeadingCommand Class Reference . . . . .	150
4.75.1 Detailed Description . . . . .	150
4.75.2 Constructor & Destructor Documentation . . . . .	150
4.75.3 Member Function Documentation . . . . .	151
4.76 Twist2d Class Reference . . . . .	151
4.76.1 Detailed Description . . . . .	152
4.76.2 Constructor & Destructor Documentation . . . . .	152
4.76.3 Member Function Documentation . . . . .	153
4.76.4 Friends And Related Symbol Documentation . . . . .	154
4.77 WaitUntilCondition Class Reference . . . . .	154
4.77.1 Detailed Description . . . . .	155
4.78 WaitUntilUpToSpeedCommand Class Reference . . . . .	155
4.78.1 Detailed Description . . . . .	155
4.78.2 Constructor & Destructor Documentation . . . . .	155
4.78.3 Member Function Documentation . . . . .	155
<b>Index</b>	<b>157</b>

## 1 Core

This is the host repository for the custom VEX libraries used by the RIT VEXU team

Automatically updated documentation is available at [here](#). There is also a downloadable [reference manual](#).



## 1.1 Getting Started

If you just want to start a project with Core, make a fork of the [Fork Template](#) and follow its instructions.

To setup core for an existing project:

1. Create a new vex project (using the VSCode extension or other methods)
2. Initialize a git repository for the project
3. Execute 

```
git subtree add --prefix=core https://github.com/RIT-VEX-U/↵ Core.git main
```
4. Update the vex Makefile (or any other build system) to know about the core files (`core/src` for source files, `core/include` for headers) (See [here](#) for an example)
5. Enable [Eigen](#) (Latest supported version is 3.4.0):
  - `mkdir vendor`
  - `git submodule add https://gitlab.com/libeigen/eigen.git vendor/eigen`
  - `cd vendor/eigen`
  - `git checkout 3.4.0`
  - Add the following to the makefile to give Core access to the library: `INC += -Ivendor/eigen` (See [here](#) for an example)

If you only wish to use a single version of Core, you can simply clone `core/` into your project and add the core source and header files to your makefile.

## 1.2 Features

Here is the current feature list this repo provides:

Subsystems (See [Wiki/Subsystems](#)):

- Tank drivetrain (user control / autonomous)
- Mecanum drivetrain (user control / autonomous)
- Odometry
  - Tank (Differential)
  - [N-Pod](#)
- [Flywheel](#)
- [Lift](#)
- Custom encoders

Utilities (See [Wiki/Utilites](#)):

- [PID](#) controller
- [FeedForward](#) controller
- Trapezoidal motion profile controller
- Pure Pursuit
- Generic auto program builder
- Auto program UI selector
- Mathematical classes (Vector2D, Moving Average)

## 2 Hierarchical Index

### 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

<b>Async</b>	<b>8</b>
<b>BasicSolenoidSet</b>	<b>10</b>
<b>BasicSpinCommand</b>	<b>11</b>
<b>BasicStopCommand</b>	<b>13</b>
<b>Branch</b>	<b>14</b>
<b>screen::ButtonWidget</b>	<b>14</b>
<b>CommandController</b>	<b>16</b>
<b>Condition</b>	<b>18</b>
<b>FunctionCondition</b>	<b>40</b>
<b>IfTimePassed</b>	<b>44</b>
<b>CustomEncoder</b>	<b>19</b>
<b>DelayCommand</b>	<b>21</b>
<b>DriveForwardCommand</b>	<b>22</b>
<b>DriveStopCommand</b>	<b>23</b>
<b>DriveToPointCommand</b>	<b>24</b>
<b>AutoChooser::entry_t</b>	<b>26</b>
<b>Feedback</b>	<b>28</b>
<b>MotionController</b>	<b>58</b>
<b>PID</b>	<b>85</b>
<b>TakeBackHalf</b>	<b>120</b>
<b>FeedForward</b>	<b>30</b>
<b>FeedForward::ff_config_t</b>	<b>32</b>
<b>Filter</b>	<b>33</b>
<b>ExponentialMovingAverage</b>	<b>26</b>
<b>MovingAverage</b>	<b>62</b>
<b>Flywheel</b>	<b>33</b>
<b>FlywheelStopCommand</b>	<b>37</b>
<b>FlywheelStopMotorsCommand</b>	<b>38</b>

FlywheelStopNonTasksCommand	39
FunctionCommand	39
GenericAuto	42
PurePursuit::hermite_point	43
InOrder	44
Lift< T >	45
Lift< T >::lift_cfg_t	49
Logger	49
MotionController::m_profile_cfg_t	52
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage	53
MecanumDrive	54
MecanumDrive::mecanumdrive_config_t	57
motion_t	57
Odometry3Wheel::odometry3wheel_cfg_t	66
OdometryBase	67
Odometry3Wheel	64
OdometrySerial	74
OdometryTank	78
OdomSetPosition	81
screen::Page	82
AutoChooser	9
screen::FunctionPage	40
screen::OdometryPage	72
screen::PIDPage	91
screen::StatsPage	119
Parallel	83
PurePursuit::Path	84
PID::pid_config_t	90
Pose2d	92
PurePursuitCommand	99
Rect	100
robot_specs_t	100

Rotation2d	101
screen::ScreenData	109
Serializer	109
screen::SliderWidget	113
SpinRPMCommand	115
PurePursuit::spline	116
StateMachine< System, IDType, Message, delay_ms, do_log >::State	116
StateMachine< System, IDType, Message, delay_ms, do_log >	117
TankDrive	122
tracking_wheel_cfg_t	132
Transform2d	133
Translation2d	139
TrapezoidProfile	146
TurnDegreesCommand	149
TurnToHeadingCommand	150
Twist2d	151
WaitUntilCondition	154
WaitUntilUpToSpeedCommand	155

## 3 Class Index

### 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<b>Async</b>	
<b>Async</b> runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT	8
<b>AutoChooser</b>	9
<b>BasicSolenoidSet</b>	10
<b>BasicSpinCommand</b>	11
<b>BasicStopCommand</b>	13
<b>Branch</b>	
<b>Branch</b> chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE	14

<a href="#">screen::ButtonWidget</a>	
Widget that does something when you tap it. The function is only called once when you first tap it	14
<a href="#">CommandController</a>	16
<a href="#">Condition</a>	18
<a href="#">CustomEncoder</a>	19
<a href="#">DelayCommand</a>	21
<a href="#">DriveForwardCommand</a>	22
<a href="#">DriveStopCommand</a>	23
<a href="#">DriveToPointCommand</a>	24
<a href="#">AutoChooser::entry_t</a>	26
<a href="#">ExponentialMovingAverage</a>	26
<a href="#">Feedback</a>	28
<a href="#">FeedForward</a>	30
<a href="#">FeedForward::ff_config_t</a>	32
<a href="#">Filter</a>	33
<a href="#">Flywheel</a>	33
<a href="#">FlywheelStopCommand</a>	37
<a href="#">FlywheelStopMotorsCommand</a>	38
<a href="#">FlywheelStopNonTasksCommand</a>	39
<a href="#">FunctionCommand</a>	39
<a href="#">FunctionCondition</a>	
<a href="#">FunctionCondition</a> is a quick and dirty <a href="#">Condition</a> to wrap some expression that should be evaluated at runtime	40
<a href="#">screen::FunctionPage</a>	
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything	40
<a href="#">GenericAuto</a>	42
<a href="#">PurePursuit::hermite_point</a>	43
<a href="#">IfTimePassed</a>	
<a href="#">IfTimePassed</a> tests based on time since the command controller was constructed. Returns true if elapsed time > time_s	44
<a href="#">InOrder</a>	
<a href="#">InOrder</a> runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?	44
<a href="#">Lift&lt; T &gt;</a>	45

Lift< T >::lift_cfg_t	49
Logger	
Class to simplify writing to files	49
MotionController::m_profile_cfg_t	52
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage	
MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty	
MaybeMessage m = Message::EnumField1	53
MecanumDrive	54
MecanumDrive::mecanumdrive_config_t	57
motion_t	57
MotionController	58
MovingAverage	62
Odometry3Wheel	64
Odometry3Wheel::odometry3wheel_cfg_t	66
OdometryBase	67
screen::OdometryPage	
Page that shows odometry position and rotation and a map (if an sd card with the file is on)	72
OdometrySerial	74
OdometryTank	78
OdomSetPosition	81
screen::Page	
Page describes one part of the screen slideshow	82
Parallel	
Parallel runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call on_timeout on all children continue	83
PurePursuit::Path	84
PID	85
PID::pid_config_t	90
screen::PIDPage	
PIDPage provides a way to tune a pid controller on the screen	91
Pose2d	92
PurePursuitCommand	99
Rect	100
robot_specs_t	100
Rotation2d	101

<a href="#">screen::ScreenData</a>	
Holds the data that will be passed to the screen thread you probably shouldnt have to use it	109
<a href="#">Serializer</a>	
Serializes Arbitrary data to a file on the SD Card	109
<a href="#">screen::SliderWidget</a>	
Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread	113
<a href="#">SpinRPMCommand</a>	115
<a href="#">PurePursuit::spline</a>	116
<a href="#">StateMachine&lt; System, IDType, Message, delay_ms, do_log &gt;::State</a>	116
<a href="#">StateMachine&lt; System, IDType, Message, delay_ms, do_log &gt;</a>	
State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message	117
<a href="#">screen::StatsPage</a>	
Draws motor stats and battery stats to the screen	119
<a href="#">TakeBackHalf</a>	
A velocity controller	120
<a href="#">TankDrive</a>	122
<a href="#">tracking_wheel_cfg_t</a>	132
<a href="#">Transform2d</a>	133
<a href="#">Translation2d</a>	139
<a href="#">TrapezoidProfile</a>	146
<a href="#">TurnDegreesCommand</a>	149
<a href="#">TurnToHeadingCommand</a>	150
<a href="#">Twist2d</a>	151
<a href="#">WaitUntilCondition</a>	
Waits until the condition is true	154
<a href="#">WaitUntilUpToSpeedCommand</a>	155

## 4 Class Documentation

### 4.1 Async Class Reference

[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

```
#include <auto_command.h>
```

### 4.1.1 Detailed Description

[Async](#) runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT.

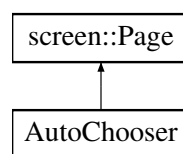
The documentation for this class was generated from the following files:

- `auto_command.h`
- `auto_command.cpp`

## 4.2 AutoChooser Class Reference

```
#include <auto_chooser.h>
```

Inheritance diagram for AutoChooser:



### Classes

- struct [entry\\_t](#)

### Public Member Functions

- [AutoChooser](#) (std::vector< std::string > paths, size\_t def=0)
- size\_t [get\\_choice](#) ()

### Protected Attributes

- size\_t [choice](#)
- std::vector< [entry\\_t](#) > [list](#)

### 4.2.1 Detailed Description

Autochooser is a utility to make selecting robot autonomous programs easier source: RIT VexU Wiki During a season, we usually code between 4 and 6 autonomous programs. Most teams will change their entire robot program as a way of choosing autonomi but this may cause issues if you have an emergency patch to upload during a competition. This class was built as a way of using the robot screen to list autonomous programs, and the touchscreen to select them.

### 4.2.2 Constructor & Destructor Documentation

#### AutoChooser()

```
AutoChooser::AutoChooser (
    std::vector< std::string > paths,
    size_t def = 0)
```

Initialize the auto-chooser. This class places a choice menu on the brain screen, so the driver can choose which autonomous to run.



**Parameters**

<i>brain</i>	the brain on which to draw the selection boxes
--------------	--

**4.2.3 Member Function Documentation****get\_choice()**

```
size_t AutoChooser::get_choice ()
```

Get the currently selected auto choice

**Returns**

the identifier to the auto path

Return the selected autonomous

**4.2.4 Member Data Documentation****choice**

```
size_t AutoChooser::choice [protected]
```

the current choice of auto

**list**

```
std::vector<entry_t> AutoChooser::list [protected]
```

< a list of all possible auto choices

The documentation for this class was generated from the following files:

- auto\_chooser.h
- auto\_chooser.cpp

**4.3 BasicSolenoidSet Class Reference**

```
#include <basic_command.h>
```

**Public Member Functions**

- [BasicSolenoidSet](#) (vex::pneumatics &solenoid, bool setting)  
*Construct a new [BasicSolenoidSet](#) Command.*
- bool [run](#) () override  
*Runs the [BasicSolenoidSet](#) Overrides run command from AutoCommand.*

### 4.3.1 Detailed Description

AutoCommand wrapper class for [BasicSolenoidSet](#) Using the Vex hardware functions

### 4.3.2 Constructor & Destructor Documentation

#### BasicSolenoidSet()

```
BasicSolenoidSet::BasicSolenoidSet (
    vex::pneumatics & solenoid,
    bool setting)
```

Construct a new [BasicSolenoidSet](#) Command.

#### Parameters

<i>solenoid</i>	Solenoid being set
<i>setting</i>	Setting of the solenoid in boolean (true,false)

### 4.3.3 Member Function Documentation

#### run()

```
bool BasicSolenoidSet::run () [override]
```

Runs the [BasicSolenoidSet](#) Overrides run command from AutoCommand.

#### Returns

True Command runs once

The documentation for this class was generated from the following files:

- basic\_command.h
- basic\_command.cpp

## 4.4 BasicSpinCommand Class Reference

```
#include <basic_command.h>
```

### Public Member Functions

- [BasicSpinCommand](#) (vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting, double power)  
Construct a new [BasicSpinCommand](#).
- bool [run](#) () override  
Runs the [BasicSpinCommand](#) Overrides run from Auto Command.

#### 4.4.1 Detailed Description

AutoCommand wrapper class for [BasicSpinCommand](#) using the vex hardware functions

#### 4.4.2 Constructor & Destructor Documentation

##### BasicSpinCommand()

```
BasicSpinCommand::BasicSpinCommand (
    vex::motor & motor,
    vex::directionType dir,
    BasicSpinCommand::type setting,
    double power)
```

Construct a new [BasicSpinCommand](#).

a BasicMotorSpin Command

##### Parameters

<i>motor</i>	Motor to spin
<i>direc</i>	Direction of motor spin
<i>setting</i>	Power setting in volts,percentage,velocity
<i>power</i>	Value of desired power
<i>motor</i>	Motor port to spin
<i>dir</i>	Direction for spinning
<i>setting</i>	Power setting in volts,percentage,velocity
<i>power</i>	Value of desired power

#### 4.4.3 Member Function Documentation

##### run()

```
bool BasicSpinCommand::run () [override]
```

Runs the [BasicSpinCommand](#) Overrides run from Auto Command.

Run the [BasicSpinCommand](#) Overrides run from Auto Command.

##### Returns

True [Async](#) running command  
True Command runs once

The documentation for this class was generated from the following files:

- basic\_command.h
- basic\_command.cpp

## 4.5 BasicStopCommand Class Reference

```
#include <basic_command.h>
```

### Public Member Functions

- [BasicStopCommand](#) (vex::motor &motor, vex::brakeType setting)  
*Construct a new BasicMotorStop Command.*
- bool [run](#) () override  
*Runs the BasicMotorStop Command Overrides run command from AutoCommand.*

#### 4.5.1 Detailed Description

AutoCommand wrapper class for [BasicStopCommand](#) Using the Vex hardware functions

#### 4.5.2 Constructor & Destructor Documentation

##### BasicStopCommand()

```
BasicStopCommand::BasicStopCommand (
    vex::motor & motor,
    vex::brakeType setting)
```

Construct a new BasicMotorStop Command.

Construct a BasicMotorStop Command.

##### Parameters

<i>motor</i>	The motor to stop
<i>setting</i>	The brake setting for the motor
<i>motor</i>	Motor to stop
<i>setting</i>	Braketype setting brake,coast,hold

#### 4.5.3 Member Function Documentation

##### run()

```
bool BasicStopCommand::run () [override]
```

Runs the BasicMotorStop Command Overrides run command from AutoCommand.

Runs the BasicMotorStop command Ovverides run command from AutoCommand.

##### Returns

True Command runs once

The documentation for this class was generated from the following files:

- basic\_command.h
- basic\_command.cpp

## 4.6 Branch Class Reference

[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO\_CHOICE; any choice that is out of bounds set to NO\_CHOICE.

```
#include <auto_command.h>
```

### 4.6.1 Detailed Description

[Branch](#) chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO\_CHOICE; any choice that is out of bounds set to NO\_CHOICE.

The documentation for this class was generated from the following files:

- `auto_command.h`
- `auto_command.cpp`

## 4.7 screen::ButtonWidget Class Reference

Widget that does something when you tap it. The function is only called once when you first tap it.

```
#include <screen.h>
```

### Public Member Functions

- [ButtonWidget](#) (std::function< void(void)> onpress, [Rect](#) rect, std::string name)  
*Create a Button widget.*
- [ButtonWidget](#) (void(\*onpress)(), [Rect](#) rect, std::string name)  
*Create a Button widget.*
- bool [update](#) (bool was\_pressed, int x, int y)  
*responds to user input*
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)  
*draws the button to the screen*

### 4.7.1 Detailed Description

Widget that does something when you tap it. The function is only called once when you first tap it.

### 4.7.2 Constructor & Destructor Documentation

#### ButtonWidget() [1/2]

```
screen::ButtonWidget::ButtonWidget (
    std::function< void(void)> onpress,
    Rect rect,
    std::string name) [inline]
```

Create a Button widget.

## Parameters

<i>onpress</i>	the function to be called when the button is tapped
<i>rect</i>	the area the button should take up on the screen
<i>name</i>	the label put on the button

**ButtonWidget()** [2/2]

```
screen::ButtonWidget::ButtonWidget (
    void(* onpress )(),
    Rect rect,
    std::string name) [inline]
```

Create a Button widget.

## Parameters

<i>onpress</i>	the function to be called when the button is tapped
<i>rect</i>	the area the button should take up on the screen
<i>name</i>	the label put on the button

**4.7.3 Member Function Documentation****update()**

```
bool screen::ButtonWidget::update (
    bool was_pressed,
    int x,
    int y)
```

responds to user input

## Parameters

<i>was_pressed</i>	if the screen is pressed
<i>x</i>	x position if the screen was pressed
<i>y</i>	y position if the screen was pressed

## Returns

true if the button was pressed

The documentation for this class was generated from the following files:

- screen.h
- screen.cpp

## 4.8 CommandController Class Reference

```
#include <command_controller.h>
```

### Public Member Functions

- **CommandController ()**  
Create an empty [CommandController](#). Add Command with [CommandController::add\(\)](#)
- **CommandController (std::initializer\_list< AutoCommand \* > cmds)**  
Create a [CommandController](#) with commands pre added. More can be added with [CommandController::add\(\)](#)
- void **add** (std::vector< AutoCommand \* > cmds)
- void **add** (AutoCommand \*cmd, double timeout\_seconds=10.0)
- void **add** (std::vector< AutoCommand \* > cmds, double timeout\_sec)
- void **add\_delay** (int ms)
- void **add\_cancel\_func** (std::function< bool(void)> true\_if\_cancel)  
*add\_cancel\_func specifies that when this func evaluates to true, to cancel the command controller*
- void **run** ()
- bool **last\_command\_timed\_out** ()

### 4.8.1 Detailed Description

File: [command\\_controller.h](#) Desc: A [CommandController](#) manages the AutoCommands that make up an autonomous route. The AutoCommands are kept in a queue and get executed and removed from the queue in FIFO order.

### 4.8.2 Constructor & Destructor Documentation

#### CommandController()

```
CommandController::CommandController (
    std::initializer_list< AutoCommand * > cmds) [inline]
```

Create a [CommandController](#) with commands pre added. More can be added with [CommandController::add\(\)](#)

#### Parameters

<i>cmds</i>	
-------------	--

### 4.8.3 Member Function Documentation

#### add() [1/3]

```
void CommandController::add (
    AutoCommand * cmd,
    double timeout_seconds = 10.0)
```

File: [command\\_controller.cpp](#) Desc: A [CommandController](#) manages the AutoCommands that make up an autonomous route. The AutoCommands are kept in a queue and get executed and removed from the queue in FIFO order. Adds a command to the queue

## Parameters

<i>cmd</i>	the AutoCommand we want to add to our list
<i>timeout_seconds</i>	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout

**add()** [2/3]

```
void CommandController::add (
    std::vector< AutoCommand * > cmds)
```

Adds a command to the queue

## Parameters

<i>cmd</i>	the AutoCommand we want to add to our list
<i>timeout_seconds</i>	the number of seconds we will let the command run for. If it exceeds this, we cancel it and run on_timeout. if it is <= 0 no time out will be applied

Add multiple commands to the queue. No timeout here.

## Parameters

<i>cmds</i>	the AutoCommands we want to add to our list
-------------	---

**add()** [3/3]

```
void CommandController::add (
    std::vector< AutoCommand * > cmds,
    double timeout_sec)
```

Add multiple commands to the queue. No timeout here.

## Parameters

<i>cmds</i>	the AutoCommands we want to add to our list Add multiple commands to the queue. No timeout here.
<i>cmds</i>	the AutoCommands we want to add to our list
<i>timeout_sec</i>	timeout in seconds to apply to all commands if they are still the default

Add multiple commands to the queue. No timeout here.

## Parameters

<i>cmds</i>	the AutoCommands we want to add to our list
<i>timeout</i>	timeout in seconds to apply to all commands if they are still the default

**add\_cancel\_func()**

```
void CommandController::add_cancel_func (
    std::function< bool(void)> true_if_cancel)
```

add\_cancel\_func specifies that when this func evaluates to true, to cancel the command controller



**Parameters**

<code>true_if_cancel</code>	a function that returns true when we want to cancel the command controller
-----------------------------	--

**add\_delay()**

```
void CommandController::add_delay (
    int ms)
```

Adds a command that will delay progression of the queue

**Parameters**

<code>ms</code>	- number of milliseconds to wait before continuing execution of autonomous
-----------------	--

**last\_command\_timed\_out()**

```
bool CommandController::last_command_timed_out ()
```

`last_command_timed_out` tells how the last command ended Use this if you want to make decisions based on the end of the last command

**Returns**

true if the last command timed out. false if it finished regularly

**run()**

```
void CommandController::run ()
```

Begin execution of the queue Execute and remove commands in FIFO order

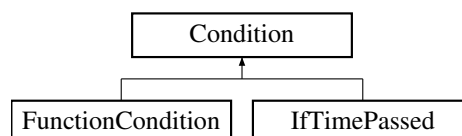
The documentation for this class was generated from the following files:

- `command_controller.h`
- `command_controller.cpp`

**4.9 Condition Class Reference**

```
#include <auto_command.h>
```

Inheritance diagram for Condition:



### 4.9.1 Detailed Description

File: [auto\\_command.h](#) Desc: Interface for module-specific commands A [Condition](#) is a function that returns true or false `is_even` is a predicate that would return true if a number is even For our purposes, a [Condition](#) is a choice to be made at runtime `drive_sys.reached_point(10, 30)` is a predicate `time.has_elapsed(10, vex::seconds)` is a predicate extend this class for different choices you wish to make

The documentation for this class was generated from the following files:

- `auto_command.h`
- `auto_command.cpp`

## 4.10 CustomEncoder Class Reference

```
#include <custom_encoder.h>
```

### Public Member Functions

- [CustomEncoder](#) (`vex::triport::port &port`, `double ticks_per_rev`)
- `void setRotation` (`double val`, `vex::rotationUnits units`)
- `void setPosition` (`double val`, `vex::rotationUnits units`)
- `double rotation` (`vex::rotationUnits units`)
- `double position` (`vex::rotationUnits units`)
- `double velocity` (`vex::velocityUnits units`)

### 4.10.1 Detailed Description

A wrapper class for the vex encoder that allows the use of 3rd party encoders with different tick-per-revolution values.

### 4.10.2 Constructor & Destructor Documentation

#### CustomEncoder()

```
CustomEncoder::CustomEncoder (
    vex::triport::port & port,
    double ticks_per_rev)
```

Construct an encoder with a custom number of ticks

#### Parameters

<i>port</i>	the triport port on the brain the encoder is plugged into
<i>ticks_per_rev</i>	the number of ticks the encoder will report for one revolution

### 4.10.3 Member Function Documentation

#### position()

```
double CustomEncoder::position (
    vex::rotationUnits units)
```

get the position that the encoder is at

**Parameters**

<i>units</i>	the unit we want the return value to be in
--------------	--

**Returns**

the position of the encoder in the units specified

**rotation()**

```
double CustomEncoder::rotation (  
    vex::rotationUnits units)
```

get the rotation that the encoder is at

**Parameters**

<i>units</i>	the unit we want the return value to be in
--------------	--

**Returns**

the rotation of the encoder in the units specified

**setPosition()**

```
void CustomEncoder::setPosition (  
    double val,  
    vex::rotationUnits units)
```

sets the stored position of the encoder. Any further movements will be from this value

**Parameters**

<i>val</i>	the numerical value of the position we are setting to
<i>units</i>	the unit of <i>val</i>

**setRotation()**

```
void CustomEncoder::setRotation (  
    double val,  
    vex::rotationUnits units)
```

sets the stored rotation of the encoder. Any further movements will be from this value

**Parameters**

<i>val</i>	the numerical value of the angle we are setting to
<i>units</i>	the unit of <i>val</i>

**velocity()**

```
double CustomEncoder::velocity (  
    vex::velocityUnits units)
```

get the velocity that the encoder is moving at

**Parameters**

<i>units</i>	the unit we want the return value to be in
--------------	--

**Returns**

the velocity of the encoder in the units specified

The documentation for this class was generated from the following files:

- custom\_encoder.h
- custom\_encoder.cpp

## 4.11 DelayCommand Class Reference

```
#include <delay_command.h>
```

**Public Member Functions**

- [DelayCommand](#) (int ms)
- bool [run](#) () override

### 4.11.1 Detailed Description

File: [delay\\_command.h](#) Desc: A [DelayCommand](#) will make the robot wait the set amount of milliseconds before continuing execution of the autonomous route

### 4.11.2 Constructor & Destructor Documentation

**DelayCommand()**

```
DelayCommand::DelayCommand (  
    int ms) [inline]
```

Construct a delay command

**Parameters**

<i>ms</i>	the number of milliseconds to delay for
-----------	---

### 4.11.3 Member Function Documentation

#### run()

```
bool DelayCommand::run () [inline], [override]
```

Delays for the amount of milliseconds stored in the command Overrides run from AutoCommand

#### Returns

true when complete

The documentation for this class was generated from the following file:

- `delay_command.h`

## 4.12 DriveForwardCommand Class Reference

```
#include <drive_commands.h>
```

### Public Member Functions

- [DriveForwardCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double inches, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

#### 4.12.1 Detailed Description

AutoCommand wrapper class for the `drive_forward` function in the [TankDrive](#) class

#### 4.12.2 Constructor & Destructor Documentation

##### DriveForwardCommand()

```
DriveForwardCommand::DriveForwardCommand (  
    TankDrive & drive_sys,  
    Feedback & feedback,  
    double inches,  
    directionType dir,  
    double max_speed = 1,  
    double end_speed = 0)
```

File: [drive\\_commands.h](#) Desc: Holds all the AutoCommand subclasses that wrap (currently) [TankDrive](#) functions

Currently includes:

- `drive_forward`
- `turn_degrees`
- `drive_to_point`
- `turn_to_heading`
- `stop`

Also holds AutoCommand subclasses that wrap [OdometryBase](#) functions

Currently includes:

- `set_position` Construct a DriveForward Command

## Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>inches</i>	how far forward to drive
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

## 4.12.3 Member Function Documentation

**on\_timeout()**

```
void DriveForwardCommand::on_timeout () [override]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

**run()**

```
bool DriveForwardCommand::run () [override]
```

Run drive\_forward Overrides run from AutoCommand

## Returns

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- drive\_commands.h
- drive\_commands.cpp

## 4.13 DriveStopCommand Class Reference

```
#include <drive_commands.h>
```

## Public Member Functions

- [DriveStopCommand](#) ([TankDrive](#) &drive\_sys)
- bool [run](#) () override

## 4.13.1 Detailed Description

AutoCommand wrapper class for the stop() function in the [TankDrive](#) class

## 4.13.2 Constructor &amp; Destructor Documentation

**DriveStopCommand()**

```
DriveStopCommand::DriveStopCommand (
    TankDrive & drive_sys)
```

Construct a DriveStop Command

**Parameters**

<code>drive_sys</code>	the drive system we are commanding
------------------------	------------------------------------

**4.13.3 Member Function Documentation****run()**

```
bool DriveStopCommand::run () [override]
```

Stop the drive system Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

Stop the drive train Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- `drive_commands.h`
- `drive_commands.cpp`

**4.14 DriveToPointCommand Class Reference**

```
#include <drive_commands.h>
```

**Public Member Functions**

- [DriveToPointCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double x, double y, directionType dir, double max\_speed=1, double end\_speed=0)
- [DriveToPointCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, [Translation2d](#) translation, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override

**4.14.1 Detailed Description**

AutoCommand wrapper class for the `drive_to_point` function in the [TankDrive](#) class

**4.14.2 Constructor & Destructor Documentation****DriveToPointCommand() [1/2]**

```
DriveToPointCommand::DriveToPointCommand (  
    TankDrive & drive_sys,  
    Feedback & feedback,  
    double x,  
    double y,  
    directionType dir,  
    double max_speed = 1,  
    double end_speed = 0)
```

Construct a DriveForward Command

**Parameters**

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>x</i>	where to drive in the x dimension
<i>y</i>	where to drive in the y dimension
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

**DriveToPointCommand() [2/2]**

```
DriveToPointCommand::DriveToPointCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    Translation2d translation,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0)
```

Construct a DriveForward Command

**Parameters**

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>translation</i>	the point to drive to
<i>dir</i>	the direction to drive
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

**4.14.3 Member Function Documentation****run()**

```
bool DriveToPointCommand::run () [override]
```

Run drive\_to\_point Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- drive\_commands.h
- drive\_commands.cpp



## 4.15 AutoChooser::entry\_t Struct Reference

```
#include <auto_chooser.h>
```

### Public Attributes

- `std::string` [name](#)

### 4.15.1 Detailed Description

[entry\\_t](#) is a datatype used to store information that the chooser knows about an auto selection button

### 4.15.2 Member Data Documentation

#### **name**

```
std::string AutoChooser::entry_t::name
```

name of the auto represented by the block

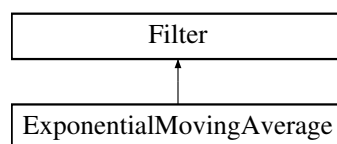
The documentation for this struct was generated from the following file:

- `auto_chooser.h`

## 4.16 ExponentialMovingAverage Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for ExponentialMovingAverage:



### Public Member Functions

- [ExponentialMovingAverage](#) (int buffer\_size)
- [ExponentialMovingAverage](#) (int buffer\_size, double starting\_value)
- void [add\\_entry](#) (double n) override
- double [get\\_value](#) () const override
- int [get\\_size](#) ()

### 4.16.1 Detailed Description

#### ExponentialMovingAverage

An exponential moving average is a way of smoothing out noisy data. For many sensor readings, the noise is roughly symmetric around the actual value. This means that if you collect enough samples those that are too high are cancelled out by the samples that are too low leaving the real value.

A simple moving average lags significantly with time as it has to counteract old samples. An exponential moving average keeps more up to date by weighting newer readings higher than older readings so it is more up to date while also still smoothed.

The [ExponentialMovingAverage](#) class provides an simple interface to do this smoothing from our noisy sensor values.

### 4.16.2 Constructor & Destructor Documentation

#### ExponentialMovingAverage() [1/2]

```
ExponentialMovingAverage::ExponentialMovingAverage (
    int buffer_size)
```

Create a moving average calculator with 0 as the default value

##### Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
--------------------	---

#### ExponentialMovingAverage() [2/2]

```
ExponentialMovingAverage::ExponentialMovingAverage (
    int buffer_size,
    double starting_value)
```

Create a moving average calculator with a specified default value

##### Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
<i>starting_value</i>	The value that the average will be before any data is added

### 4.16.3 Member Function Documentation

#### add\_entry()

```
void ExponentialMovingAverage::add_entry (
    double n) [override], [virtual]
```

Add a reading to the buffer Before: [ 1 1 2 2 3 3] => 2 ^ After: [ 2 1 2 2 3 3] => 2.16 ^

**Parameters**

$n$	the sample that will be added to the moving average.
-----	--

Implements [Filter](#).

**get\_size()**

```
int ExponentialMovingAverage::get_size ()
```

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

**get\_value()**

```
double ExponentialMovingAverage::get_value () const [override], [virtual]
```

Returns the average based off of all the samples collected so far

**Returns**

the calculated average.  $\text{sum}(\text{samples})/\text{numsamples}$

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

Implements [Filter](#).

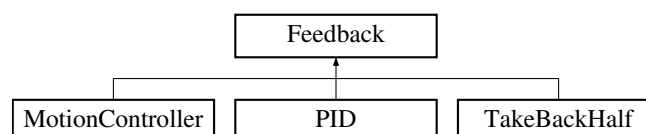
The documentation for this class was generated from the following files:

- moving\_average.h
- moving\_average.cpp

## 4.17 Feedback Class Reference

```
#include <feedback_base.h>
```

Inheritance diagram for Feedback:



## Public Member Functions

- virtual void [init](#) (double start\_pt, double set\_pt)=0
- virtual double [update](#) (double val)=0
- virtual double [get](#) ()=0
- virtual void [set\\_limits](#) (double lower, double upper)=0
- virtual bool [is\\_on\\_target](#) ()=0

### 4.17.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

#### Author

Ryan McGee

#### Date

9/25/2022

### 4.17.2 Member Function Documentation

#### get()

```
virtual double Feedback::get () [pure virtual]
```

#### Returns

the last saved result from the feedback controller

Implemented in [MotionController](#), [PID](#), and [TakeBackHalf](#).

#### init()

```
virtual void Feedback::init (  
    double start_pt,  
    double set_pt) [pure virtual]
```

Initialize the feedback controller for a movement

#### Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity
<i>end_vel</i>	Movement ending velocity

Implemented in [MotionController](#), [PID](#), and [TakeBackHalf](#).

**is\_on\_target()**

```
virtual bool Feedback::is_on_target () [pure virtual]
```

**Returns**

true if the feedback controller has reached it's setpoint

Implemented in [MotionController](#), [PID](#), and [TakeBackHalf](#).

**set\_limits()**

```
virtual void Feedback::set_limits (
    double lower,
    double upper) [pure virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

**Parameters**

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implemented in [MotionController](#), [PID](#), and [TakeBackHalf](#).

**update()**

```
virtual double Feedback::update (
    double val) [pure virtual]
```

Iterate the feedback loop once with an updated sensor value

**Parameters**

<i>val</i>	value from the sensor
------------	-----------------------

**Returns**

feedback loop result

Implemented in [MotionController](#), [PID](#), and [TakeBackHalf](#).

The documentation for this class was generated from the following file:

- [feedback\\_base.h](#)

## 4.18 FeedForward Class Reference

```
#include <feedforward.h>
```

## Classes

- struct [ff\\_config\\_t](#)

## Public Member Functions

- [FeedForward](#) ([ff\\_config\\_t](#) &cfg)
- double [calculate](#) (double v, double a, double pid\_ref=0.0)  
*Perform the feedforward calculation.*

### 4.18.1 Detailed Description

#### [FeedForward](#)

Stores the feedforward constants, and allows for quick computation. Feedforward should be used in systems that require smooth precise movements and have high inertia, such as drivetrains and lifts.

This is best used alongside a [PID](#) loop, with the form: `output = pid.get() + feedforward.calculate(v, a);`

In this case, the feedforward does the majority of the heavy lifting, and the pid loop only corrects for inconsistencies

For information about tuning feedforward, I recommend looking at this post: <https://www.chiefdelphi.com/t/paper-frc-drivetrain-characterization/160915> (yes I know it's for FRC but trust me, it's useful)

#### Author

Ryan McGee

#### Date

6/13/2022

### 4.18.2 Constructor & Destructor Documentation

#### [FeedForward\(\)](#)

```
FeedForward::FeedForward (
    ff_config_t & cfg) [inline]
```

Creates a [FeedForward](#) object.

#### Parameters

<code>cfg</code>	Configuration Struct for tuning
------------------	---------------------------------

### 4.18.3 Member Function Documentation

#### [calculate\(\)](#)

```
double FeedForward::calculate (
    double v,
    double a,
    double pid_ref = 0.0) [inline]
```

Perform the feedforward calculation.

This calculation is the equation:  $F = kG + kS*\text{sgn}(v) + kV*v + kA*a$

**Parameters**

$v$	Requested velocity of system
$a$	Requested acceleration of system

**Returns**

A feedforward that should closely represent the system if tuned correctly

The documentation for this class was generated from the following file:

- feedforward.h

## 4.19 FeedForward::ff\_config\_t Struct Reference

```
#include <feedforward.h>
```

**Public Attributes**

- double [kS](#)
- double [kV](#)
- double [kA](#)
- double [kG](#)

### 4.19.1 Detailed Description

[ff\\_config\\_t](#) holds the parameters to make the theoretical model of a real world system equation is of the form  $kS$  if the system is not stopped, 0 otherwise

- $kV * \text{desired velocity}$
- $kA * \text{desired acceleration}$
- $kG$

### 4.19.2 Member Data Documentation

**kA**

```
double FeedForward::ff_config_t::kA
```

kA - Acceleration coefficient: the power required to change the mechanism's speed. Multiplied by the requested acceleration.

**kG**

```
double FeedForward::ff_config_t::kG
```

kG - Gravity coefficient: only needed for lifts. The power required to overcome gravity and stay at steady state.

**kS**

```
double FeedForward::ff_config_t::kS
```

Coefficient to overcome static friction: the point at which the motor *starts* to move.

**kV**

```
double FeedForward::ff_config_t::kV
```

Veclocity coefficient: the power required to keep the mechanism in motion. Multiplied by the requested velocity.

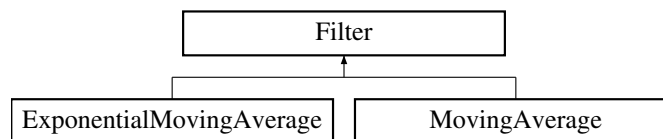
The documentation for this struct was generated from the following file:

- feedforward.h

## 4.20 Filter Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for Filter:



### 4.20.1 Detailed Description

Interface for filters Use `add_entry` to supply data and `get_value` to retrieve the filtered value

The documentation for this class was generated from the following file:

- moving\_average.h

## 4.21 Flywheel Class Reference

```
#include <flywheel.h>
```



## Public Member Functions

- [Flywheel](#) (vex::motor\_group &motors, [Feedback](#) &feedback, [FeedForward](#) &helper, const double ratio, [Filter](#) &filt)
- double [get\\_target](#) () const
- double [getRPM](#) () const
- vex::motor\_group & [get\\_motors](#) () const
- void [spin\\_manual](#) (double speed, directionType dir=fwd)
- void [spin\\_rpm](#) (double rpm)
- void [stop](#) ()
- bool [is\\_on\\_target](#) ()  
*check if the feedback controller thinks the flywheel is on target*
- [screen::Page](#) \* [Page](#) () const  
*Creates a page displaying info about the flywheel.*
- AutoCommand \* [SpinRpmCmd](#) (int rpm)  
*Creates a new auto command to spin the flywheel at the desired velocity.*
- AutoCommand \* [WaitUntilUpToSpeedCmd](#) ()  
*Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.*

## Friends

- int [spinRPMTask](#) (void \*wheelPointer)

### 4.21.1 Detailed Description

a [Flywheel](#) class that handles all control of a high inertia spinning disk. It gives multiple options for what control system to use in order to control wheel velocity and functions alerting the user when the flywheel is up to speed. [Flywheel](#) is a set and forget class. Once you create it you can call [spin\\_rpm](#) or [stop](#) on it at any time and it will take all necessary steps to accomplish this.

### 4.21.2 Constructor & Destructor Documentation

#### Flywheel()

```
Flywheel::Flywheel (
    vex::motor_group & motors,
    Feedback & feedback,
    FeedForward & helper,
    const double ratio,
    Filter & filt)
```

Create the [Flywheel](#) object using [PID](#) + feedforward for control.

#### Parameters

<i>motors</i>	pointer to the motors on the fly wheel
<i>feedback</i>	a feedback controller
<i>helper</i>	a feedforward config (only kV is used) to help the feedback controller along
<i>ratio</i>	ratio of the gears from the motor to the flywheel just multiplies the velocity
<i>filter</i>	the filter to use to smooth noisy motor readings

### 4.21.3 Member Function Documentation

#### **get\_motors()**

```
motor_group & Flywheel::get_motors () const
```

Returns the motors

##### Returns

the motors used to run the flywheel

#### **get\_target()**

```
double Flywheel::get_target () const
```

Return the target\_rpm that the flywheel is currently trying to achieve

##### Returns

target\_rpm the target rpm

Return the current value that the target\_rpm should be set to

#### **getRPM()**

```
double Flywheel::getRPM () const
```

return the velocity of the flywheel

#### **is\_on\_target()**

```
bool Flywheel::is_on_target () [inline]
```

check if the feedback controller thinks the flywheel is on target

##### Returns

true if on target

#### **Page()**

```
screen::Page * Flywheel::Page () const
```

Creates a page displaying info about the flywheel.

##### Returns

the page should be used for `screen::start_screen(screen, {fw.Page()});`

#### **spin\_manual()**

```
void Flywheel::spin_manual (  
    double speed,  
    directionType dir = fwd)
```

Spin motors using voltage; defaults forward at 12 volts FOR USE BY OPCONTROL AND AUTONOMOUS - this only applies if the target\_rpm thread is not running

**Parameters**

<i>speed</i>	- speed (between -1 and 1) to set the motor
<i>dir</i>	- direction that the motor moves in; defaults to forward

Spin motors using voltage; defaults forward at 12 volts FOR USE BY OPCONTROL AND AUTONOMOUS - this only applies if the RPM thread is not running

**Parameters**

<i>speed</i>	- speed (between -1 and 1) to set the motor
<i>dir</i>	- direction that the motor moves in; defaults to forward

**spin\_rpm()**

```
void Flywheel::spin_rpm (  
    double input_rpm)
```

starts or sets the target\_rpm thread at new value what control scheme is dependent on control\_style

**Parameters**

<i>rpm</i>	- the target_rpm we want to spin at
------------	-------------------------------------

starts or sets the RPM thread at new value what control scheme is dependent on control\_style

**Parameters**

<i>input_rpm</i>	- set the current RPM
------------------	-----------------------

**SpinRpmCmd()**

```
AutoCommand * Flywheel::SpinRpmCmd (  
    int rpm) [inline]
```

Creates a new auto command to spin the flywheel at the desired velocity.

**Parameters**

<i>rpm</i>	the rpm to spin at
------------	--------------------

**Returns**

an auto command to add to a command controller

**stop()**

```
void Flywheel::stop ()
```

Stops the motors. If manually spinning, this will do nothing just call `spin_manual(0.0)` to send 0 volts  
stop the RPM thread and the wheel

**WaitUntilUpToSpeedCmd()**

```
AutoCommand * Flywheel::WaitUntilUpToSpeedCmd () [inline]
```

Creates a new auto command that will hold until the flywheel has its target as defined by its feedback controller.

**Returns**

an auto command to add to a command controller

**4.21.4 Friends And Related Symbol Documentation****spinRPMTask**

```
int spinRPMTask (
    void * wheelPointer) [friend]
```

Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

The documentation for this class was generated from the following files:

- flywheel.h
- flywheel.cpp

**4.22 FlywheelStopCommand Class Reference**

```
#include <flywheel_commands.h>
```

**Public Member Functions**

- [FlywheelStopCommand](#) ([Flywheel](#) &flywheel)
- bool [run](#) () override

**4.22.1 Detailed Description**

AutoCommand wrapper class for the stop function in the [Flywheel](#) class

**4.22.2 Constructor & Destructor Documentation****FlywheelStopCommand()**

```
FlywheelStopCommand::FlywheelStopCommand (
    Flywheel & flywheel)
```

Construct a [FlywheelStopCommand](#)

**Parameters**

<i>flywheel</i>	the flywheel system we are commanding
-----------------	---------------------------------------

**4.22.3 Member Function Documentation****run()**

```
bool FlywheelStopCommand::run () [override]
```

Run stop Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- flywheel\_commands.h
- flywheel\_commands.cpp

**4.23 FlywheelStopMotorsCommand Class Reference**

```
#include <flywheel_commands.h>
```

**Public Member Functions**

- [FlywheelStopMotorsCommand](#) ([Flywheel](#) &flywheel)
- bool [run](#) () override

**4.23.1 Detailed Description**

AutoCommand wrapper class for the stopMotors function in the [Flywheel](#) class

**4.23.2 Constructor & Destructor Documentation****FlywheelStopMotorsCommand()**

```
FlywheelStopMotorsCommand::FlywheelStopMotorsCommand (  
    Flywheel & flywheel)
```

Construct a FlywheelStopMotors Command

## Parameters

<i>flywheel</i>	the flywheel system we are commanding
-----------------	---------------------------------------

## 4.23.3 Member Function Documentation

**run()**

```
bool FlywheelStopMotorsCommand::run () [override]
```

Run stop Overrides run from AutoCommand

## Returns

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- flywheel\_commands.h
- flywheel\_commands.cpp

## 4.24 FlywheelStopNonTasksCommand Class Reference

```
#include <flywheel_commands.h>
```

## 4.24.1 Detailed Description

AutoCommand wrapper class for the stopNonTasks function in the [Flywheel](#) class

The documentation for this class was generated from the following files:

- flywheel\_commands.h
- flywheel\_commands.cpp

## 4.25 FunctionCommand Class Reference

```
#include <auto_command.h>
```

## 4.25.1 Detailed Description

[FunctionCommand](#) is fun and good way to do simple things Printing, launching nukes, and other quick and dirty one time things

The documentation for this class was generated from the following file:

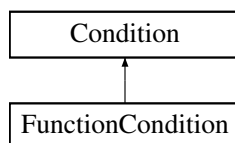
- auto\_command.h

## 4.26 FunctionCondition Class Reference

[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

```
#include <auto_command.h>
```

Inheritance diagram for FunctionCondition:



### 4.26.1 Detailed Description

[FunctionCondition](#) is a quick and dirty [Condition](#) to wrap some expression that should be evaluated at runtime.

The documentation for this class was generated from the following files:

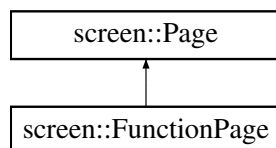
- `auto_command.h`
- `auto_command.cpp`

## 4.27 screen::FunctionPage Class Reference

Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

```
#include <screen.h>
```

Inheritance diagram for `screen::FunctionPage`:



### Public Member Functions

- [FunctionPage](#) (`update_func_t` update\_f, `draw_func_t` draw\_t)  
*Creates a function page.*
- void [update](#) (`bool` was\_pressed, `int` x, `int` y) override  
*update uses the supplied update function to update this page*
- void [draw](#) (`vex::brain::lcd` &, `bool` first\_draw, `unsigned int` frame\_number) override  
*draw uses the supplied draw function to draw to the screen*

### 4.27.1 Detailed Description

Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything.

### 4.27.2 Constructor & Destructor Documentation

#### FunctionPage()

```
screen::FunctionPage::FunctionPage (
    update_func_t update_f,
    draw_func_t draw_f)
```

Creates a function page.

[FunctionPage](#).

#### Parameters

<i>update↔ _f</i>	the function called every tick to respond to user input or do data collection
<i>draw_t</i>	the function called to draw to the screen
<i>update↔ _f</i>	drawing function
<i>draw_f</i>	drawing function

### 4.27.3 Member Function Documentation

#### draw()

```
void screen::FunctionPage::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number) [override], [virtual]
```

draw uses the supplied draw function to draw to the screen

#### See also

[Page::draw](#)

Reimplemented from [screen::Page](#).



## update()

```
void screen::FunctionPage::update (
    bool was_pressed,
    int x,
    int y) [override], [virtual]
```

update uses the supplied update function to update this page

See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

- screen.h
- screen.cpp

## 4.28 GenericAuto Class Reference

```
#include <generic_auto.h>
```

### Public Member Functions

- bool [run](#) (bool blocking)
- void [add](#) (state\_ptr new\_state)
- void [add\\_async](#) (state\_ptr async\_state)
- void [add\\_delay](#) (int ms)

#### 4.28.1 Detailed Description

[GenericAuto](#) provides a pleasant interface for organizing an auto path steps of the path can be added with [add\(\)](#) and when ready, calling [run\(\)](#) will begin executing the path

#### 4.28.2 Member Function Documentation

##### add()

```
void GenericAuto::add (
    state_ptr new_state)
```

Add a new state to the autonomous via function point of type "bool (ptr\*)()"

Parameters

<i>new_state</i>	the function to run
------------------	---------------------

##### add\_async()

```
void GenericAuto::add_async (
    state_ptr async_state)
```

Add a new state to the autonomous via function point of type "bool (ptr\*)()" that will run asynchronously

## Parameters

<i>async_state</i>	the function to run
--------------------	---------------------

**add\_delay()**

```
void GenericAuto::add_delay (
    int ms)
```

`add_delay` adds a period where the auto system will simply wait for the specified time

## Parameters

<i>ms</i>	how long to wait in milliseconds
-----------	----------------------------------

**run()**

```
bool GenericAuto::run (
    bool blocking)
```

The method that runs the autonomous. If 'blocking' is true, then this method will run through every state until it finished.

If blocking is false, then assuming every state is also non-blocking, the method will run through the current state in the list and return immediately.

## Parameters

<i>blocking</i>	Whether or not to block the thread until all states have run
-----------------	--

## Returns

true after all states have finished.

The documentation for this class was generated from the following files:

- generic\_auto.h
- generic\_auto.cpp

**4.29 PurePursuit::hermite\_point Struct Reference**

```
#include <pure_pursuit.h>
```

#### 4.29.1 Detailed Description

a position along the hermite path contains a position and orientation information that the robot would be at at this point

The documentation for this struct was generated from the following file:

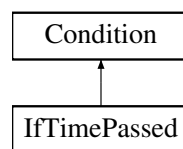
- pure\_pursuit.h

#### 4.30 IfTimePassed Class Reference

[IfTimePassed](#) tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.

```
#include <auto_command.h>
```

Inheritance diagram for IfTimePassed:



##### 4.30.1 Detailed Description

[IfTimePassed](#) tests based on time since the command controller was constructed. Returns true if elapsed time > time\_s.

The documentation for this class was generated from the following files:

- auto\_command.h
- auto\_command.cpp

#### 4.31 InOrder Class Reference

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

```
#include <auto_command.h>
```

##### 4.31.1 Detailed Description

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

[InOrder](#) runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?

The documentation for this class was generated from the following files:

- auto\_command.h
- auto\_command.cpp

## 4.32 Lift< T > Class Template Reference

```
#include <lift.h>
```

### Classes

- struct [lift\\_cfg\\_t](#)

### Public Member Functions

- [Lift](#) (motor\_group &lift\_motors, [lift\\_cfg\\_t](#) &lift\_cfg, map< T, double > &setpoint\_map, limit \*homing\_switch=NULL)
- void [control\\_continuous](#) (bool up\_ctrl, bool down\_ctrl)
- void [control\\_manual](#) (bool up\_btn, bool down\_btn, int volt\_up, int volt\_down)
- void [control\\_setpoints](#) (bool up\_step, bool down\_step, vector< T > pos\_list)
- bool [set\\_position](#) (T pos)
- bool [set\\_setpoint](#) (double val)
- double [get\\_setpoint](#) ()
- void [hold](#) ()
- void [home](#) ()
- bool [get\\_async](#) ()
- void [set\\_async](#) (bool val)
- void [set\\_sensor\\_function](#) (double(\*fn\_ptr)(void))
- void [set\\_sensor\\_reset](#) (void(\*fn\_ptr)(void))

### 4.32.1 Detailed Description

```
template<typename T>
class Lift< T >
```

LIFT A general class for lifts (e.g. 4bar, dr4bar, linear, etc) Uses a [PID](#) to hold the lift at a certain height under load, and to move the lift to different heights

#### Author

Ryan McGee

### 4.32.2 Constructor & Destructor Documentation

#### Lift()

```
template<typename T>
Lift< T >::Lift (
    motor_group & lift_motors,
    lift_cfg_t & lift_cfg,
    map< T, double > & setpoint_map,
    limit * homing_switch = NULL) [inline]
```

Construct the [Lift](#) object and begin the background task that controls the lift.

Usage example: 

```
/code{.cpp} enum Positions {UP, MID, DOWN}; map<Positions, double> setpt_map { {DOWN, 0.0}, {MID, 0.5}, {UP, 1.0} }; Lift<Positions> my_lift(motors, lift_cfg, setpt_map); /endcode
```

## Parameters

<i>lift_motors</i>	A set of motors, all set that positive rotation correlates with the lift going up
<i>lift_cfg</i>	Lift characterization information; PID tunings and movement speeds
<i>setpoint_map</i>	A map of enum type T, in which each enum entry corresponds to a different lift height

## 4.32.3 Member Function Documentation

**control\_continuous()**

```
template<typename T>
void Lift< T >::control_continuous (
    bool up_ctrl,
    bool down_ctrl) [inline]
```

Control the lift with an "up" button and a "down" button. Use PID to hold the lift when letting go.

## Parameters

<i>up_ctrl</i>	Button controlling the "UP" motion
<i>down_ctrl</i>	Button controlling the "DOWN" motion

**control\_manual()**

```
template<typename T>
void Lift< T >::control_manual (
    bool up_btn,
    bool down_btn,
    int volt_up,
    int volt_down) [inline]
```

Control the lift with manual controls (no holding voltage)

## Parameters

<i>up_btn</i>	Raise the lift when true
<i>down_btn</i>	Lower the lift when true
<i>volt_up</i>	Motor voltage when raising the lift
<i>volt_down</i>	Motor voltage when lowering the lift

**control\_setpoints()**

```
template<typename T>
void Lift< T >::control_setpoints (
    bool up_step,
    bool down_step,
    vector< T > pos_list) [inline]
```

Control the lift in "steps". When the "up" button is pressed, the lift will go to the next position as defined by pos\_list. Order matters!

**Parameters**

<i>up_step</i>	A button that increments the position of the lift.
<i>down_step</i>	A button that decrements the position of the lift.
<i>pos_list</i>	A list of positions for the lift to go through. The higher the index, the higher the lift should be (generally).

**get\_async()**

```
template<typename T>
bool Lift< T >::get_async () [inline]
```

**Returns**

whether or not the background thread is running the lift

**get\_setpoint()**

```
template<typename T>
double Lift< T >::get_setpoint () [inline]
```

**Returns**

The current setpoint for the lift

**hold()**

```
template<typename T>
void Lift< T >::hold () [inline]
```

Target the class's setpoint. Calculate the [PID](#) output and set the lift motors accordingly.

**home()**

```
template<typename T>
void Lift< T >::home () [inline]
```

A blocking function that automatically homes the lift based on a sensor or hard stop, and sets the position to 0. A watchdog times out after 3 seconds, to avoid damage.

**set\_async()**

```
template<typename T>
void Lift< T >::set_async (
    bool val) [inline]
```

Enables or disables the background task. Note that running the control functions, or `set_position` functions will immediately re-enable the task for autonomous use.

#### Parameters

<i>val</i>	Whether or not the background thread should run the lift
------------	--

### set\_position()

```
template<typename T>
bool Lift< T >::set_position (
    T pos) [inline]
```

Enable the background task, and send the lift to a position, specified by the setpoint map from the constructor.

#### Parameters

<i>pos</i>	A lift position enum type
------------	---------------------------

#### Returns

True if the pid has reached the setpoint

### set\_sensor\_function()

```
template<typename T>
void Lift< T >::set_sensor_function (
    double(* fn_ptr ) (void)) [inline]
```

Creates a custom hook for any other type of sensor to be used on the lift. Example: `/code{.cpp} my_lift.set_sensor_function( [](){return my_sensor.position();} ); /endcode`

#### Parameters

<i>fn_ptr</i>	Pointer to custom sensor function
---------------	-----------------------------------

### set\_sensor\_reset()

```
template<typename T>
void Lift< T >::set_sensor_reset (
    void(* fn_ptr ) (void)) [inline]
```

Creates a custom hook to reset the sensor used in [set\\_sensor\\_function\(\)](#). Example: `/code{.cpp} my_lift.set_sensor_reset( my_sensor.resetPosition ); /endcode`

### set\_setpoint()

```
template<typename T>
bool Lift< T >::set_setpoint (
    double val) [inline]
```

Manually set a setpoint value for the lift [PID](#) to go to.

## Parameters

val	Lift setpoint, in motor revolutions or sensor units defined by get_sensor. Cannot be outside the softstops.
-----	---

## Returns

True if the pid has reached the setpoint

The documentation for this class was generated from the following file:

- lift.h

## 4.33 Lift< T >::lift\_cfg\_t Struct Reference

```
#include <lift.h>
```

### 4.33.1 Detailed Description

```
template<typename T>
struct Lift< T >::lift_cfg_t
```

[lift\\_cfg\\_t](#) holds the physical parameter specifications of a lify system. includes:

- maximum speeds for the system
- softstops to stop the lift from hitting the hard stops too hard

The documentation for this struct was generated from the following file:

- lift.h

## 4.34 Logger Class Reference

Class to simplify writing to files.

```
#include <logger.h>
```



## Public Member Functions

- **Logger** (const std::string &filename)  
*Create a logger that will save to a file.*
- **Logger** (const **Logger** &l)=delete  
*copying not allowed*
- **Logger** & **operator=** (const **Logger** &l)=delete  
*copying not allowed*
- void **Log** (const std::string &s)  
*Write a string to the log.*
- void **Log** (LogLevel level, const std::string &s)  
*Write a string to the log with a loglevel.*
- void **Logln** (const std::string &s)  
*Write a string and newline to the log.*
- void **Logln** (LogLevel level, const std::string &s)  
*Write a string and a newline to the log with a loglevel.*
- void **Logf** (const char \*fmt,...)  
*Write a formatted string to the log.*
- void **Logf** (LogLevel level, const char \*fmt,...)  
*Write a formatted string to the log with a loglevel.*

## Static Public Attributes

- static constexpr int **MAX\_FORMAT\_LEN** = 512  
*maximum size for a string to be before it's written*

### 4.34.1 Detailed Description

Class to simplify writing to files.

### 4.34.2 Constructor & Destructor Documentation

#### Logger()

```
Logger::Logger (
    const std::string & filename) [explicit]
```

Create a logger that will save to a file.

#### Parameters

<i>filename</i>	the file to save to
-----------------	---------------------

### 4.34.3 Member Function Documentation

#### Log() [1/2]

```
void Logger::Log (
    const std::string & s)
```

Write a string to the log.

## Parameters

<i>s</i>	the string to write
----------	---------------------

**Log()** [2/2]

```
void Logger::Log (  
    LogLevel level,  
    const std::string & s)
```

Write a string to the log with a loglevel.

## Parameters

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>s</i>	the string to write

**Logf()** [1/2]

```
void Logger::Logf (  
    const char * fmt,  
    ...)
```

Write a formatted string to the log.

## Parameters

<i>fmt</i>	the format string (like printf)
...	the args

**Logf()** [2/2]

```
void Logger::Logf (  
    LogLevel level,  
    const char * fmt,  
    ...)
```

Write a formatted string to the log with a loglevel.

## Parameters

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>fmt</i>	the format string (like printf)
...	the args

**Logln()** [1/2]

```
void Logger::Logln (  
    const std::string & s)
```

Write a string and newline to the log.

**Parameters**

<i>s</i>	the string to write
----------	---------------------

**LogIn() [2/2]**

```
void Logger::LogIn (
    LogLevel level,
    const std::string & s)
```

Write a string and a newline to the log with a loglevel.

**Parameters**

<i>level</i>	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
<i>s</i>	the string to write

The documentation for this class was generated from the following files:

- logger.h
- logger.cpp

**4.35 MotionController::m\_profile\_cfg\_t Struct Reference**

```
#include <motion_controller.h>
```

**Public Attributes**

- double **max\_v**  
*the maximum velocity the robot can drive*
- double **accel**  
*the most acceleration the robot can do*
- [PID::pid\\_config\\_t](#) **pid\_cfg**  
*configuration parameters for the internal [PID](#) controller*
- [FeedForward::ff\\_config\\_t](#) **ff\_cfg**  
*configuration parameters for the internal*

**4.35.1 Detailed Description**

m\_profile\_config holds all data the motion controller uses to plan paths When motion profile is given a target to drive to, max\_v and accel are used to make the trapezoid profile instructing the controller how to drive pid\_cfg, ff\_cfg are used to find the motor outputs necessary to execute this path

The documentation for this struct was generated from the following file:

- motion\_controller.h

## 4.36 StateMachine< System, IDType, Message, delay\_ms, do\_log >::MaybeMessage Class Reference

[MaybeMessage](#) a message of Message type or nothing [MaybeMessage](#) m = {}; // empty [MaybeMessage](#) m = Message::EnumField1.

```
#include <state_machine.h>
```

### Public Member Functions

- [MaybeMessage](#) ()  
*Empty message - when theres no message.*
- [MaybeMessage](#) (Message msg)  
*Create a maybemessage with a message.*
- bool [has\\_message](#) ()  
*check if the message is here*
- Message [message](#) ()  
*Get the message stored. The return value is invalid unless has\_message returned true.*

### 4.36.1 Detailed Description

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
class StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage
```

[MaybeMessage](#) a message of Message type or nothing [MaybeMessage](#) m = {}; // empty [MaybeMessage](#) m = Message::EnumField1.

### 4.36.2 Constructor & Destructor Documentation

#### MaybeMessage()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
```

```
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::MaybeMessage (
    Message msg) [inline]
```

Create a maybemessage with a message.

#### Parameters

<i>msg</i>	the message to hold on to
------------	---------------------------

### 4.36.3 Member Function Documentation

#### has\_message()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
bool StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::has_message ()
[inline]
```

check if the message is here

#### Returns

true if there is a message

#### message()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
Message StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage::message ()
[inline]
```

Get the message stored. The return value is invalid unless has\_message returned true.

#### Returns

The message if it exists. Undefined otherwise

The documentation for this class was generated from the following file:

- [state\\_machine.h](#)

## 4.37 MecanumDrive Class Reference

```
#include <mecanum_drive.h>
```

### Classes

- struct [mecanumdrive\\_config\\_t](#)

### Public Member Functions

- [MecanumDrive](#) (vex::motor &left\_front, vex::motor &right\_front, vex::motor &left\_rear, vex::motor &right\_rear, vex::rotation \*lateral\_wheel=NULL, vex::inertial \*imu=NULL, [mecanumdrive\\_config\\_t](#) \*config=NULL)
- void [drive\\_raw](#) (double direction\_deg, double magnitude, double rotation)
- void [drive](#) (double left\_y, double left\_x, double right\_x, int power=2)
- bool [auto\\_drive](#) (double inches, double direction, double speed, bool gyro\_correction=true)
- bool [auto\\_turn](#) (double degrees, double speed, bool ignore\_imu=false)

### 4.37.1 Detailed Description

A class representing the Mecanum drivetrain. Contains 4 motors, a possible IMU (intertial), and a possible undriven perpendicular wheel.

### 4.37.2 Constructor & Destructor Documentation

#### MecanumDrive()

```
MecanumDrive::MecanumDrive (
    vex::motor & left_front,
    vex::motor & right_front,
    vex::motor & left_rear,
    vex::motor & right_rear,
    vex::rotation * lateral_wheel = NULL,
    vex::inertial * imu = NULL,
    mecanumdrive_config_t * config = NULL)
```

Create the Mecanum drivetrain object

### 4.37.3 Member Function Documentation

#### auto\_drive()

```
bool MecanumDrive::auto_drive (
    double inches,
    double direction,
    double speed,
    bool gyro_correction = true)
```

Drive the robot in a straight line automatically. If the inertial was declared in the constructor, use it to correct while driving. If the lateral wheel was declared in the constructor, use it for more accurate positioning while strafing.

#### Parameters

<i>inches</i>	How far the robot should drive, in inches
<i>direction</i>	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
<i>speed</i>	The maximum speed the robot should travel, in percent: -1.0->+1.0
<i>gyro_correction</i>	=true Whether or not to use the gyro to help correct while driving. Will always be false if no gyro was declared in the constructor.

Drive the robot in a straight line automatically. If the inertial was declared in the constructor, use it to correct while driving. If the lateral wheel was declared in the constructor, use it for more accurate positioning while strafing.

#### Parameters

<i>inches</i>	How far the robot should drive, in inches
<i>direction</i>	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
<i>speed</i>	The maximum speed the robot should travel, in percent: -1.0->+1.0
<i>gyro_correction</i>	= true Whether or not to use the gyro to help correct while driving. Will always be false if no gyro was declared in the constructor.

#### Returns

Whether or not the maneuver is complete.

**auto\_turn()**

```
bool MecanumDrive::auto_turn (
    double degrees,
    double speed,
    bool ignore_imu = false)
```

Autonomously turn the robot X degrees over it's center point. Uses a closed loop for control.

**Parameters**

<i>degrees</i>	How many degrees to rotate the robot. Clockwise postive.
<i>speed</i>	What percentage to run the motors at: 0.0 -> 1.0
<i>ignore_imu</i>	=false Whether or not to use the Inertial for determining angle. Will instead use circumference formula + robot's wheelbase + encoders to determine.

**Returns**

whether or not the robot has finished the maneuver

Autonomously turn the robot X degrees over it's center point. Uses a closed loop for control.

**Parameters**

<i>degrees</i>	How many degrees to rotate the robot. Clockwise postive.
<i>speed</i>	What percentage to run the motors at: 0.0 -> 1.0
<i>ignore_imu</i>	= false Whether or not to use the Inertial for determining angle. Will instead use circumference formula + robot's wheelbase + encoders to determine.

**Returns**

whether or not the robot has finished the maneuver

**drive()**

```
void MecanumDrive::drive (
    double left_y,
    double left_x,
    double right_x,
    int power = 2)
```

Drive the robot with a mecanum-style / arcade drive. Inputs are in percent (-100.0 -> 100.0) straight from the controller. Controls are mixed, so the robot can drive forward / strafe / rotate all at the same time.

**Parameters**

<i>left_y</i>	left joystick, Y axis (forward / backwards)
<i>left_x</i>	left joystick, X axis (strafe left / right)
<i>right_x</i>	right joystick, X axis (rotation left / right)
<i>power</i>	=2 how much of a "curve" there should be on drive controls; better for low speed maneuvers. Leave blank for a default curve of 2 (higher means more fidelity)

Drive the robot with a mecanum-style / arcade drive. Inputs are in percent (-100.0 -> 100.0) straight from the controller. Controls are mixed, so the robot can drive forward / strafe / rotate all at the same time.

## Parameters

<i>left_y</i>	left joystick, Y axis (forward / backwards)
<i>left_x</i>	left joystick, X axis (strafe left / right)
<i>right_x</i>	right joystick, X axis (rotation left / right)
<i>power</i>	= 2 how much of a "curve" there should be on drive controls; better for low speed maneuvers. Leave blank for a default curve of 2 (higher means more fidelity)

**drive\_raw()**

```
void MecanumDrive::drive_raw (
    double direction_deg,
    double magnitude,
    double rotation)
```

Drive the robot using vectors. This handles all the math required for mecanum control.

## Parameters

<i>direction_deg</i>	the direction to drive the robot, in degrees. 0 is forward, 180 is back, clockwise is positive, counterclockwise is negative.
<i>magnitude</i>	How fast the robot should drive, in percent: 0.0->1.0
<i>rotation</i>	How fast the robot should rotate, in percent: -1.0->+1.0

The documentation for this class was generated from the following files:

- mecanum\_drive.h
- mecanum\_drive.cpp

**4.38 MecanumDrive::mecanumdrive\_config\_t Struct Reference**

```
#include <mecanum_drive.h>
```

**4.38.1 Detailed Description**

Configure the Mecanum drive [PID](#) tunings and robot configurations

The documentation for this struct was generated from the following file:

- mecanum\_drive.h

**4.39 motion\_t Struct Reference**

```
#include <trapezoid_profile.h>
```



### Public Attributes

- double **pos**  
*1d position at this point in time*
- double **vel**  
*1d velocity at this point in time*
- double **accel**  
*1d acceleration at this point in time*

#### 4.39.1 Detailed Description

[motion\\_t](#) is a description of 1 dimensional motion at a point in time.

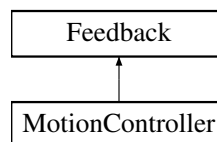
The documentation for this struct was generated from the following file:

- [trapezoid\\_profile.h](#)

## 4.40 MotionController Class Reference

```
#include <motion_controller.h>
```

Inheritance diagram for MotionController:



### Classes

- struct [m\\_profile\\_cfg\\_t](#)

### Public Member Functions

- [MotionController](#) ([m\\_profile\\_cfg\\_t](#) &config)  
*Construct a new Motion Controller object.*
- void [init](#) (double start\_pt, double end\_pt) override  
*Initialize the motion profile for a new movement This will also reset the [PID](#) and profile timers.*
- double [update](#) (double sensor\_val) override  
*Update the motion profile with a new sensor value.*
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override
- [motion\\_t get\\_motion](#) () const

### Static Public Member Functions

- static [FeedForward::ff\\_config\\_t](#) `tune_feedforward` ([TankDrive](#) &drive, [OdometryTank](#) &odometry, double pct=0.6, double duration=2)

#### 4.40.1 Detailed Description

Motion Controller class

This class defines a top-level motion profile, which can act as an intermediate between a subsystem class and the motors themselves

This takes the constants kS, kV, kA, kP, kI, kD, max\_v and acceleration and wraps around a feedforward, [PID](#) and trapezoid profile. It does so with the following formula:

```
out = feedforward.calculate(motion_profile.get(time_s)) + pid.get(motion_profile.get(time_s))
```

For [PID](#) and Feedforward specific formulae, see [pid.h](#), [feedforward.h](#), and [trapezoid\\_profile.h](#)

Author

Ryan McGee

Date

7/13/2022

#### 4.40.2 Constructor & Destructor Documentation

##### MotionController()

```
MotionController::MotionController (
    m_profile_cfg_t & config)
```

Construct a new Motion Controller object.

Parameters

<i>config</i>	The definition of how the robot is able to move max_v Maximum velocity the movement is capable of accel Acceleration / deceleration of the movement pid_cfg Definitions of kP, kI, and kD ff_cfg Definitions of kS, kV, and kA
---------------	--

#### 4.40.3 Member Function Documentation

##### get()

```
double MotionController::get () [override], [virtual]
```

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

## get\_motion()

```
motion_t MotionController::get_motion () const
```

### Returns

The current position, velocity and acceleration setpoints

## init()

```
void MotionController::init (
    double start_pt,
    double end_pt) [override], [virtual]
```

Initialize the motion profile for a new movement. This will also reset the [PID](#) and profile timers.

### Parameters

<i>start_pt</i>	Movement starting position
<i>end_pt</i>	Movement ending position

Implements [Feedback](#).

## is\_on\_target()

```
bool MotionController::is_on_target () [override], [virtual]
```

### Returns

Whether or not the movement has finished, and the [PID](#) confirms it is on target

Implements [Feedback](#).

## set\_limits()

```
void MotionController::set_limits (
    double lower,
    double upper) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied. If limits are applied, the controller will not target any value below lower or above upper

### Parameters

<i>lower</i>	upper limit
<i>upper</i>	lower limit

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.

## Parameters

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

**tune\_feedforward()**

```
FeedForward::ff_config_t MotionController::tune_feedforward (
    TankDrive & drive,
    OdometryTank & odometry,
    double pct = 0.6,
    double duration = 2) [static]
```

This method attempts to characterize the robot's drivetrain and automatically tune the feedforward. It does this by first calculating the kS (voltage to overcome static friction) by slowly increasing the voltage until it moves.

Next is kV (voltage to sustain a certain velocity), where the robot will record it's steady-state velocity at 'pct' speed.

Finally, kA (voltage needed to accelerate by a certain rate), where the robot will record the entire movement's velocity and acceleration, record a plot of  $[X=(pct-kV*V-kS), Y=(Acceleration)]$  along the movement, and since  $kA*Accel = pct-kV*V-kS$ , the reciprocal of the linear regression is the kA value.

## Parameters

<i>drive</i>	The tankdrive to operate on
<i>odometry</i>	The robot's odometry subsystem
<i>pct</i>	Maximum velocity in percent (0->1.0)
<i>duration</i>	Amount of time the robot should be moving for the test

## Returns

A tuned feedforward object

**update()**

```
double MotionController::update (
    double sensor_val) [override], [virtual]
```

Update the motion profile with a new sensor value.

## Parameters

<i>sensor_val</i>	Value from the sensor
-------------------	-----------------------

## Returns

the motor input generated from the motion profile

Implements [Feedback](#).

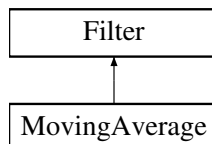
The documentation for this class was generated from the following files:

- motion\_controller.h
- motion\_controller.cpp

## 4.41 MovingAverage Class Reference

```
#include <moving_average.h>
```

Inheritance diagram for MovingAverage:



### Public Member Functions

- [MovingAverage](#) (int buffer\_size)
- [MovingAverage](#) (int buffer\_size, double starting\_value)
- void [add\\_entry](#) (double n) override
- double [get\\_value](#) () const override
- int [get\\_size](#) () const

#### 4.41.1 Detailed Description

##### [MovingAverage](#)

A moving average is a way of smoothing out noisy data. For many sensor readings, the noise is roughly symmetric around the actual value. This means that if you collect enough samples those that are too high are cancelled out by the samples that are too low leaving the real value.

The [MovingAverage](#) class provides a simple interface to do this smoothing from our noisy sensor values.

WARNING: because we need a lot of samples to get the actual value, the value given by the [MovingAverage](#) will 'lag' behind the actual value that the sensor is reading. Using a [MovingAverage](#) is thus a tradeoff between accuracy and lag time (more samples) vs. less accuracy and faster updating (less samples).

#### 4.41.2 Constructor & Destructor Documentation

##### **MovingAverage()** [1/2]

```
MovingAverage::MovingAverage (
    int buffer_size)
```

Create a moving average calculator with 0 as the default value

##### Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
--------------------	---

##### **MovingAverage()** [2/2]

```
MovingAverage::MovingAverage (
    int buffer_size,
    double starting_value)
```

Create a moving average calculator with a specified default value

## Parameters

<i>buffer_size</i>	The size of the buffer. The number of samples that constitute a valid reading
<i>starting_value</i>	The value that the average will be before any data is added

## 4.41.3 Member Function Documentation

**add\_entry()**

```
void MovingAverage::add_entry (  
    double n) [override], [virtual]
```

Add a reading to the buffer Before: [ 1 1 2 2 3 3] => 2 ^ After: [ 2 1 2 2 3 3] => 2.16 ^

## Parameters

<i>n</i>	the sample that will be added to the moving average.
----------	--

Implements [Filter](#).

**get\_size()**

```
int MovingAverage::get_size () const
```

How many samples the average is made from

## Returns

the number of samples used to calculate this average

**get\_value()**

```
double MovingAverage::get_value () const [override], [virtual]
```

Returns the average based off of all the samples collected so far

## Returns

the calculated average. `sum(samples)/numsamples`

How many samples the average is made from

## Returns

the number of samples used to calculate this average

Implements [Filter](#).

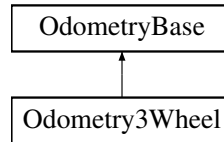
The documentation for this class was generated from the following files:

- `moving_average.h`
- `moving_average.cpp`

## 4.42 Odometry3Wheel Class Reference

```
#include <odometry_3wheel.h>
```

Inheritance diagram for Odometry3Wheel:



### Classes

- struct [odometry3wheel\\_cfg\\_t](#)

### Public Member Functions

- [Odometry3Wheel](#) ([CustomEncoder](#) &lside\_fwd, [CustomEncoder](#) &rside\_fwd, [CustomEncoder](#) &off\_axis, [odometry3wheel\\_cfg\\_t](#) &cfg, bool is\_async=true)
- [Pose2d update](#) () override
- void [tune](#) (vex::controller &con, [TankDrive](#) &drive)

### Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase](#) (bool is\_async)
- virtual [Pose2d get\\_position](#) (void)
- virtual void [set\\_position](#) (const [Pose2d](#) &newpos=zero\_pos)
- void [end\\_async](#) ()
- virtual double [get\\_speed](#) ()
- virtual double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

### Additional Inherited Members

### Static Public Member Functions inherited from [OdometryBase](#)

- static int [background\\_task](#) (void \*ptr)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

### Public Attributes inherited from [OdometryBase](#)

- bool **end\_task** = false  
*end\_task is true if we instruct the odometry thread to shut down*
- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [Pose2d](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 4.42.1 Detailed Description

#### Odometry3Wheel

This class handles the code for a standard 3-pod odometry setup, where there are 3 "pods" made up of undriven (dead) wheels connected to encoders in the following configuration:

+Y ----- ^ || || || || || O || || || || || == | | ----- | +-----> + X

Where O is the center of rotation. The robot will monitor the changes in rotation of these wheels and calculate the robot's X, Y and rotation on the field.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking it's movement in the background.

#### Author

Ryan McGee

#### Date

Oct 31 2022

### 4.42.2 Constructor & Destructor Documentation

#### Odometry3Wheel()

```
Odometry3Wheel::Odometry3Wheel (
    CustomEncoder & lside_fwd,
    CustomEncoder & rside_fwd,
    CustomEncoder & off_axis,
    odometry3wheel_cfg_t & cfg,
    bool is_async = true)
```

Construct a new Odometry 3 Wheel object

#### Parameters

<i>lside_fwd</i>	left-side encoder reference
<i>rside_fwd</i>	right-side encoder reference
<i>off_axis</i>	off-axis (perpendicular) encoder reference
<i>cfg</i>	robot odometry configuration
<i>is_async</i>	true to constantly run in the background

### 4.42.3 Member Function Documentation

#### tune()

```
void Odometry3Wheel::tune (
    vex::controller & con,
    TankDrive & drive)
```

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process



**Parameters**

<i>con</i>	Controller reference, for screen and button control
<i>drive</i>	Drivetrain reference for robot control

A guided tuning process to automatically find tuning parameters. This method is blocking, and returns when tuning has finished. Follow the instructions on the controller to complete the tuning process

It is assumed the gear ratio and encoder PPR have been set correctly

**update()**

```
Pose2d Odometry3Wheel::update () [override], [virtual]
```

Update the current position of the robot once, using the current state of the encoders and the previous known location

**Returns**

the robot's updated position

Implements [OdometryBase](#).

The documentation for this class was generated from the following files:

- odometry\_3wheel.h
- odometry\_3wheel.cpp

**4.43 Odometry3Wheel::odometry3wheel\_cfg\_t Struct Reference**

```
#include <odometry_3wheel.h>
```

**Public Attributes**

- double [wheelbase\\_dist](#)
- double [off\\_axis\\_center\\_dist](#)
- double [wheel\\_diam](#)

**4.43.1 Detailed Description**

[odometry3wheel\\_cfg\\_t](#) holds all the specifications for how to calculate position with 3 encoders See the core wiki for what exactly each of these parameters measures

**4.43.2 Member Data Documentation****off\_axis\_center\_dist**

```
double Odometry3Wheel::odometry3wheel_cfg_t::off_axis_center_dist
```

distance from the center of the robot to the center off axis wheel

**wheel\_diam**

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheel_diam
```

the diameter of the tracking wheel

**wheelbase\_dist**

```
double Odometry3Wheel::odometry3wheel_cfg_t::wheelbase_dist
```

distance from the center of the left wheel to the center of the right wheel

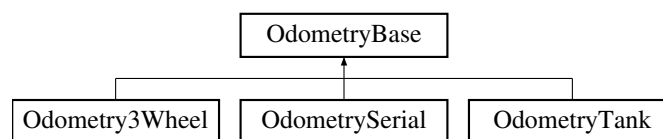
The documentation for this struct was generated from the following file:

- odometry\_3wheel.h

**4.44 OdometryBase Class Reference**

```
#include <odometry_base.h>
```

Inheritance diagram for OdometryBase:

**Public Member Functions**

- [OdometryBase](#) (bool is\_async)
- virtual [Pose2d](#) [get\\_position](#) (void)
- virtual void [set\\_position](#) (const [Pose2d](#) &newpos=zero\_pos)
- virtual [Pose2d](#) [update](#) ()=0
- void [end\\_async](#) ()
- virtual double [get\\_speed](#) ()
- virtual double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

**Static Public Member Functions**

- static int [background\\_task](#) (void \*ptr)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

## Public Attributes

- bool **end\_task** = false  
*end\_task is true if we instruct the odometry thread to shut down*
- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [Pose2d](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 4.44.1 Detailed Description

#### [OdometryBase](#)

This base class contains all the shared code between different implementations of odometry. It handles the asynchronous management, position input/output and basic math functions, and holds positional types specific to field orientation.

All future odometry implementations should extend this file and redefine [update\(\)](#) function.

#### Author

Ryan McGee

#### Date

Aug 11 2021

### 4.44.2 Constructor & Destructor Documentation

#### **OdometryBase()**

```
OdometryBase::OdometryBase (  
    bool is_async)
```

Construct a new Odometry Base object

#### Parameters

<i>is_async</i>	True to run constantly in the background, false to call <a href="#">update()</a> manually
-----------------	---

### 4.44.3 Member Function Documentation

#### **background\_task()**

```
int OdometryBase::background_task (  
    void * ptr) [static]
```

Function that runs in the background task. This function pointer is passed to the vex::task constructor.

## Parameters

<i>ptr</i>	Pointer to <a href="#">OdometryBase</a> object
------------	--

## Returns

Required integer return code. Unused.

**end\_async()**

```
void OdometryBase::end_async ()
```

End the background task. Cannot be restarted. If the user wants to end the thread but keep the data up to date, they must run the [update\(\)](#) function manually from then on.

**get\_accel()**

```
double OdometryBase::get_accel () [virtual]
```

Get the current acceleration

## Returns

the acceleration rate of the robot (inch/s<sup>2</sup>)

**get\_angular\_accel\_deg()**

```
double OdometryBase::get_angular_accel_deg ()
```

Get the current angular acceleration in degrees

## Returns

the angular acceleration at which we are turning (deg/s<sup>2</sup>)

**get\_angular\_speed\_deg()**

```
double OdometryBase::get_angular_speed_deg ()
```

Get the current angular speed in degrees

## Returns

the angular velocity at which we are turning (deg/s)

**get\_position()**

```
Pose2d OdometryBase::get_position (
    void ) [virtual]
```

Gets the current position and rotation

**Returns**

the position that the odometry believes the robot is at

Gets the current position and rotation

Reimplemented in [OdometrySerial](#).

**get\_speed()**

```
double OdometryBase::get_speed () [virtual]
```

Get the current speed

**Returns**

the speed at which the robot is moving and grooving (inch/s)

**set\_position()**

```
void OdometryBase::set_position (
    const Pose2d & newpos = zero_pos) [virtual]
```

Sets the current position of the robot

**Parameters**

<i>newpos</i>	the new position that the odometry will believe it is at
---------------	--

Sets the current position of the robot

Reimplemented in [OdometrySerial](#), and [OdometryTank](#).

**smallest\_angle()**

```
double OdometryBase::smallest_angle (
    double start_deg,
    double end_deg) [static]
```

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

## Parameters

<i>start_deg</i>	initial angle (degrees)
<i>end_deg</i>	final angle (degrees)

## Returns

the smallest angle from the initial to the final angle. This takes into account the wrapping of rotations around 360 degrees

Get the smallest difference in angle between a start heading and end heading. Returns the difference between -180 degrees and +180 degrees, representing the robot turning left or right, respectively.

**update()**

```
virtual Pose2d OdometryBase::update () [pure virtual]
```

Update the current position on the field based on the sensors

## Returns

the location that the robot is at after the odometry does its calculations

Implemented in [Odometry3Wheel](#), [OdometrySerial](#), and [OdometryTank](#).

**4.44.4 Member Data Documentation****accel**

```
double OdometryBase::accel
```

the rate at which we are accelerating (inch/s<sup>2</sup>)

**ang\_accel\_deg**

```
double OdometryBase::ang_accel_deg
```

the rate at which we are accelerating our turn (deg/s<sup>2</sup>)

**ang\_speed\_deg**

```
double OdometryBase::ang_speed_deg
```

the speed at which we are turning (deg/s)

### current\_pos

```
Pose2d OdometryBase::current_pos
```

Current position of the robot in terms of x,y,rotation

### handle

```
vex::task* OdometryBase::handle
```

handle to the vex task that is running the odometry code

### mut

```
vex::mutex OdometryBase::mut
```

Mutex to control multithreading

### speed

```
double OdometryBase::speed
```

the speed at which we are travelling (inch/s)

The documentation for this class was generated from the following files:

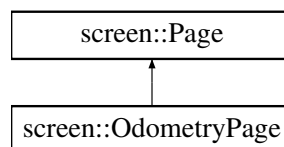
- odometry\_base.h
- odometry\_base.cpp

## 4.45 screen::OdometryPage Class Reference

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

```
#include <screen.h>
```

Inheritance diagram for screen::OdometryPage:



### Public Member Functions

- [OdometryPage](#) ([OdometryBase](#) &odom, double robot\_width, double robot\_height, bool do\_trail)  
*Create an odometry trail. Make sure odometry is initlized before now.*
- void [update](#) (bool was\_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

### 4.45.1 Detailed Description

a page that shows odometry position and rotation and a map (if an sd card with the file is on)

### 4.45.2 Constructor & Destructor Documentation

#### OdometryPage()

```
screen::OdometryPage::OdometryPage (
    OdometryBase & odom,
    double robot_width,
    double robot_height,
    bool do_trail)
```

Create an odometry trail. Make sure odometry is initilized before now.

#### Parameters

<i>odom</i>	the odometry system to monitor
<i>robot_width</i>	the width (side to side) of the robot in inches. Used for visualization
<i>robot_height</i>	the robot_height (front to back) of the robot in inches. Used for visualization
<i>do_trail</i>	whether or not to calculate and draw the trail. Drawing and storing takes a very <i>slight</i> extra amount of processing power

### 4.45.3 Member Function Documentation

#### draw()

```
void screen::OdometryPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number) [override], [virtual]
```

#### See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

#### update()

```
void screen::OdometryPage::update (
    bool was_pressed,
    int x,
    int y) [override], [virtual]
```

#### See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

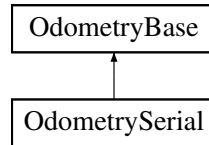
- screen.h
- screen.cpp



## 4.46 OdometrySerial Class Reference

```
#include <odometry_serial.h>
```

Inheritance diagram for OdometrySerial:



### Public Member Functions

- [OdometrySerial](#) (bool is\_async, bool calc\_vel\_acc\_on\_brain, [Pose2d](#) initial\_pose, [Pose2d](#) sensor\_offset, int32\_t port, int32\_t baudrate)
- void [send\\_config](#) (const [Pose2d](#) &initial\_pose, const [Pose2d](#) &sensor\_offset, const bool &calc\_vel\_acc\_on\_brain)
- [Pose2d](#) [update](#) () override
- void [set\\_position](#) (const [Pose2d](#) &new\_pose) override
- int [receive\\_cobs\\_packet](#) (uint32\_t port, uint8\_t \*buffer, size\_t buffer\_size)
- [Pose2d](#) [get\\_position](#) (void) override
- [Pose2d](#) [get\\_pose2d](#) (void)
- size\_t [cobs\\_decode](#) (const uint8\_t \*buffer, size\_t length, void \*data)
- size\_t [cobs\\_encode](#) (const void \*data, size\_t length, uint8\_t \*buffer)

### Public Member Functions inherited from [OdometryBase](#)

- [OdometryBase](#) (bool is\_async)
- void [end\\_async](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

### Additional Inherited Members

### Static Public Member Functions inherited from [OdometryBase](#)

- static int [background\\_task](#) (void \*ptr)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

### Public Attributes inherited from [OdometryBase](#)

- bool [end\\_task](#) = false  
*end\_task is true if we instruct the odometry thread to shut down*
- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [Pose2d](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 4.46.1 Detailed Description

#### OdometrySerial

This class handles the code for an odometry setup where calculations are done on an external coprocessor. Data is sent to the brain via smart port, using a generic serial (UART) connection.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking it's movement in the background.

<https://rit.enterprise.slack.com/files/U04112Y5RB6/F080M01KPA5/predictperpindiculars2.pdf> 2024-2025 Notebook: Entries/Software Entries/Localization/N-Pod Odometry

#### Author

Jack Cammarata

#### Date

Jan 16 2025

### 4.46.2 Constructor & Destructor Documentation

#### OdometrySerial()

```
OdometrySerial::OdometrySerial (
    bool is_async,
    bool calc_vel_acc_on_brain,
    Pose2d initial_pose,
    Pose2d sensor_offset,
    int32_t port,
    int32_t baudrate)
```

Construct a new Odometry Serial Object

#### OdometrySerial

This class handles the code for an odometry setup where calculations are done on an external coprocessor. Data is sent to the brain via smart port, using a generic serial (UART) connection.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking it's movement in the background.

<https://rit.enterprise.slack.com/files/U04112Y5RB6/F080M01KPA5/predictperpindiculars2.pdf> 2024-2025 Notebook: Entries/Software Entries/Localization/N-Pod Odometry

#### Author

Jack Cammarata

#### Date

Jan 16 2025 Construct a new Odometry Serial Object

### 4.46.3 Member Function Documentation

#### cobs\_decode()

```
size_t OdometrySerial::cobs_decode (
    const uint8_t * buffer,
    size_t length,
    void * data)
```

COBS decode data from buffer

**Parameters**

<i>buffer</i>	Pointer to encoded input bytes
<i>length</i>	Number of bytes to decode
<i>data</i>	Pointer to decoded output data

**Returns**

Number of bytes successfully decoded

**Note**

Stops decoding if delimiter byte is found

**cobs\_encode()**

```
size_t OdometrySerial::cobs_encode (
    const void * data,
    size_t length,
    uint8_t * buffer)
```

COBS encode data to buffer

**Parameters**

<i>data</i>	Pointer to input data to encode
<i>length</i>	Number of bytes to encode
<i>buffer</i>	Pointer to encoded output buffer

**Returns**

Encoded buffer length in bytes

**Note**

Does not output delimiter byte

**get\_pose2d()**

```
Pose2d OdometrySerial::get_pose2d (
    void )
```

Gets the current position and rotation

**Returns**

the position that the odometry believes the robot is at

**get\_position()**

```
Pose2d OdometrySerial::get_position (
    void ) [override], [virtual]
```

Gets the current position and rotation

**Returns**

the position that the odometry believes the robot is at

Reimplemented from [OdometryBase](#).

**receive\_cobs\_packet()**

```
int OdometrySerial::receive_cobs_packet (
    uint32_t port,
    uint8_t * buffer,
    size_t buffer_size)
```

Attempts to receive an entire packet encoded with COBS, stops at delimiter or there's a buffer overflow

**Parameters**

<i>port</i>	the port number the serial is plugged into, counts from 0 instead of 1
<i>buffer</i>	pointer to a <code>uint8_t[]</code> where we put the data
<i>buffer_size</i>	length in bytes of the buffer

**Returns**

0 success

**send\_config()**

```
void OdometrySerial::send_config (
    const Pose2d & initial_pose,
    const Pose2d & sensor_offset,
    const bool & calc_vel_acc_on_brain)
```

Send

**set\_position()**

```
void OdometrySerial::set_position (
    const Pose2d & new_pose) [override], [virtual]
```

Resets the position and rotational data to the input.

Resets the position and rotational data to the input.

**Parameters**

<code>new_pose</code>	the pose to set the odometry to
-----------------------	---------------------------------

Reimplemented from [OdometryBase](#).

**update()**

```
Pose2d OdometrySerial::update () [override], [virtual]
```

Update the current position of the robot once by reading a single packet from the serial port

**Returns**

the robot's updated position

Update the current position of the robot once by reading a single packet from the serial port, then updating all over values, velocity, accel

**Returns**

the robot's updated position

Implements [OdometryBase](#).

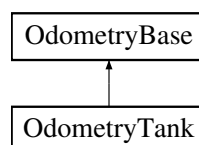
The documentation for this class was generated from the following files:

- odometry\_serial.h
- odometry\_serial.cpp

**4.47 OdometryTank Class Reference**

```
#include <odometry_tank.h>
```

Inheritance diagram for OdometryTank:

**Public Member Functions**

- [OdometryTank](#) (vex::motor\_group &left\_side, vex::motor\_group &right\_side, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [OdometryTank](#) ([CustomEncoder](#) &left\_custom\_enc, [CustomEncoder](#) &right\_custom\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [OdometryTank](#) (vex::encoder &left\_vex\_enc, vex::encoder &right\_vex\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [Pose2d update](#) () override
- void [set\\_position](#) (const [Pose2d](#) &newpos=zero\_pos) override

**Public Member Functions inherited from [OdometryBase](#)**

- [OdometryBase](#) (bool is\_async)
- virtual [Pose2d](#) [get\\_position](#) (void)
- void [end\\_async](#) ()
- virtual double [get\\_speed](#) ()
- virtual double [get\\_accel](#) ()
- double [get\\_angular\\_speed\\_deg](#) ()
- double [get\\_angular\\_accel\\_deg](#) ()

**Additional Inherited Members****Static Public Member Functions inherited from [OdometryBase](#)**

- static int [background\\_task](#) (void \*ptr)
- static double [smallest\\_angle](#) (double start\_deg, double end\_deg)

**Public Attributes inherited from [OdometryBase](#)**

- bool **end\_task** = false  
*end\_task is true if we instruct the odometry thread to shut down*
- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [Pose2d](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

**4.47.1 Detailed Description**

[OdometryTank](#) defines an odometry system for a tank drivetrain. This requires encoders in the same orientation as the drive wheels. Odometry is a "start and forget" subsystem, which means once it's created and configured, it will constantly run in the background and track the robot's X, Y and rotation coordinates.

**4.47.2 Constructor & Destructor Documentation****OdometryTank()** [1/3]

```
OdometryTank::OdometryTank (
    vex::motor_group & left_side,
    vex::motor_group & right_side,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true)
```

Initialize the Odometry module, calculating position from the drive motors.

## Parameters

<i>left_side</i>	The left motors
<i>right_side</i>	The right motors
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

**OdometryTank()** [2/3]

```
OdometryTank::OdometryTank (
    CustomEncoder & left_custom_enc,
    CustomEncoder & right_custom_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true)
```

Initialize the Odometry module, calculating position from the drive motors.

## Parameters

<i>left_custom_enc</i>	The left custom encoder
<i>right_custom_enc</i>	The right custom encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

**OdometryTank()** [3/3]

```
OdometryTank::OdometryTank (
    vex::encoder & left_vex_enc,
    vex::encoder & right_vex_enc,
    robot_specs_t & config,
    vex::inertial * imu = NULL,
    bool is_async = true)
```

Initialize the Odometry module, calculating position from the drive motors.

## Parameters

<i>left_vex_enc</i>	The left vex encoder
<i>right_vex_enc</i>	The right vex encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See <a href="#">robot_specs_t</a> for what is contained
<i>imu</i>	The robot's inertial sensor. If not included, rotation is calculated from the encoders.
<i>is_async</i>	If true, position will be updated in the background continuously. If false, the programmer will have to manually call <a href="#">update()</a> .

### 4.47.3 Member Function Documentation

#### set\_position()

```
void OdometryTank::set_position (
    const Pose2d & newpos = zero_pos) [override], [virtual]
```

set\_position tells the odometry to place itself at a position

##### Parameters

<i>newpos</i>	the position the odometry will take
---------------	-------------------------------------

Resets the position and rotational data to the input.

Reimplemented from [OdometryBase](#).

#### update()

```
Pose2d OdometryTank::update () [override], [virtual]
```

Update the current position on the field based on the sensors

##### Returns

the position that odometry has calculated itself to be at

Update, store and return the current position of the robot. Only use if not initializing with a separate thread.

Implements [OdometryBase](#).

The documentation for this class was generated from the following files:

- odometry\_tank.h
- odometry\_tank.cpp

## 4.48 OdomSetPosition Class Reference

```
#include <drive_commands.h>
```

### Public Member Functions

- [OdomSetPosition](#) ([OdometryBase](#) &odom, const [Pose2d](#) &newpos=[OdometryBase::zero\\_pos](#))
- bool [run](#) () override

#### 4.48.1 Detailed Description

AutoCommand wrapper class for the set\_position function in the Odometry class

#### 4.48.2 Constructor & Destructor Documentation

##### OdomSetPosition()

```
OdomSetPosition::OdomSetPosition (
    OdometryBase & odom,
    const Pose2d & newpos = OdometryBase::zero_pos)
```

constructs a new [OdomSetPosition](#) command



**Parameters**

<i>odom</i>	the odometry system we are setting
<i>newpos</i>	the position we are telling the odometry to take. defaults to (0, 0), angle = 90

Construct an Odometry set pos

**Parameters**

<i>odom</i>	the odometry system we are setting
<i>newpos</i>	the now position to set the odometry to

**4.48.3 Member Function Documentation****run()**

```
bool OdomSetPosition::run () [override]
```

Run set\_position Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

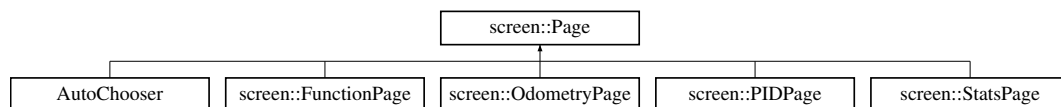
- drive\_commands.h
- drive\_commands.cpp

**4.49 screen::Page Class Reference**

[Page](#) describes one part of the screen slideshow.

```
#include <screen.h>
```

Inheritance diagram for screen::Page:

**Public Member Functions**

- virtual void [update](#) (bool was\_pressed, int x, int y)  
*collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))*
- virtual void [draw](#) (vex::brain::lcd &screen, bool first\_draw, unsigned int frame\_number)  
*draw stored data to the screen (runs at 10 hz and only runs if this page is in front)*

### 4.49.1 Detailed Description

[Page](#) describes one part of the screen slideshow.

### 4.49.2 Member Function Documentation

#### draw()

```
virtual void screen::Page::draw (
    vex::brain::lcd & screen,
    bool first_draw,
    unsigned int frame_number) [virtual]
```

draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

#### Parameters

<i>first_draw</i>	true if we just switched to this page
<i>frame_number</i>	frame of drawing we are on (basically an animation tick)

Reimplemented in [screen::FunctionPage](#), [screen::OdometryPage](#), [screen::PIDPage](#), and [screen::StatsPage](#).

#### update()

```
virtual void screen::Page::update (
    bool was_pressed,
    int x,
    int y) [virtual]
```

collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))

#### Parameters

<i>was_pressed</i>	true if the screen has been pressed
<i>x</i>	x position of screen press (if the screen was pressed)
<i>y</i>	y position of screen press (if the screen was pressed)

Reimplemented in [screen::FunctionPage](#), [screen::OdometryPage](#), [screen::PIDPage](#), and [screen::StatsPage](#).

The documentation for this class was generated from the following file:

- [screen.h](#)

## 4.50 Parallel Class Reference

[Parallel](#) runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

```
#include <auto_command.h>
```

#### 4.50.1 Detailed Description

[Parallel](#) runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call `on_timeout` on all children continue.

The documentation for this class was generated from the following files:

- `auto_command.h`
- `auto_command.cpp`

### 4.51 PurePursuit::Path Class Reference

```
#include <pure_pursuit.h>
```

#### Public Member Functions

- [Path](#) (`std::vector< Translation2d > points`, `double radius`)
- `const std::vector< Translation2d > get\_points ()`
- `double get\_radius ()`
- `bool is\_valid ()`

#### 4.51.1 Detailed Description

Wrapper for a vector of points, checking if any of the points are too close for pure pursuit

#### 4.51.2 Constructor & Destructor Documentation

##### Path()

```
PurePursuit::Path::Path (
    std::vector< Translation2d > points,
    double radius)
```

Create a [Path](#)

##### Parameters

<i>points</i>	the points that make up the path
<i>radius</i>	the lookahead radius for pure pursuit

#### 4.51.3 Member Function Documentation

##### get\_points()

```
const std::vector< Translation2d > PurePursuit::Path::get_points ()
```

Get the points associated with this [Path](#)

**get\_radius()**

```
double PurePursuit::Path::get_radius ()
```

Get the radius associated with this [Path](#)

**is\_valid()**

```
bool PurePursuit::Path::is_valid ()
```

Get whether this path will behave as expected

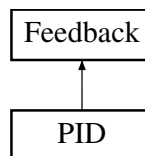
The documentation for this class was generated from the following files:

- [pure\\_pursuit.h](#)
- [pure\\_pursuit.cpp](#)

**4.52 PID Class Reference**

```
#include <pid.h>
```

Inheritance diagram for PID:

**Classes**

- struct [pid\\_config\\_t](#)

**Public Types**

- enum [ERROR\\_TYPE](#)

**Public Member Functions**

- [PID](#) ([pid\\_config\\_t](#) &config)
- void [init](#) (double start\_pt, double set\_pt) override
- double [update](#) (double sensor\_val) override
- double [update](#) (double sensor\_val, double v\_setpt)
- double [get\\_sensor\\_val](#) () const  
*gets the sensor value that we were last updated with*
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override
- void [reset](#) ()
- double [get\\_error](#) ()
- double [get\\_target](#) () const
- void [set\\_target](#) (double target)

## Public Attributes

- [pid\\_config\\_t](#) & [config](#)

### 4.52.1 Detailed Description

#### [PID](#) Class

Defines a standard feedback loop using the constants kP, kI, kD, deadband, and on\_target\_time. The formula is:

$$\text{out} = kP * \text{error} + kI * \text{integral}(\text{d Error}) + kD * (\text{dError}/\text{dt})$$

The [PID](#) object will determine it is "on target" when the error is within the deadband, for a duration of on\_target\_time

#### Author

Ryan McGee

#### Date

4/3/2020

### 4.52.2 Member Enumeration Documentation

#### **ERROR\_TYPE**

```
enum PID::ERROR\_TYPE
```

An enum to distinguish between a linear and angular caluculation of [PID](#) error.

### 4.52.3 Constructor & Destructor Documentation

#### **PID()**

```
PID::PID (  
    pid\_config\_t & config)
```

Create the [PID](#) object

#### Parameters

<i>config</i>	the configuration data for this controller
---------------	--

Create the [PID](#) object

#### 4.52.4 Member Function Documentation

##### get()

```
double PID::get () [override], [virtual]
```

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

##### Returns

the Out value of the controller (voltage, RPM, whatever the [PID](#) controller is controlling)

Gets the current [PID](#) out value, from when [update\(\)](#) was last run

Implements [Feedback](#).

##### get\_error()

```
double PID::get_error ()
```

Get the delta between the current sensor data and the target

##### Returns

the error calculated. how it is calculated depends on `error_method` specified in [pid\\_config\\_t](#)

Get the delta between the current sensor data and the target

##### get\_sensor\_val()

```
double PID::get_sensor_val () const
```

gets the sensor value that we were last updated with

##### Returns

sensor\_val

##### get\_target()

```
double PID::get_target () const
```

Get the [PID](#)'s target

##### Returns

the target the [PID](#) controller is trying to achieve

##### init()

```
void PID::init (  
    double start_pt,  
    double set_pt) [override], [virtual]
```

Inherited from [Feedback](#) for interoperability. Update the setpoint and reset integral accumulation

`start_pt` can be safely ignored in this feedback controller

#### Parameters

<i>start_pt</i>	completely ignored for <a href="#">PID</a> . necessary to satisfy <a href="#">Feedback</a> base
<i>set_pt</i>	sets the target of the <a href="#">PID</a> controller
<i>start_vel</i>	completely ignored for <a href="#">PID</a> . necessary to satisfy <a href="#">Feedback</a> base
<i>end_vel</i>	sets the target end velocity of the <a href="#">PID</a> controller

Implements [Feedback](#).

#### is\_on\_target()

```
bool PID::is_on_target () [override], [virtual]
```

Checks if the [PID](#) controller is on target.

#### Returns

true if the loop is within [deadband] for [on\_target\_time] seconds

Returns true if the loop is within [deadband] for [on\_target\_time] seconds

Implements [Feedback](#).

#### reset()

```
void PID::reset ()
```

Reset the [PID](#) loop by resetting time since 0 and accumulated error.

#### set\_limits()

```
void PID::set_limits (
    double lower,
    double upper) [override], [virtual]
```

Set the limits on the [PID](#) out. The [PID](#) out will "clip" itself to be between the limits.

#### Parameters

<i>lower</i>	the lower limit. the <a href="#">PID</a> controller will never command the output go below <i>lower</i>
<i>upper</i>	the upper limit. the <a href="#">PID</a> controller will never command the output go higher than <i>upper</i>

Set the limits on the [PID](#) out. The [PID](#) out will "clip" itself to be between the limits.

Implements [Feedback](#).

#### set\_target()

```
void PID::set_target (
    double target)
```

Set the target for the [PID](#) loop, where the robot is trying to end up

## Parameters

<i>target</i>	the sensor reading we would like to achieve
---------------	---

Set the target for the [PID](#) loop, where the robot is trying to end up

**update()** [1/2]

```
double PID::update (
    double sensor_val) [override], [virtual]
```

Update the [PID](#) loop by taking the time difference from last update, and running the [PID](#) formula with the new sensor data

## Parameters

<i>sensor_val</i>	the distance, angle, encoder position or whatever it is we are measuring
-------------------	--

## Returns

the new output. What would be returned by [PID::get\(\)](#)

Implements [Feedback](#).

**update()** [2/2]

```
double PID::update (
    double sensor_val,
    double v_setpt)
```

Update the [PID](#) loop by taking the time difference from last update, and running the [PID](#) formula with the new sensor data

## Parameters

<i>sensor_val</i>	the distance, angle, encoder position or whatever it is we are measuring
<i>v_setpt</i>	Expected velocity setpoint, to subtract from the D term (for velocity control)

## Returns

the new output. What would be returned by [PID::get\(\)](#)

**4.52.5 Member Data Documentation****config**

[pid\\_config\\_t](#)& [PID::config](#)

configuration struct for this controller. see [pid\\_config\\_t](#) for information about what this contains

The documentation for this class was generated from the following files:

- [pid.h](#)
- [pid.cpp](#)



## 4.53 PID::pid\_config\_t Struct Reference

```
#include <pid.h>
```

### Public Attributes

- double **p**  
*proportional coefficient  $p * error()$*
- double **i**  
*integral coefficient  $i * integral(error)$*
- double **d**  
*derivitave coefficient  $d * derivative(error)$*
- double **deadband**  
*at what threshold are we close enough to be finished*
- double [on\\_target\\_time](#)
- [ERROR\\_TYPE](#) [error\\_method](#)

### 4.53.1 Detailed Description

[pid\\_config\\_t](#) holds the configuration parameters for a pid controller In addition to the constant of proportional, integral and derivative, these parameters include:

- deadband -
- on\_target\_time - for how long do we have to be at the target to stop As well, [pid\\_config\\_t](#) holds an error type which determines whether errors should be calculated as if the sensor position is a measure of distance or an angle

### 4.53.2 Member Data Documentation

#### error\_method

[ERROR\\_TYPE](#) PID::pid\_config\_t::error\_method

Linear or angular. wheter to do error as a simple subtraction or to wrap

#### on\_target\_time

double PID::pid\_config\_t::on\_target\_time

the time in seconds that we have to be on target for to say we are officially at the target

The documentation for this struct was generated from the following file:

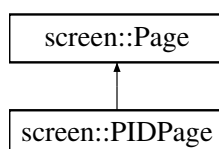
- [pid.h](#)

## 4.54 screen::PIDPage Class Reference

[PIDPage](#) provides a way to tune a pid controller on the screen.

```
#include <screen.h>
```

Inheritance diagram for screen::PIDPage:



### Public Member Functions

- [PIDPage](#) ([PID](#) &pid, std::string name, std::function< void(void)> onchange=[]() {})  
Create a [PIDPage](#).
- void [update](#) (bool was\_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

### 4.54.1 Detailed Description

[PIDPage](#) provides a way to tune a pid controller on the screen.

### 4.54.2 Constructor & Destructor Documentation

#### PIDPage()

```
screen::PIDPage::PIDPage (
    PID & pid,
    std::string name,
    std::function< void(void)> onchange = []() {})
```

Create a [PIDPage](#).

#### Parameters

<i>pid</i>	the pid controller we're changing
<i>name</i>	a name to recognize this pid controller if we've got multiple pid screens
<i>onchange</i>	a function that is called when a tuning parameter is changed. If you need to update stuff on that change register a handler here

### 4.54.3 Member Function Documentation

#### draw()

```
void screen::PIDPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number) [override], [virtual]
```

#### See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

#### update()

```
void screen::PIDPage::update (
    bool was_pressed,
    int x,
    int y) [override], [virtual]
```

#### See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

- `screen.h`
- `screen.cpp`

## 4.55 Pose2d Class Reference

```
#include <pose2d.h>
```

**Public Member Functions**

- constexpr [Pose2d](#) ()
- [Pose2d](#) (const [Translation2d](#) &translation, const [Rotation2d](#) &rotation)
- [Pose2d](#) (const double &x, const double &y, const [Rotation2d](#) &rotation)
- [Pose2d](#) (const double &x, const double &y, const double &radians)
- [Pose2d](#) (const [Translation2d](#) &translation, const double &radians)
- [Pose2d](#) (const Eigen::Vector3d &pose\_vector)
- [Translation2d](#) translation () const
- double x () const
- void setX (double x)
- double y () const
- void setY (double y)
- [Rotation2d](#) rotation () const
- void setRotationRad (double rotRad)
- void setRotationDeg (double rotDeg)
- bool operator== (const [Pose2d](#) other) const
- [Pose2d](#) operator\* (const double &scalar) const
- [Pose2d](#) operator/ (const double &scalar) const
- [Pose2d](#) operator+ (const [Transform2d](#) &transform) const
- [Transform2d](#) operator- (const [Pose2d](#) &other) const
- [Pose2d](#) relative\_to (const [Pose2d](#) &other) const
- [Pose2d](#) transform\_by (const [Transform2d](#) &transform) const
- [Pose2d](#) exp (const [Twist2d](#) &twist) const
- [Twist2d](#) log (const [Pose2d](#) &end\_pose) const

**Friends**

- std::ostream & operator<< (std::ostream &os, const [Pose2d](#) &pose)

**4.55.1 Detailed Description**

Class representing a pose in 2d space with x, y, and rotational components

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up  
+Theta is counterclockwise

**4.55.2 Constructor & Destructor Documentation****Pose2d()** [1/6]

```
Pose2d::Pose2d () [inline], [constexpr]
```

Default Constructor for [Pose2d](#)

**Pose2d()** [2/6]

```
Pose2d::Pose2d (
    const Translation2d & translation,
    const Rotation2d & rotation)
```

Constructs a pose with given translation and rotation components.

**Parameters**

<i>translation</i>	translational component.
<i>rotation</i>	rotational component.

**Pose2d()** [3/6]

```
Pose2d::Pose2d (  
    const double & x,  
    const double & y,  
    const Rotation2d & rotation)
```

Constructs a pose with given translation and rotation components.

**Parameters**

<i>x</i>	x component.
<i>y</i>	y component.
<i>rotation</i>	rotational component.

**Pose2d()** [4/6]

```
Pose2d::Pose2d (  
    const double & x,  
    const double & y,  
    const double & radians)
```

Constructs a pose with given translation and rotation components.

**Parameters**

<i>x</i>	x component.
<i>y</i>	y component.
<i>radians</i>	rotational component in radians.

**Pose2d()** [5/6]

```
Pose2d::Pose2d (  
    const Translation2d & translation,  
    const double & radians)
```

Constructs a pose with given translation and rotation components.

**Parameters**

<i>translation</i>	translational component.
<i>radians</i>	rotational component in radians.

**Pose2d()** [6/6]

```
Pose2d::Pose2d (  
    const Eigen::Vector3d & pose_vector)
```

Constructs a pose with given translation and rotation components.

## Parameters

<i>pose_vector</i>	vector of the form [x, y, theta].
--------------------	-----------------------------------

## 4.55.3 Member Function Documentation

**exp()**

```
Pose2d Pose2d::exp (
    const Twist2d & twist) const
```

Applies a twist (pose delta) to a pose by including first order dynamics of heading.

When applying a twist, imagine a constant angular velocity, the translational components must be rotated into the global frame at every point along the twist, simply adding the deltas does not do this, and using euler integration results in some error. This is the analytic solution that that problem.

Can also be thought of more simply as applying a twist as following an arc rather than a straight line.

See this document for more information on the pose exponential and its derivation. <https://file.tavsys.net/control/controls-engineering-in-frc.pdf#section.10.2>

## Parameters

<i>old_pose</i>	The pose to which the twist will be applied.
<i>twist</i>	The twist, represents a pose delta.

## Returns

new pose that has been moved forward according to the twist.

**log()**

```
Twist2d Pose2d::log (
    const Pose2d & end_pose) const
```

The inverse of the pose exponential.

Determines the twist required to go from this pose to the given end pose. suppose you have `Pose2d a`, `Twist2d twist` if `a.exp(twist) = b` then `a.log(b) = twist`

## Parameters

<i>end_pose</i>	the end pose to find the mapping to.
-----------------	--------------------------------------

## Returns

the twist required to go from this pose to the given end

**operator\*()**

```
Pose2d Pose2d::operator* (
    const double & scalar) const
```

Multiplies this pose by a scalar. Simply multiplies each component.

**Parameters**

<i>scalar</i>	the scalar value to multiply by.
---------------	----------------------------------

**operator+()**

```
Pose2d Pose2d::operator+ (
    const Transform2d & transform) const
```

Adds a transform to this pose. Transforms the pose in the pose's frame.

**Parameters**

<i>transform</i>	the change in pose.
------------------	---------------------

**operator-()**

```
Transform2d Pose2d::operator- (
    const Pose2d & other) const
```

Subtracts one pose from another to find the transform between them.

**Parameters**

<i>other</i>	the pose to subtract.
--------------	-----------------------

**operator/()**

```
Pose2d Pose2d::operator/ (
    const double & scalar) const
```

Divides this pose by a scalar. Simply divides each component.

**Parameters**

<i>scalar</i>	the scalar value to divide by.
---------------	--------------------------------

**operator==()**

```
bool Pose2d::operator== (
    const Pose2d other) const
```

Compares this to another pose.

## Parameters

<i>other</i>	the other pose to compare to.
--------------	-------------------------------

## Returns

true if each of the components are within 1e-9 of each other.

**relative\_to()**

```
Pose2d Pose2d::relative_to (  
    const Pose2d & other) const
```

Finds the pose equivalent to this pose relative to another arbitrary pose rather than the origin.

## Parameters

<i>other</i>	the pose representing the new origin.
--------------	---------------------------------------

## Returns

this pose relative to another pose.

**rotation()**

```
Rotation2d Pose2d::rotation () const
```

Returns the rotational component.

## Returns

the rotational component.

**setRotationDeg()**

```
void Pose2d::setRotationDeg (  
    double rotDeg)
```

sets the rotation value of the rotational component in Degrees

**setRotationRad()**

```
void Pose2d::setRotationRad (  
    double rotRad)
```

sets the rotation value of the rotational component in Radians



**setX()**

```
void Pose2d::setX (
    double x)
```

sets the x value of the translational component.

**setY()**

```
void Pose2d::setY (
    double y)
```

sets the y value of the translational component.

**transform\_by()**

```
Pose2d Pose2d::transform_by (
    const Transform2d & transform) const
```

Adds a transform to this pose. Simply adds each component.

**Parameters**

<i>transform</i>	the change in pose.
------------------	---------------------

**Returns**

the pose after being transformed.

**translation()**

```
Translation2d Pose2d::translation () const
```

Returns the translational component.

**Returns**

the translational component.

**x()**

```
double Pose2d::x () const
```

Returns the x value of the translational component.

**Returns**

the x value of the translational component.

**y()**

```
double Pose2d::y () const
```

Returns the y value of the translational component.

**Returns**

the y value of the translational component.

#### 4.55.4 Friends And Related Symbol Documentation

**operator<<**

```
std::ostream & operator<< (
    std::ostream & os,
    const Pose2d & pose) [friend]
```

Sends a pose to an output stream. Ex. `std::cout << pose;`

prints "Pose2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

The documentation for this class was generated from the following files:

- pose2d.h
- pose2d.cpp

## 4.56 PurePursuitCommand Class Reference

```
#include <drive_commands.h>
```

### Public Member Functions

- **PurePursuitCommand** ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, [PurePursuit::Path](#) path, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

#### 4.56.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the [TankDrive](#) class

#### 4.56.2 Constructor & Destructor Documentation

**PurePursuitCommand()**

```
PurePursuitCommand::PurePursuitCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    PurePursuit::Path path,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0)
```

Construct a Pure Pursuit AutoCommand

**Parameters**

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

**4.56.3 Member Function Documentation****on\_timeout()**

```
void PurePursuitCommand::on_timeout () [override]
```

Reset the drive system when it times out

**run()**

```
bool PurePursuitCommand::run () [override]
```

Direct call to [TankDrive::pure\\_pursuit](#)

The documentation for this class was generated from the following files:

- drive\_commands.h
- drive\_commands.cpp

**4.57 Rect Struct Reference**

```
#include <geometry.h>
```

**4.57.1 Detailed Description**

Describes a Rectangle with a minimum and maximum point

The documentation for this struct was generated from the following file:

- geometry.h

**4.58 robot\_specs\_t Struct Reference**

```
#include <robot_specs.h>
```

## Public Attributes

- double **robot\_radius**  
*if you were to draw a circle with this radius, the robot would be entirely contained within it*
- double **odom\_wheel\_diam**  
*the diameter of the wheels used for*
- double **odom\_gear\_ratio**  
*the ratio of the odometry wheel to the encoder reading odometry data*
- double **dist\_between\_wheels**  
*the distance between centers of the central drive wheels*
- double **drive\_correction\_cutoff**
- [Feedback](#) \* **drive\_feedback**  
*the default feedback for autonomous driving*
- [Feedback](#) \* **turn\_feedback**  
*the default feedback for autonomous turning*
- [PID::pid\\_config\\_t](#) **correction\_pid**  
*the pid controller to keep the robot driving in as straight a line as possible*

### 4.58.1 Detailed Description

Main robot characterization struct. This will be passed to all the major subsystems that require info about the robot. All distance measurements are in inches.

### 4.58.2 Member Data Documentation

#### **drive\_correction\_cutoff**

```
double robot_specs_t::drive_correction_cutoff
```

the distance at which to stop trying to turn towards the target. If we are less than this value, we can continue driving forward to minimize our distance but will not try to spin around to point directly at the target

The documentation for this struct was generated from the following file:

- robot\_specs.h

## 4.59 Rotation2d Class Reference

```
#include <rotation2d.h>
```

## Public Member Functions

- constexpr [Rotation2d](#) ()
- [Rotation2d](#) (const double &[radians](#))
- [Rotation2d](#) (const double &x, const double &y)
- [Rotation2d](#) (const [Translation2d](#) &translation)
- double [radians](#) () const
- void [setRad](#) (double radRot)
- double [degrees](#) () const
- void [setDeg](#) (double degRot)
- double [revolutions](#) () const
- double [f\\_cos](#) () const
- double [f\\_sin](#) () const
- double [f\\_tan](#) () const
- Eigen::Matrix2d [rotation\\_matrix](#) () const
- double [wrapped\\_radians\\_180](#) () const
- double [wrapped\\_degrees\\_180](#) () const
- double [wrapped\\_revolutions\\_180](#) () const
- double [wrapped\\_radians\\_360](#) () const
- double [wrapped\\_degrees\\_360](#) () const
- double [wrapped\\_revolutions\\_360](#) () const
- [Rotation2d](#) operator+ (const [Rotation2d](#) &other) const
- [Rotation2d](#) operator- (const [Rotation2d](#) &other) const
- [Rotation2d](#) operator- () const
- [Rotation2d](#) operator\* (const double &scalar) const
- [Rotation2d](#) operator/ (const double &scalar) const
- bool operator== (const [Rotation2d](#) &other) const

## Friends

- std::ostream & [operator<<](#) (std::ostream &os, const [Rotation2d](#) &rotation)

### 4.59.1 Detailed Description

Class representing a rotation in 2d space. Stores theta in radians, as well as cos and sin.

Internally this angle is stored continuously, however there are functions that return wrapped angles: "180" is from [-pi, pi), [-180, 180), [-0.5, 0.5) "360" is from [0, 2pi), [0, 360), [0, 1)

### 4.59.2 Constructor & Destructor Documentation

#### [Rotation2d](#)() [1/4]

```
Rotation2d::Rotation2d () [inline], [constexpr]
```

Default Constructor for [Rotation2d](#)

#### [Rotation2d](#)() [2/4]

```
Rotation2d::Rotation2d (
    const double & radians)
```

Constructs a rotation with the given value in radians.

## Parameters

<i>radians</i>	the value of the rotation in radians.
----------------	---------------------------------------

**Rotation2d()** [3/4]

```
Rotation2d::Rotation2d (
    const double & x,
    const double & y)
```

Constructs a rotation given x and y values. Does not have to be normalized. The angle from the x axis to the point.

[theta] = [atan2(y, x)]

## Parameters

<i>x</i>	the x value of the point
<i>y</i>	the y value of the point

**Rotation2d()** [4/4]

```
Rotation2d::Rotation2d (
    const Translation2d & translation)
```

Constructs a rotation given x and y values in the form of a [Translation2d](#). Does not have to be normalized. The angle from the x axis to the point.

[theta] = [atan2(y, x)]

## Parameters

<i>translation</i>	
--------------------	--

**4.59.3 Member Function Documentation****degrees()**

```
double Rotation2d::degrees () const
```

Returns the degree angle value.

## Returns

the degree angle value.

**f\_cos()**

```
double Rotation2d::f_cos () const
```

Returns the cosine of the angle value.

**Returns**

the cosine of the angle value

**f\_sin()**

```
double Rotation2d::f_sin () const
```

Returns the sine of the angle value.

**Returns**

the sine of the angle value.

**f\_tan()**

```
double Rotation2d::f_tan () const
```

Returns the tangent of the angle value.

**Returns**

the tangent of the angle value.

**operator\*()**

```
Rotation2d Rotation2d::operator* (
    const double & scalar) const
```

Multiplies this rotation by a scalar.

**Parameters**

<i>scalar</i>	the scalar value to multiply the rotation by.
---------------	---

**Returns**

the rotation multiplied by the scalar.

**operator+()**

```
Rotation2d Rotation2d::operator+ (
    const Rotation2d & other) const
```

Adds the values of two rotations using a rotation matrix

$$\begin{bmatrix} \text{new\_cos} \\ \text{new\_sin} \end{bmatrix} = \begin{bmatrix} \text{other.cos} & -\text{other.sin} \\ \text{other.sin} & \text{other.cos} \end{bmatrix} \begin{bmatrix} \text{cos} \\ \text{sin} \end{bmatrix}$$
  
$$\text{new\_value} = \text{atan2}(\text{new\_sin}, \text{new\_cos})$$

## Parameters

<i>other</i>	the other rotation to add to this rotation.
--------------	---

## Returns

the sum of the two rotations.

Adds the values of two rotations using a rotation matrix.

```
[new_cos] = [other.cos, -other.sin][cos] [new_sin] = [other.sin, other.cos][sin] new_value = atan2(new_sin, new_cos)
```

## Parameters

<i>other</i>	the other rotation to add to this rotation.
--------------	---

## Returns

the sum of the two rotations.

**operator-()** [1/2]

```
Rotation2d Rotation2d::operator- () const
```

Takes the inverse of this rotation by flipping it. Equivalent to adding 180 degrees.

## Returns

this inverse of the rotation.

Takes the inverse of this rotation by flipping it.

## Returns

this inverse of the rotation.

**operator-()** [2/2]

```
Rotation2d Rotation2d::operator- (
    const Rotation2d & other) const
```

Subtracts the values of two rotations.

## Parameters

<i>other</i>	the other rotation to subtract from this rotation.
--------------	--

## Returns

the difference between the two rotations.

**operator/()**

```
Rotation2d Rotation2d::operator/ (
    const double & scalar) const
```

Divides this rotation by a scalar.



**Parameters**

<i>scalar</i>	the scalar value to divide the rotation by.
---------------	---

**Returns**

the rotation divided by the scalar.

**operator==()**

```
bool Rotation2d::operator== (
    const Rotation2d & other) const
```

Compares two rotations. Returns true if their values are within 1e-9 radians of each other, to account for floating point error.

**Parameters**

<i>other</i>	the other rotation to compare to
--------------	----------------------------------

**Returns**

whether the values of the rotations are within 1e-9 radians of each other

**radians()**

```
double Rotation2d::radians () const
```

Returns the radian angle value.

**Returns**

the radian angle value.

**revolutions()**

```
double Rotation2d::revolutions () const
```

Returns the revolution angle value.

**Returns**

the revolution angle value.

**rotation\_matrix()**

```
Eigen::Matrix2d Rotation2d::rotation_matrix () const
```

Returns the rotation matrix equivalent to this rotation  $[\cos, -\sin]$   $R = [\sin, \cos]$

**Returns**

the rotation matrix equivalent to this rotation

**setDeg()**

```
void Rotation2d::setDeg (  
    double degRot)
```

sets the angle value in degrees

**setRad()**

```
void Rotation2d::setRad (  
    double radRot)
```

sets the angle value in radians

**wrapped\_degrees\_180()**

```
double Rotation2d::wrapped_degrees_180 () const
```

Returns the degree angle value, wrapped from  $[-180, 180)$ .

**Returns**

the degree angle value, wrapped from  $[-180, 180)$

**wrapped\_degrees\_360()**

```
double Rotation2d::wrapped_degrees_360 () const
```

Returns the degree angle value, wrapped from  $[0, 360)$ .

**Returns**

the degree angle value, wrapped from  $[0, 360)$

**wrapped\_radians\_180()**

```
double Rotation2d::wrapped_radians_180 () const
```

Returns the radian angle value, wrapped from  $[-\pi, \pi)$ .

**Returns**

the radian angle value, wrapped from  $[-\pi, \pi)$

**wrapped\_radians\_360()**

```
double Rotation2d::wrapped_radians_360 () const
```

Returns the radian angle value, wrapped from  $[0, 2\pi)$ .

**Returns**

the radian angle value, wrapped from  $[0, 2\pi)$

**wrapped\_revolutions\_180()**

```
double Rotation2d::wrapped_revolutions_180 () const
```

Returns the revolution angle value, wrapped from  $[-0.5, 0.5)$ .

**Returns**

the revolution angle value, wrapped from  $[-0.5, 0.5)$

**wrapped\_revolutions\_360()**

```
double Rotation2d::wrapped_revolutions_360 () const
```

Returns the revolution angle value, wrapped from  $[0, 1)$ .

**Returns**

the revolution angle value, wrapped from  $[0, 1)$

#### 4.59.4 Friends And Related Symbol Documentation

**operator<<**

```
std::ostream & operator<< (  
    std::ostream & os,  
    const Rotation2d & rotation) [friend]
```

Sends a rotation to an output stream. Ex. `std::cout << rotation;`

prints "Rotation2d[rad: (radians), deg: (degrees)]"

The documentation for this class was generated from the following files:

- rotation2d.h
- rotation2d.cpp

## 4.60 screen::ScreenData Struct Reference

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

### 4.60.1 Detailed Description

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

The documentation for this struct was generated from the following file:

- screen.cpp

## 4.61 Serializer Class Reference

Serializes Arbitrary data to a file on the SD Card.

```
#include <serializer.h>
```

### Public Member Functions

- **~Serializer ()**  
*Save and close upon destruction (bc of vex, this doesnt always get called when the program ends. To be sure, call save\_to\_disk)*
- **Serializer (const std::string &filename, bool flush\_always=true)**  
*create a [Serializer](#)*
- void **save\_to\_disk ()** const  
*saves current [Serializer](#) state to disk*
- void **set\_int** (const std::string &name, int i)  
*Setters - not saved until save\_to\_disk is called.*
- void **set\_bool** (const std::string &name, bool b)  
*sets a bool by the name of name to b. If flush\_always == true, this will save to the sd card*
- void **set\_double** (const std::string &name, double d)  
*sets a double by the name of name to d. If flush\_always == true, this will save to the sd card*
- void **set\_string** (const std::string &name, std::string str)  
*sets a string by the name of name to s. If flush\_always == true, this will save to the sd card*
- int **int\_or** (const std::string &name, int otherwise)  
*gets a value stored in the serializer. If not found, sets the value to otherwise*
- bool **bool\_or** (const std::string &name, bool otherwise)  
*gets a value stored in the serializer. If not, sets the value to otherwise*
- double **double\_or** (const std::string &name, double otherwise)  
*gets a value stored in the serializer. If not, sets the value to otherwise*
- std::string **string\_or** (const std::string &name, std::string otherwise)  
*gets a value stored in the serializer. If not, sets the value to otherwise*

### 4.61.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

#### 4.61.2 Constructor & Destructor Documentation

##### Serializer()

```
Serializer::Serializer (  
    const std::string & filename,  
    bool flush_always = true) [inline], [explicit]
```

create a [Serializer](#)

## Parameters

<i>filename</i>	the file to read from. If filename does not exist we will create that file
<i>flush_always</i>	If true, after every write flush to a file. If false, you are responsible for calling <code>save_to_disk</code>

## 4.61.3 Member Function Documentation

**bool\_or()**

```
bool Serializer::bool_or (
    const std::string & name,
    bool otherwise)
```

gets a value stored in the serializer. If not, sets the value to otherwise

## Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

## Returns

the value if found or otherwise

**double\_or()**

```
double Serializer::double_or (
    const std::string & name,
    double otherwise)
```

gets a value stored in the serializer. If not, sets the value to otherwise

## Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

## Returns

the value if found or otherwise

**int\_or()**

```
int Serializer::int_or (
    const std::string & name,
    int otherwise)
```

gets a value stored in the serializer. If not found, sets the value to otherwise

Getters Return value if it exists in the serializer

**Parameters**

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

**Returns**

the value if found or otherwise

**save\_to\_disk()**

```
void Serializer::save_to_disk () const
```

saves current [Serializer](#) state to disk

forms data bytes then saves to filename this was opened with

**set\_bool()**

```
void Serializer::set_bool (
    const std::string & name,
    bool b)
```

sets a bool by the name of name to b. If flush\_always == true, this will save to the sd card

**Parameters**

<i>name</i>	name of bool
<i>b</i>	value of bool

**set\_double()**

```
void Serializer::set_double (
    const std::string & name,
    double d)
```

sets a double by the name of name to d. If flush\_always == true, this will save to the sd card

**Parameters**

<i>name</i>	name of double
<i>d</i>	value of double

**set\_int()**

```
void Serializer::set_int (
    const std::string & name,
    int i)
```

Setters - not saved until save\_to\_disk is called.

sets an integer by the name of name to i. If flush\_always == true, this will save to the sd card

## Parameters

<i>name</i>	name of integer
<i>i</i>	value of integer

**set\_string()**

```
void Serializer::set_string (
    const std::string & name,
    std::string str)
```

sets a string by the name of name to s. If flush\_always == true, this will save to the sd card

## Parameters

<i>name</i>	name of string
<i>i</i>	value of string

**string\_or()**

```
std::string Serializer::string_or (
    const std::string & name,
    std::string otherwise)
```

gets a value stored in the serializer. If not, sets the value to otherwise

## Parameters

<i>name</i>	name of value
<i>otherwise</i>	value if the name is not specified

## Returns

the value if found or otherwise

The documentation for this class was generated from the following files:

- serializer.h
- serializer.cpp

**4.62 screen::SliderWidget Class Reference**

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

```
#include <screen.h>
```



## Public Member Functions

- **SliderWidget** (double &val, double low, double high, **Rect** rect, std::string name)  
*Creates a slider widget.*
- bool **update** (bool was\_pressed, int x, int y)  
*responds to user input*
- void **draw** (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number)  
*Page::draws the slide to the screen*

### 4.62.1 Detailed Description

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

### 4.62.2 Constructor & Destructor Documentation

#### SliderWidget()

```
screen::SliderWidget::SliderWidget (
    double & val,
    double low,
    double high,
    Rect rect,
    std::string name) [inline]
```

Creates a slider widget.

#### Parameters

<i>val</i>	reference to the value to modify
<i>low</i>	minimum value to go to
<i>high</i>	maximum value to go to
<i>rect</i>	rect to draw it
<i>name</i>	name of the value

### 4.62.3 Member Function Documentation

#### update()

```
bool screen::SliderWidget::update (
    bool was_pressed,
    int x,
    int y)
```

responds to user input

#### Parameters

<i>was_pressed</i>	if the screen is pressed
<i>x</i>	x position if the screen was pressed
<i>y</i>	y position if the screen was pressed

**Returns**

true if the value updated

The documentation for this class was generated from the following files:

- screen.h
- screen.cpp

## 4.63 SpinRPMCommand Class Reference

```
#include <flywheel_commands.h>
```

**Public Member Functions**

- [SpinRPMCommand](#) ([Flywheel](#) &flywheel, int rpm)
- bool [run](#) () override

### 4.63.1 Detailed Description

File: [flywheel\\_commands.h](#) Desc: [insert meaningful desc] AutoCommand wrapper class for the spin\_rpm function in the [Flywheel](#) class

### 4.63.2 Constructor & Destructor Documentation

**SpinRPMCommand()**

```
SpinRPMCommand::SpinRPMCommand (  
    Flywheel & flywheel,  
    int rpm)
```

Construct a SpinRPM Command

**Parameters**

<i>flywheel</i>	the flywheel sys to command
<i>rpm</i>	the rpm that we should spin at

File: flywheel\_commands.cpp Desc: [insert meaningful desc]

### 4.63.3 Member Function Documentation

#### run()

```
bool SpinRPMCommand::run () [override]
```

Run spin\_manual Overrides run from AutoCommand

#### Returns

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- flywheel\_commands.h
- flywheel\_commands.cpp

## 4.64 PurePursuit::spline Struct Reference

```
#include <pure_pursuit.h>
```

### 4.64.1 Detailed Description

Represents a piece of a cubic spline with  $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$  The `x_start` and `x_end` shows where the equation is valid.

The documentation for this struct was generated from the following file:

- pure\_pursuit.h

## 4.65 StateMachine< System, IDType, Message, delay\_ms, do\_log >::State Struct Reference

```
#include <state_machine.h>
```

### 4.65.1 Detailed Description

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
struct StateMachine< System, IDType, Message, delay_ms, do_log >::State
```

Abstract class that all states for this machine must inherit from States MUST override `respond()` and `id()` in order to function correctly (the compiler won't have it any other way)

The documentation for this struct was generated from the following file:

- state\_machine.h

## 4.66 StateMachine< System, IDType, Message, delay\_ms, do\_log > Class Template Reference

[State Machine :\)\)\)\)\)\)](#) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through `current_state` and `send_message`.

```
#include <state_machine.h>
```

### Classes

- class [MaybeMessage](#)

[MaybeMessage](#) a message of Message type or nothing [MaybeMessage](#) `m = {};` // empty [MaybeMessage](#) `m = Message::EnumField1.`

- struct [State](#)

### Public Member Functions

- [StateMachine](#) ([State](#) \*initial)

Construct a state machine and immediatly start running it.

- IDType [current\\_state](#) () const

retrieve the current state of the state machine. This is safe to call from external threads

- void [send\\_message](#) (Message msg)

send a message to the state machine from outside

### 4.66.1 Detailed Description

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
class StateMachine< System, IDType, Message, delay_ms, do_log >
```

[State Machine :\)\)\)\)\)\)](#) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through `current_state` and `send_message`.

Designwise: the System class should hold onto any motors, feedback controllers, etc that are persistent in the system States themselves should hold any data that *only* that state needs. For example if a state should be exited after a certain amount of time, it should hold a timer rather than the System holding that timer. (see Junder from 2024 for an example of this design)

### Template Parameters

<i>System</i>	The system that this is the base class of <code>class Thing : public StateMachine&lt;Thing&gt; @tparam IDType The ID enum that recognizes states. Hint hint, use anenum class`</code>
<i>Message</i>	the message enum that a state or an outside can send and that states respond to
<i>delay_ms</i>	the delay to wait between each state processing to allow other threads to work
<i>do_log</i>	true if you want print statements describing incoming messages and current states. If true, it is expected that IDType and Message have a function called <code>to_string</code> that takes them as its only parameter and returns a <code>std::string</code>

### 4.66.2 Constructor & Destructor Documentation

#### StateMachine()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
StateMachine< System, IDType, Message, delay_ms, do_log >::StateMachine (
    State * initial) [inline]
```

Construct a state machine and immediatly start running it.

##### Parameters

<i>initial</i>	the state that the machine will begin in
----------------	--

### 4.66.3 Member Function Documentation

#### current\_state()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
IDType StateMachine< System, IDType, Message, delay_ms, do_log >::current_state () const
[inline]
```

retrieve the current state of the state machine. This is safe to call from external threads

##### Returns

the current state

#### send\_message()

```
template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log =
false>
void StateMachine< System, IDType, Message, delay_ms, do_log >::send_message (
    Message msg) [inline]
```

send a message to the state machine from outside

##### Parameters

<i>msg</i>	the message to send This is safe to call from external threads
------------	--

The documentation for this class was generated from the following file:

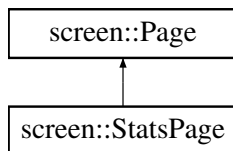
- state\_machine.h

## 4.67 screen::StatsPage Class Reference

Draws motor stats and battery stats to the screen.

```
#include <screen.h>
```

Inheritance diagram for screen::StatsPage:



### Public Member Functions

- [StatsPage](#) (std::map< std::string, vex::motor & > motors)  
*Creates a stats page.*
- void [update](#) (bool was\_pressed, int x, int y) override
- void [draw](#) (vex::brain::lcd &, bool first\_draw, unsigned int frame\_number) override

#### 4.67.1 Detailed Description

Draws motor stats and battery stats to the screen.

#### 4.67.2 Constructor & Destructor Documentation

##### StatsPage()

```
screen::StatsPage::StatsPage (
    std::map< std::string, vex::motor & > motors)
```

Creates a stats page.

##### Parameters

<i>motors</i>	a map of string to motor that we want to draw on this page
---------------	--

#### 4.67.3 Member Function Documentation

##### draw()

```
void screen::StatsPage::draw (
    vex::brain::lcd & scr,
    bool first_draw,
    unsigned int frame_number) [override], [virtual]
```

##### See also

[Page::draw](#)

Reimplemented from [screen::Page](#).

## update()

```
void screen::StatsPage::update (
    bool was_pressed,
    int x,
    int y) [override], [virtual]
```

### See also

[Page::update](#)

Reimplemented from [screen::Page](#).

The documentation for this class was generated from the following files:

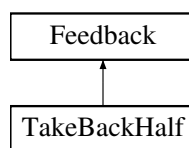
- screen.h
- screen.cpp

## 4.68 TakeBackHalf Class Reference

A velocity controller.

```
#include <take_back_half.h>
```

Inheritance diagram for TakeBackHalf:



### Public Member Functions

- void [init](#) (double start\_pt, double set\_pt)
- double [update](#) (double val) override
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override

### Public Attributes

- double **TBH\_gain**  
*tuned parameter*

#### 4.68.1 Detailed Description

A velocity controller.

#### Warning

If you try to use this as a position controller, it will fail.

## 4.68.2 Member Function Documentation

### get()

```
double TakeBackHalf::get () [override], [virtual]
```

#### Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### init()

```
void TakeBackHalf::init (  
    double start_pt,  
    double set_pt) [virtual]
```

Initialize the feedback controller for a movement

#### Parameters

<i>start_pt</i>	the current sensor value
<i>set_pt</i>	where the sensor value should be
<i>start_vel</i>	Movement starting velocity (IGNORED)
<i>end_vel</i>	Movement ending velocity (IGNORED)

Implements [Feedback](#).

### is\_on\_target()

```
bool TakeBackHalf::is_on_target () [override], [virtual]
```

#### Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

### set\_limits()

```
void TakeBackHalf::set_limits (  
    double lower,  
    double upper) [override], [virtual]
```

Clamp the upper and lower limits of the output. If both are 0, no limits should be applied.



**Parameters**

<i>lower</i>	Upper limit
<i>upper</i>	Lower limit

Implements [Feedback](#).

**update()**

```
double TakeBackHalf::update (  
    double val) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value

**Parameters**

<i>val</i>	value from the sensor
------------	-----------------------

**Returns**

feedback loop result

Implements [Feedback](#).

The documentation for this class was generated from the following files:

- take\_back\_half.h
- take\_back\_half.cpp

## 4.69 TankDrive Class Reference

```
#include <tank_drive.h>
```

**Public Types**

- enum class [BrakeType](#) { [None](#) , [ZeroVelocity](#) , [Smart](#) , [TurnOnly](#) }

**Public Member Functions**

- [TankDrive](#) (motor\_group &left\_motors, motor\_group &right\_motors, [robot\\_specs\\_t](#) &config, [OdometryBase](#) \*odom=NULL)
- void [stop](#) ()
- [Pose2d](#) [get\\_position](#) ()
- void [drive\\_tank](#) (double left, double right, int power=1, [BrakeType](#) bt=[BrakeType::None](#))
- void [drive\\_tank\\_raw](#) (double left, double right)
- void [drive\\_arcade](#) (double forward\_back, double left\_right, int power=1, [BrakeType](#) bt=[BrakeType::None](#))
- bool [drive\\_forward](#) (double inches, directionType dir, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [drive\\_forward](#) (double inches, directionType dir, double max\_speed=1, double end\_speed=0)
- bool [turn\\_degrees](#) (double degrees, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [turn\\_degrees](#) (double degrees, double max\_speed=1, double end\_speed=0)
- bool [drive\\_to\\_point](#) (double x, double y, vex::directionType dir, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [drive\\_to\\_point](#) (double x, double y, vex::directionType dir, double max\_speed=1, double end\_speed=0)
- bool [turn\\_to\\_heading](#) (double heading\_deg, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)
- bool [turn\\_to\\_heading](#) (double heading\_deg, double max\_speed=1, double end\_speed=0)
- void [reset\\_auto](#) ()
- bool [pure\\_pursuit](#) ([PurePursuit::Path](#) path, directionType dir, [Feedback](#) &feedback, double max\_speed=1, double end\_speed=0)

**Static Public Member Functions**

- static double [modify\\_inputs](#) (double input, int power=2)

**4.69.1 Detailed Description**

[TankDrive](#) is a class to run a tank drive system. A tank drive system, sometimes called differential drive, has a motor (or group of synchronized motors) on the left and right side

**4.69.2 Member Enumeration Documentation****BrakeType**

```
enum class TankDrive::BrakeType [strong]
```

**Enumerator**

None	just send 0 volts to the motors
ZeroVelocity	try to bring the robot to rest. But don't try to hold position
Smart	bring the robot to rest and once it's stopped, try to hold that position

**4.69.3 Constructor & Destructor Documentation****TankDrive()**

```
TankDrive::TankDrive (
    motor_group & left_motors,
    motor_group & right_motors,
    robot_specs_t & config,
    OdometryBase * odom = NULL)
```

Create the

## Parameters

<i>left_motors</i>	left side drive motors
<i>right_motors</i>	right side drive motors
<i>config</i>	the configuration specification defining physical dimensions about the robot. See <a href="#">robot_specs_t</a> for more info
<i>odom</i>	an odometry system to track position and rotation. this is necessary to execute autonomous paths

## 4.69.4 Member Function Documentation

**drive\_arcade()**

```
void TankDrive::drive_arcade (
    double forward_back,
    double left_right,
    int power = 1,
    BrakeType bt = BrakeType::None)
```

Drive the robot using arcade style controls. forward\_back controls the linear motion, left\_right controls the turning.

forward\_back and left\_right are in "percent": -1.0 -> 1.0

## Parameters

<i>forward_back</i>	the percent to move forward or backward
<i>left_right</i>	the percent to turn left or right
<i>power</i>	modifies the input velocities $\text{left}^{\text{power}}$ , $\text{right}^{\text{power}}$
<i>bt</i>	breaktype. What to do if the driver lets go of the sticks

Drive the robot using arcade style controls. forward\_back controls the linear motion, left\_right controls the turning.

left\_motors and right\_motors are in "percent": -1.0 -> 1.0

**drive\_forward()** [1/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    double max_speed = 1,
    double end_speed = 0)
```

Autonomously drive the robot forward a certain distance

## Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously drive the robot forward a certain distance

## Parameters

<i>inches</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we have finished driving to our point

**drive\_forward()** [2/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0)
```

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

## Parameters

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true when we have reached our target distance

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

## Parameters

<i>inches</i>	the distance to drive forward
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**drive\_tank()**

```
void TankDrive::drive_tank (
    double left,
    double right,
    int power = 1,
    BrakeType bt = BrakeType::None)
```

Drive the robot using differential style controls. `left_motors` controls the left motors, `right_motors` controls the right motors.

`left_motors` and `right_motors` are in "percent": -1.0 -> 1.0

**Parameters**

<i>left</i>	the percent to run the left motors
<i>right</i>	the percent to run the right motors
<i>power</i>	modifies the input velocities $\text{left}^{\text{power}}$ , $\text{right}^{\text{power}}$
<i>bt</i>	breaktype. What to do if the driver lets go of the sticks

**drive\_tank\_raw()**

```
void TankDrive::drive_tank_raw (
    double left,
    double right)
```

Drive the robot raw-ly

**Parameters**

<i>left</i>	the percent to run the left motors (-1, 1)
<i>right</i>	the percent to run the right motors (-1, 1)

**drive\_to\_point()** [1/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    double max_speed = 1,
    double end_speed = 0)
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the `drive_sys`

Returns whether or not the robot has reached it's destination.

**Parameters**

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target

<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive\_sys

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

#### Returns

true if we have reached our target point

#### drive\_to\_point() [2/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0)
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

#### Parameters

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

**Parameters**

<i>x</i>	the x position of the target
<i>y</i>	the y position of the target
<i>dir</i>	the direction we want to travel forward and backward
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we have reached our target point

**get\_position()**

```
Pose2d TankDrive::get_position ()
```

Returns the Robot position as a [Pose2d](#)

**modify\_inputs()**

```
double TankDrive::modify_inputs (
    double input,
    int power = 2) [static]
```

Create a curve for the inputs, so that drivers have more control at lower speeds. Curves are exponential, with the default being squaring the inputs.

**Parameters**

<i>input</i>	the input before modification
<i>power</i>	the power to raise input to

**Returns**

$\text{input}^{\text{power}}$  (accounts for negative inputs and odd numbered powers)

Modify the inputs from the controller by squaring / cubing, etc Allows for better control of the robot at slower speeds

**Parameters**

<i>input</i>	the input signal -1 -> 1
<i>power</i>	the power to raise the signal to

**Returns**

$\text{input}^{\text{power}}$  accounting for any sign issues that would arise with this naive solution

**pure\_pursuit()**

```
bool TankDrive::pure_pursuit (
    PurePursuit::Path path,
    directionType dir,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0)
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

**Parameters**

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

True when the path is complete

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

**Parameters**

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

**Returns**

True when the path is complete

**reset\_auto()**

```
void TankDrive::reset_auto ()
```

Reset the initialization for autonomous drive functions

**stop()**

```
void TankDrive::stop ()
```

Stops rotation of all the motors using their "brake mode"



**turn\_degrees()** [1/2]

```
bool TankDrive::turn_degrees (
    double degrees,
    double max_speed = 1,
    double end_speed = 0)
```

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

**Parameters**

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the default turning feedback of the drive system.

**Parameters**

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we turned to target number of degrees

**turn\_degrees()** [2/2]

```
bool TankDrive::turn_degrees (
    double degrees,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0)
```

Autonomously turn the robot X degrees counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses [PID](#) + Feedforward for it's control.

**Parameters**

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent\_speed (-1.0 -> 1.0)

Uses the specified feedback for it's control.

## Parameters

<i>degrees</i>	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we have turned our target number of degrees

**turn\_to\_heading()** [1/2]

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    double max_speed = 1,
    double end_speed = 0)
```

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the default turn feedback of the drive system

## Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the default turn feedback of the drive system

## Parameters

<i>heading_deg</i>	the heading to which we will turn
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

## Returns

true if we have reached our target heading

**turn\_to\_heading()** [2/2]

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    Feedback & feedback,
    double max_speed = 1,
    double end_speed = 0)
```

Turn the robot in place to an exact heading relative to the field. 0 is forward.

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward.

**Parameters**

<i>heading_deg</i>	the heading to which we will turn
<i>feedback</i>	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
<i>max_speed</i>	the maximum percentage of robot speed at which the robot will travel. 1 = full power
<i>end_speed</i>	the movement profile will attempt to reach this velocity by its completion

**Returns**

true if we have reached our target heading

The documentation for this class was generated from the following files:

- tank\_drive.h
- tank\_drive.cpp

**4.70 tracking\_wheel\_cfg\_t Struct Reference**

```
#include <odometry_nwheel.h>
```

**Public Attributes**

- double *x*
- double *y*
- double *theta\_rad*
- double *radius*

**4.70.1 Detailed Description****OdometryNWheel**

This class handles the code for an N-pod odometry setup, where there are N <WHEELS> free spinning omni wheels (dead wheels) placed in any known configuration on the robot.

Example of a possible wheel configuration:

```
+Y ----- ^ | === ||||| O ||||| === | | ----- | +-----> + X
```

Where O is the center of rotation. The robot will monitor the changes in rotation of these wheels, use this to calculate a pose delta, then integrate the deltas over time to determine the robot's position.

This is a "set and forget" class, meaning once the object is created, the robot will immediately begin tracking it's movement in the background.

<https://rit.enterprise.slack.com/files/U04112Y5RB6/F080M01KPA5/predictperpindiculars2.pdf> 2024-2025 Notebook: Entries/Software Entries/Localization/N-Pod Odometry

**Author**

Jack Cammarata, Richie Sommers

**Date**

Nov 14 2024 [tracking\\_wheel\\_cfg\\_t](#) holds all the specifications for a single tracking wheel. The units for x, y, and radius will determine the units of the position estimate.

**4.70.2 Member Data Documentation****radius**

```
double tracking_wheel_cfg_t::radius
```

radius of the wheel

**theta\_rad**

```
double tracking_wheel_cfg_t::theta_rad
```

angle between wheel direction and x axis in the robot frame

**x**

```
double tracking_wheel_cfg_t::x
```

x position of the center of the wheel

**y**

```
double tracking_wheel_cfg_t::y
```

y position of the center of the wheel

The documentation for this struct was generated from the following file:

- `odometry_nwheel.h`

**4.71 Transform2d Class Reference**

```
#include <transform2d.h>
```

## Public Member Functions

- constexpr [Transform2d](#) ()
- [Transform2d](#) (const [Translation2d](#) &translation, const [Rotation2d](#) &rotation)
- [Transform2d](#) (const double &x, const double &y, const [Rotation2d](#) &rotation)
- [Transform2d](#) (const double &x, const double &y, const double &radians)
- [Transform2d](#) (const [Translation2d](#) &translation, const double &radians)
- [Transform2d](#) (const Eigen::Vector3d &transform\_vector)
- [Transform2d](#) (const [Pose2d](#) &start, const [Pose2d](#) &end)
- [Translation2d](#) translation () const
- double x () const
- double y () const
- [Rotation2d](#) rotation () const
- [Transform2d](#) inverse () const
- [Transform2d](#) operator\* (const double &scalar) const
- [Transform2d](#) operator/ (const double &scalar) const
- [Transform2d](#) operator- () const
- bool operator== (const [Transform2d](#) &other) const

## Friends

- std::ostream & operator<< (std::ostream &os, const [Transform2d](#) &transform)

### 4.71.1 Detailed Description

Class representing a transformation of a pose2d, or a linear difference between the components of poses.

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up +Theta is counterclockwise

### 4.71.2 Constructor & Destructor Documentation

#### [Transform2d\(\)](#) [1/7]

```
Transform2d::Transform2d () [constexpr]
```

Default Constructor for [Transform2d](#)

#### [Transform2d\(\)](#) [2/7]

```
Transform2d::Transform2d (
    const Translation2d & translation,
    const Rotation2d & rotation)
```

Constructs a transform given translation and rotation components.

#### Parameters

<i>translation</i>	the translational component of the transform.
<i>rotation</i>	the rotational component of the transform.

**Transform2d()** [3/7]

```
Transform2d::Transform2d (  
    const double & x,  
    const double & y,  
    const Rotation2d & rotation)
```

Constructs a transform given translation and rotation components.

**Parameters**

<i>x</i>	the x component of the transform.
<i>y</i>	the y component of the transform.
<i>rotation</i>	the rotational component of the transform.

**Transform2d()** [4/7]

```
Transform2d::Transform2d (  
    const double & x,  
    const double & y,  
    const double & radians)
```

Constructs a transform given translation and rotation components.

**Parameters**

<i>x</i>	the x component of the transform.
<i>y</i>	the y component of the transform.
<i>radians</i>	the rotational component of the transform in radians.

**Transform2d()** [5/7]

```
Transform2d::Transform2d (  
    const Translation2d & translation,  
    const double & radians)
```

Constructs a transform given translation and rotation components.

**Parameters**

<i>translation</i>	the translational component of the transform.
<i>radians</i>	the rotational component of the transform in radians.

**Transform2d()** [6/7]

```
Transform2d::Transform2d (  
    const Eigen::Vector3d & transform_vector)
```

Constructs a transform given translation and rotation components given as a vector.

**Parameters**

<i>transform_vector</i>	vector of the form [x, y, theta]
-------------------------	----------------------------------

**Transform2d()** [7/7]

```
Transform2d::Transform2d (  
    const Pose2d & start,  
    const Pose2d & end)
```

Constructs a transform given translation and rotation components.

## Parameters

<i>translation</i>	the translational component of the transform.
<i>rotation</i>	the rotational component of the transform.

## 4.71.3 Member Function Documentation

**inverse()**

```
Transform2d Transform2d::inverse () const
```

Inverts the transform.

## Returns

the inverted transform.

**operator\*()**

```
Transform2d Transform2d::operator* (  
    const double & scalar) const
```

Multiplies this transform by a scalar.

## Parameters

<i>scalar</i>	the scalar to multiply this transform by.
---------------	---

**operator-()**

```
Transform2d Transform2d::operator- () const
```

Inverts the transform.

## Returns

the inverted transform.

**operator/()**

```
Transform2d Transform2d::operator/ (  
    const double & scalar) const
```

Divides this transform by a scalar.



**Parameters**

<i>scalar</i>	the scalar to divide this transform by.
---------------	---

**operator==()**

```
bool Transform2d::operator== (
    const Transform2d & other) const
```

Compares this to another transform.

**Parameters**

<i>other</i>	the other transform to compare to.
--------------	------------------------------------

**Returns**

true if the components are within 1e-9 of each other.

**rotation()**

```
Rotation2d Transform2d::rotation () const
```

Returns the rotational component of the transform.

**Returns**

the rotational component of the transform.

**translation()**

```
Translation2d Transform2d::translation () const
```

Returns the translational component of the transform.

**Returns**

the translational component of the transform.

**x()**

```
double Transform2d::x () const
```

Returns the x component of the transform.

**Returns**

the x component of the transform.

**y()**

```
double Transform2d::y () const
```

Returns the y component of the transform.

**Returns**

the y component of the transform.

#### 4.71.4 Friends And Related Symbol Documentation

**operator<<**

```
std::ostream & operator<< (
    std::ostream & os,
    const Transform2d & transform) [friend]
```

Sends a transform to an output stream. Ex. `std::cout << transform;`

prints "Transform2d[dx: (value), dy: (value), drad: (radians), ddeg: (degrees)]"

Sends a transform to an output stream. Ex. `std::cout << transform;`

prints "Transform2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

The documentation for this class was generated from the following files:

- transform2d.h
- transform2d.cpp

## 4.72 Translation2d Class Reference

```
#include <translation2d.h>
```

### Public Member Functions

- constexpr [Translation2d](#) ()
- [Translation2d](#) (const double &x, const double &y)
- [Translation2d](#) (const Eigen::Vector2d &vector)
- [Translation2d](#) (const double &r, const [Rotation2d](#) &theta)
- double [x](#) () const
- void [setX](#) (double x)
- double [y](#) () const
- void [setY](#) (double y)
- [Rotation2d](#) [theta](#) () const
- Eigen::Vector2d [as\\_vector](#) () const
- double [get\\_mag](#) () const
- double [norm](#) () const
- [Translation2d](#) [normalize](#) () const
- double [distance](#) (const [Translation2d](#) &other) const
- [Translation2d](#) [rotate\\_by](#) (const [Rotation2d](#) &rotation) const
- [Translation2d](#) [rotate\\_around](#) (const [Translation2d](#) &other, const [Rotation2d](#) &rotation) const
- [Translation2d](#) [operator+](#) (const [Translation2d](#) &other) const
- [Translation2d](#) [operator-](#) (const [Translation2d](#) &other) const
- [Translation2d](#) [operator-](#) () const
- [Translation2d](#) [operator\\*](#) (const double &scalar) const
- [Translation2d](#) [operator/](#) (const double &scalar) const
- double [operator\\*](#) (const [Translation2d](#) &other) const
- bool [operator==](#) (const [Translation2d](#) &other) const

## Friends

- `std::ostream & operator<< (std::ostream &os, const Translation2d &translation)`

### 4.72.1 Detailed Description

Class representing a point in 2d space with x and y coordinates.

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up  
+Theta is counterclockwise

### 4.72.2 Constructor & Destructor Documentation

#### Translation2d() [1/4]

```
Translation2d::Translation2d () [inline], [constexpr]
```

Default Constructor for [Translation2d](#)

#### Translation2d() [2/4]

```
Translation2d::Translation2d (  
    const double & x,  
    const double & y)
```

Constructs a [Translation2d](#) with the given x and y values.

##### Parameters

<i>x</i>	The x component of the translation.
<i>y</i>	The y component of the translation.

#### Translation2d() [3/4]

```
Translation2d::Translation2d (  
    const Eigen::Vector2d & vector)
```

Constructs a [Translation2d](#) with the values from the given vector.

##### Parameters

<i>vector</i>	The vector whose values will be used.
---------------	---------------------------------------

#### Translation2d() [4/4]

```
Translation2d::Translation2d (  
    const double & r,  
    const Rotation2d & theta)
```

Constructs a [Translation2d](#) given polar coordinates of the form (r, theta).

## Parameters

<i>r</i>	The radius (magnitude) of the vector.
<i>theta</i>	The angle (direction) of the vector.

### 4.72.3 Member Function Documentation

**as\_vector()**

```
Eigen::Vector2d Translation2d::as_vector () const
```

Returns the vector as an Eigen::Vector2d.

**Returns**

Eigen::Vector2d with the same values as the translation.

**distance()**

```
double Translation2d::distance (  
    const Translation2d & other) const
```

Returns the distance between two translations.

**Returns**

the distance between two translations.

**get\_mag()**

```
double Translation2d::get_mag () const
```

**Returns**

the magnitude of the directions if it were a vector

**norm()**

```
double Translation2d::norm () const
```

Returns the norm/radius/magnitude/distance from origin.

**Returns**

the norm of the translation.

**normalize()**

```
Translation2d Translation2d::normalize () const
```

Changes the translation so that it has a vector magnitude of 1

**Returns**

the norm of the translation.

Returns a translation so that it has a vector magnitude of 1

**Returns**

the norm of the translation.

**operator\*() [1/2]**

```
Translation2d Translation2d::operator* (
    const double & scalar) const
```

Returns this translation multiplied by a scalar.

$[x] = [x] * [scalar]$   $[y] = [y] * [scalar]$

**Parameters**

<i>scalar</i>	the scalar to multiply by.
---------------	----------------------------

**Returns**

this translation multiplied by a scalar.

**operator\*() [2/2]**

```
double Translation2d::operator* (
    const Translation2d & other) const
```

Returns the dot product of two translations.

$[scalar] = [x][otherx] + [y][othery]$

**Parameters**

<i>other</i>	the other translation to find the dot product with.
--------------	---

**Returns**

the scalar valued dot product.

**operator+()**

```
Translation2d Translation2d::operator+ (
    const Translation2d & other) const
```

Returns the sum of two translations.

$[x] = [x] + [otherx]; [y] = [y] + [othery];$

## Parameters

<i>other</i>	the other translation to add to this translation.
--------------	---

## Returns

the sum of the two translations.

**operator-()** [1/2]

```
Translation2d Translation2d::operator- () const
```

Returns the inverse of this translation. Equivalent to flipping the vector across the origin.

$$[x] = [-x] \quad [y] = [-y]$$

## Returns

the inverse of this translation.

**operator-()** [2/2]

```
Translation2d Translation2d::operator- (
    const Translation2d & other) const
```

Returns the difference of two translations.

$$[x] = [x] - [\text{other}x] \quad [y] = [y] - [\text{other}y]$$

## Parameters

<i>other</i>	the translation to subtract from this translation.
--------------	--

## Returns

the difference of the two translations.

**operator/()**

```
Translation2d Translation2d::operator/ (
    const double & scalar) const
```

Returns this translation divided by a scalar.

$$[x] = [x] / [\text{scalar}] \quad [y] = [y] / [\text{scalar}]$$

**Parameters**

<i>scalar</i>	the scalar to divide by.
---------------	--------------------------

**Returns**

this translation divided by a scalar.

**operator==()**

```
bool Translation2d::operator== (
    const Translation2d & other) const
```

Compares two translations. Returns true if their components are each within 1e-9, to account for floating point error.

**Parameters**

<i>other</i>	the translation to compare to.
--------------	--------------------------------

**Returns**

whether the two translations are equal.

**rotate\_around()**

```
Translation2d Translation2d::rotate_around (
    const Translation2d & other,
    const Rotation2d & rotation) const
```

Applies a rotation to this translation around another given point.

$[x] = [\cos, -\sin][x - \text{otherx}] + [\text{otherx}]$   $[y] = [\sin, \cos][y - \text{othersy}] + [\text{othersy}]$

**Parameters**

<i>other</i>	the center of rotation.
<i>rotation</i>	the angle amount the translation will be rotated.

**Returns**

the translation that has been rotated.

**rotate\_by()**

```
Translation2d Translation2d::rotate_by (
    const Rotation2d & rotation) const
```

Applies a rotation to this translation around the origin.

Equivalent to multiplying a vector by a rotation matrix:  $x = [\cos, -\sin][x]$   $y = [\sin, \cos][y]$

**Parameters**

<i>rotation</i>	the angle amount the translation will be rotated.
-----------------	---

**Returns**

the new translation that has been rotated around the origin.

**setX()**

```
void Translation2d::setX (  
    double x)
```

Sets the x value of the translation.

**setY()**

```
void Translation2d::setY (  
    double y)
```

Sets the y value of the translation.

**theta()**

```
Rotation2d Translation2d::theta () const
```

Returns the angle of the translation.

**Returns**

the angle of the translation.

**x()**

```
double Translation2d::x () const
```

Returns the x value of the translation.

**Returns**

the x value of the translation.



**y()**

```
double Translation2d::y () const
```

Returns the y value of the translation.

**Returns**

the y value of the translation.

#### 4.72.4 Friends And Related Symbol Documentation

**operator<<**

```
std::ostream & operator<< (  
    std::ostream & os,  
    const Translation2d & translation) [friend]
```

Sends a translation to an output stream. Ex. `std::cout << translation;`

prints "Translation2d[x: (value), y: (value)]"

Sends a translation to an output stream. Ex. `std::cout << translation;`

prints "Translation2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

The documentation for this class was generated from the following files:

- translation2d.h
- translation2d.cpp

#### 4.73 TrapezoidProfile Class Reference

```
#include <trapezoid_profile.h>
```

##### Public Member Functions

- [TrapezoidProfile](#) (double max\_v, double accel)  
*Construct a new Trapezoid Profile object.*
- [motion\\_t calculate](#) (double time\_s)  
*Run the trapezoidal profile based on the time that's ellapsed.*
- void [set\\_endpts](#) (double start, double end)
- void [set\\_accel](#) (double accel)
- void [set\\_max\\_v](#) (double max\_v)
- double [get\\_movement\\_time](#) ()

### 4.73.1 Detailed Description

#### Trapezoid Profile

This is a motion profile defined by an acceleration, maximum velocity, start point and end point. Using this information, a parametric function is generated, with a period of acceleration, constant velocity, and deceleration. The velocity graph looks like a trapezoid, giving it its name.

If the maximum velocity is set high enough, this will become a S-curve profile, with only acceleration and deceleration.

This class is designed for use in properly modelling the motion of the robots to create a feedforward and target for [PID](#). Acceleration and Maximum velocity should be measured on the robot and tuned down slightly to account for battery drop.

Here are the equations graphed for ease of understanding: <https://www.desmos.com/calculator/rkm3ivulyk>

#### Author

Ryan McGee

#### Date

7/12/2022

### 4.73.2 Constructor & Destructor Documentation

#### TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
    double max_v,
    double accel)
```

Construct a new Trapezoid Profile object.

#### Parameters

<i>max_v</i>	Maximum velocity the robot can run at
<i>accel</i>	Maximum acceleration of the robot

### 4.73.3 Member Function Documentation

#### calculate()

```
motion_t TrapezoidProfile::calculate (
    double time_s)
```

Run the trapezoidal profile based on the time that's elapsed.

**Parameters**

<i>time</i> ↔ _s	Time since start of movement
---------------------	------------------------------

**Returns**

[motion\\_t](#) Position, velocity and acceleration

**get\_movement\_time()**

```
double TrapezoidProfile::get_movement_time ()
```

uses the kinematic equations to and specified accel and max\_v to figure out how long moving along the profile would take

**Returns**

the time the path will take to travel

**set\_accel()**

```
void TrapezoidProfile::set_accel (
    double accel)
```

set\_accel sets the acceleration this profile will use (the left and right legs of the trapezoid)

**Parameters**

<i>accel</i>	the acceleration amount to use
--------------	--------------------------------

**set\_endpts()**

```
void TrapezoidProfile::set_endpts (
    double start,
    double end)
```

set\_endpts defines a start and end position

**Parameters**

<i>start</i>	the starting position of the path
<i>end</i>	the ending position of the path

**set\_max\_v()**

```
void TrapezoidProfile::set_max_v (
    double max_v)
```

sets the maximum velocity for the profile (the height of the top of the trapezoid)

## Parameters

<i>max</i> ↔ _v	the maximum velocity the robot can travel at
--------------------	--

The documentation for this class was generated from the following files:

- trapezoid\_profile.h
- trapezoid\_profile.cpp

## 4.74 TurnDegreesCommand Class Reference

```
#include <drive_commands.h>
```

### Public Member Functions

- [TurnDegreesCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double degrees, double max\_speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

#### 4.74.1 Detailed Description

AutoCommand wrapper class for the turn\_degrees function in the [TankDrive](#) class

#### 4.74.2 Constructor & Destructor Documentation

##### TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double degrees,
    double max_speed = 1,
    double end_speed = 0)
```

Construct a [TurnDegreesCommand](#) Command

## Parameters

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the turn
<i>degrees</i>	how many degrees to rotate
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

### 4.74.3 Member Function Documentation

#### on\_timeout()

```
void TurnDegreesCommand::on_timeout () [override]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

#### run()

```
bool TurnDegreesCommand::run () [override]
```

Run turn\_degrees Overrides run from AutoCommand

#### Returns

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- drive\_commands.h
- drive\_commands.cpp

## 4.75 TurnToHeadingCommand Class Reference

```
#include <drive_commands.h>
```

### Public Member Functions

- [TurnToHeadingCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double heading\_deg, double speed=1, double end\_speed=0)
- bool [run](#) () override
- void [on\\_timeout](#) () override

### 4.75.1 Detailed Description

AutoCommand wrapper class for the turn\_to\_heading() function in the [TankDrive](#) class

### 4.75.2 Constructor & Destructor Documentation

#### TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (  
    TankDrive & drive_sys,  
    Feedback & feedback,  
    double heading_deg,  
    double max_speed = 1,  
    double end_speed = 0)
```

Construct a [TurnToHeadingCommand](#) Command

**Parameters**

<i>drive_sys</i>	the drive system we are commanding
<i>feedback</i>	the feedback controller we are using to execute the drive
<i>heading_deg</i>	the heading to turn to in degrees
<i>max_speed</i>	0 -> 1 percentage of the drive systems speed to drive at

**4.75.3 Member Function Documentation****on\_timeout()**

```
void TurnToHeadingCommand::on_timeout () [override]
```

Cleans up drive system if we time out before finishing

reset the drive system if we don't hit our target

**run()**

```
bool TurnToHeadingCommand::run () [override]
```

Run turn\_to\_heading Overrides run from AutoCommand

**Returns**

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- drive\_commands.h
- drive\_commands.cpp

**4.76 Twist2d Class Reference**

```
#include <twist2d.h>
```

**Public Member Functions**

- constexpr [Twist2d](#) ()
- [Twist2d](#) (const double &dx, const double &dy, const double &dtheta)
- [Twist2d](#) (const Eigen::Vector3d &twist\_vector)
- double dx () const
- double dy () const
- double dtheta () const
- bool operator== (const [Twist2d](#) &other) const
- [Twist2d](#) operator\* (const double &scalar) const
- [Twist2d](#) operator/ (const double &scalar) const

## Friends

- `std::ostream & operator<< (std::ostream &os, const Twist2d &twist)`

### 4.76.1 Detailed Description

Class representing a difference between two poses, more specifically a distance along an arc from a pose.

Assumes conventional cartesian coordinate system: Looking down at the coordinate plane, +X is right +Y is up +Theta is counterclockwise

### 4.76.2 Constructor & Destructor Documentation

#### Twist2d() [1/3]

```
Twist2d::Twist2d () [constexpr]
```

Default Constructor for [Twist2d](#)

#### Twist2d() [2/3]

```
Twist2d::Twist2d (  
    const double & dx,  
    const double & dy,  
    const double & dtheta)
```

Constructs a twist with given translation and angle deltas.

##### Parameters

<i>dx</i>	the linear dx component.
<i>dy</i>	the linear dy component.
<i>dtheta</i>	the angular dtheta component.

#### Twist2d() [3/3]

```
Twist2d::Twist2d (  
    const Eigen::Vector3d & twist_vector)
```

Constructs a twist with given translation and angle deltas.

##### Parameters

<i>twist_vector</i>	vector of the form [dx, dy, dtheta]
---------------------	-------------------------------------

### 4.76.3 Member Function Documentation

#### dtheta()

```
double Twist2d::dtheta () const
```

Returns the angular dtheta component.

##### Returns

the angular dtheta component.

#### dx()

```
double Twist2d::dx () const
```

Returns the linear dx component.

##### Returns

the linear dx component.

#### dy()

```
double Twist2d::dy () const
```

Returns the linear dy component.

##### Returns

the linear dy component.

#### operator\*()

```
Twist2d Twist2d::operator* (
    const double & scalar) const
```

Multiplies this twist by a scalar.

##### Parameters

<i>scalar</i>	the scalar value to multiply by.
---------------	----------------------------------

#### operator/()

```
Twist2d Twist2d::operator/ (
    const double & scalar) const
```

Divides this twist by a scalar.



**Parameters**

<i>scalar</i>	the scalar value to divide by.
---------------	--------------------------------

**operator==()**

```
bool Twist2d::operator== (
    const Twist2d & other) const
```

Compares this to another twist.

**Parameters**

<i>other</i>	the other twist to compare to.
--------------	--------------------------------

**Returns**

true if each of the components are within 1e-9 of each other.

**4.76.4 Friends And Related Symbol Documentation****operator<<**

```
std::ostream & operator<< (
    std::ostream & os,
    const Twist2d & twist) [friend]
```

Sends a twist to an output stream. Ex. `std::cout << twist;`

prints "Twist2d[dx: (value), dy: (value), drad: (radians)]"

Sends a twist to an output stream. Ex. `std::cout << twist;`

prints "Twist2d[x: (value), y: (value), rad: (radians), deg: (degrees)]"

The documentation for this class was generated from the following files:

- twist2d.h
- twist2d.cpp

**4.77 WaitUntilCondition Class Reference**

Waits until the condition is true.

```
#include <auto_command.h>
```

### 4.77.1 Detailed Description

Waits until the condition is true.

The documentation for this class was generated from the following file:

- `auto_command.h`

## 4.78 WaitUntilUpToSpeedCommand Class Reference

```
#include <flywheel_commands.h>
```

### Public Member Functions

- [WaitUntilUpToSpeedCommand](#) ([Flywheel](#) &flywheel, int threshold\_rpm)
- bool [run](#) () override

### 4.78.1 Detailed Description

AutoCommand that listens to the [Flywheel](#) and waits until it is at its target speed +/- the specified threshold

### 4.78.2 Constructor & Destructor Documentation

#### WaitUntilUpToSpeedCommand()

```
WaitUntilUpToSpeedCommand::WaitUntilUpToSpeedCommand (
    Flywheel & flywheel,
    int threshold_rpm)
```

Creates a [WaitUntilUpToSpeedCommand](#)

#### Parameters

<i>flywheel</i>	the flywheel system we are commanding
<i>threshold_rpm</i>	the threshold over and under the flywheel target RPM that we define to be acceptable

### 4.78.3 Member Function Documentation

#### run()

```
bool WaitUntilUpToSpeedCommand::run () [override]
```

Run spin\_manual Overrides run from AutoCommand

#### Returns

true when execution is complete, false otherwise

The documentation for this class was generated from the following files:

- `flywheel_commands.h`
- `flywheel_commands.cpp`



## Index

- accel
  - OdometryBase, [71](#)
- add
  - CommandController, [16](#), [17](#)
  - GenericAuto, [42](#)
- add\_async
  - GenericAuto, [42](#)
- add\_cancel\_func
  - CommandController, [17](#)
- add\_delay
  - CommandController, [18](#)
  - GenericAuto, [43](#)
- add\_entry
  - ExponentialMovingAverage, [27](#)
  - MovingAverage, [63](#)
- ang\_accel\_deg
  - OdometryBase, [71](#)
- ang\_speed\_deg
  - OdometryBase, [71](#)
- as\_vector
  - Translation2d, [141](#)
- Async, [8](#)
- auto\_drive
  - MecanumDrive, [55](#)
- auto\_turn
  - MecanumDrive, [55](#)
- AutoChooser, [9](#)
  - AutoChooser, [9](#)
  - choice, [10](#)
  - get\_choice, [10](#)
  - list, [10](#)
- AutoChooser::entry\_t, [26](#)
  - name, [26](#)
- background\_task
  - OdometryBase, [68](#)
- BasicSolenoidSet, [10](#)
  - BasicSolenoidSet, [11](#)
  - run, [11](#)
- BasicSpinCommand, [11](#)
  - BasicSpinCommand, [12](#)
  - run, [12](#)
- BasicStopCommand, [13](#)
  - BasicStopCommand, [13](#)
  - run, [13](#)
- bool\_or
  - Serializer, [111](#)
- BrakeType
  - TankDrive, [123](#)
- Branch, [14](#)
- ButtonWidget
  - screen::ButtonWidget, [14](#), [15](#)
- calculate
  - FeedForward, [31](#)
  - TrapezoidProfile, [147](#)
- choice
  - AutoChooser, [10](#)
- cobs\_decode
  - OdometrySerial, [75](#)
- cobs\_encode
  - OdometrySerial, [76](#)
- CommandController, [16](#)
  - add, [16](#), [17](#)
  - add\_cancel\_func, [17](#)
  - add\_delay, [18](#)
  - CommandController, [16](#)
  - last\_command\_timed\_out, [18](#)
  - run, [18](#)
- Condition, [18](#)
- config
  - PID, [89](#)
- control\_continuous
  - Lift< T >, [46](#)
- control\_manual
  - Lift< T >, [46](#)
- control\_setpoints
  - Lift< T >, [46](#)
- Core, [1](#)
- current\_pos
  - OdometryBase, [71](#)
- current\_state
  - StateMachine< System, IDType, Message, delay\_ms, do\_log >, [118](#)
- CustomEncoder, [19](#)
  - CustomEncoder, [19](#)
  - position, [19](#)
  - rotation, [20](#)
  - setPosition, [20](#)
  - setRotation, [20](#)
  - velocity, [20](#)
- degrees
  - Rotation2d, [103](#)
- DelayCommand, [21](#)
  - DelayCommand, [21](#)
  - run, [22](#)
- distance
  - Translation2d, [141](#)
- double\_or
  - Serializer, [111](#)
- draw
  - screen::FunctionPage, [41](#)
  - screen::OdometryPage, [73](#)
  - screen::Page, [83](#)
  - screen::PIDPage, [92](#)
  - screen::StatsPage, [119](#)
- drive
  - MecanumDrive, [56](#)
- drive\_arcade
  - TankDrive, [124](#)

- drive\_correction\_cutoff
  - robot\_specs\_t, 101
- drive\_forward
  - TankDrive, 124, 125
- drive\_raw
  - MecanumDrive, 57
- drive\_tank
  - TankDrive, 125
- drive\_tank\_raw
  - TankDrive, 126
- drive\_to\_point
  - TankDrive, 126, 127
- DriveForwardCommand, 22
  - DriveForwardCommand, 22
  - on\_timeout, 23
  - run, 23
- DriveStopCommand, 23
  - DriveStopCommand, 23
  - run, 24
- DriveToPointCommand, 24
  - DriveToPointCommand, 24, 25
  - run, 25
- dtheta
  - Twist2d, 153
- dx
  - Twist2d, 153
- dy
  - Twist2d, 153
- end\_async
  - OdometryBase, 69
- error\_method
  - PID::pid\_config\_t, 90
- ERROR\_TYPE
  - PID, 86
- exp
  - Pose2d, 95
- ExponentialMovingAverage, 26
  - add\_entry, 27
  - ExponentialMovingAverage, 27
  - get\_size, 28
  - get\_value, 28
- f\_cos
  - Rotation2d, 103
- f\_sin
  - Rotation2d, 104
- f\_tan
  - Rotation2d, 104
- Feedback, 28
  - get, 29
  - init, 29
  - is\_on\_target, 29
  - set\_limits, 30
  - update, 30
- FeedForward, 30
  - calculate, 31
  - FeedForward, 31
- FeedForward::ff\_config\_t, 32
  - kA, 32
  - kG, 32
  - kS, 32
  - kV, 33
- Filter, 33
- Flywheel, 33
  - Flywheel, 34
  - get\_motors, 35
  - get\_target, 35
  - getRPM, 35
  - is\_on\_target, 35
  - Page, 35
  - spin\_manual, 35
  - spin\_rpm, 36
  - SpinRpmCmd, 36
  - spinRPMTask, 37
  - stop, 36
  - WaitUntilUpToSpeedCmd, 37
- FlywheelStopCommand, 37
  - FlywheelStopCommand, 37
  - run, 38
- FlywheelStopMotorsCommand, 38
  - FlywheelStopMotorsCommand, 38
  - run, 39
- FlywheelStopNonTasksCommand, 39
- FunctionCommand, 39
- FunctionCondition, 40
- FunctionPage
  - screen::FunctionPage, 41
- GenericAuto, 42
  - add, 42
  - add\_async, 42
  - add\_delay, 43
  - run, 43
- get
  - Feedback, 29
  - MotionController, 59
  - PID, 87
  - TakeBackHalf, 121
- get\_accel
  - OdometryBase, 69
- get\_angular\_accel\_deg
  - OdometryBase, 69
- get\_angular\_speed\_deg
  - OdometryBase, 69
- get\_async
  - Lift< T >, 47
- get\_choice
  - AutoChooser, 10
- get\_error
  - PID, 87
- get\_mag
  - Translation2d, 141
- get\_motion
  - MotionController, 59
- get\_motors
  - Flywheel, 35
- get\_movement\_time

- TrapezoidProfile, 148
- get\_points
  - PurePursuit::Path, 84
- get\_pose2d
  - OdometrySerial, 76
- get\_position
  - OdometryBase, 69
  - OdometrySerial, 76
  - TankDrive, 128
- get\_radius
  - PurePursuit::Path, 84
- get\_sensor\_val
  - PID, 87
- get\_setpoint
  - Lift< T >, 47
- get\_size
  - ExponentialMovingAverage, 28
  - MovingAverage, 63
- get\_speed
  - OdometryBase, 70
- get\_target
  - Flywheel, 35
  - PID, 87
- get\_value
  - ExponentialMovingAverage, 28
  - MovingAverage, 63
- getRPM
  - Flywheel, 35
- handle
  - OdometryBase, 72
- has\_message
  - StateMachine< System, IDType, Message, delay\_ms, do\_log >::MaybeMessage, 54
- hold
  - Lift< T >, 47
- home
  - Lift< T >, 47
- IfTimePassed, 44
- init
  - Feedback, 29
  - MotionController, 60
  - PID, 87
  - TakeBackHalf, 121
- InOrder, 44
- int\_or
  - Serializer, 111
- inverse
  - Transform2d, 137
- is\_on\_target
  - Feedback, 29
  - Flywheel, 35
  - MotionController, 60
  - PID, 88
  - TakeBackHalf, 121
- is\_valid
  - PurePursuit::Path, 85
- kA
  - FeedForward::ff\_config\_t, 32
- kG
  - FeedForward::ff\_config\_t, 32
- kS
  - FeedForward::ff\_config\_t, 32
- kV
  - FeedForward::ff\_config\_t, 33
- last\_command\_timed\_out
  - CommandController, 18
- Lift
  - Lift< T >, 45
- Lift< T >, 45
  - control\_continuous, 46
  - control\_manual, 46
  - control\_setpoints, 46
  - get\_async, 47
  - get\_setpoint, 47
  - hold, 47
  - home, 47
  - Lift, 45
  - set\_async, 47
  - set\_position, 48
  - set\_sensor\_function, 48
  - set\_sensor\_reset, 48
  - set\_setpoint, 48
- Lift< T >::lift\_cfg\_t, 49
- list
  - AutoChooser, 10
- Log
  - Logger, 50, 51
- log
  - Pose2d, 95
- Logf
  - Logger, 51
- Logger, 49
  - Log, 50, 51
  - Logf, 51
  - Logger, 50
  - LogIn, 51, 52
- LogIn
  - Logger, 51, 52
- MaybeMessage
  - StateMachine< System, IDType, Message, delay\_ms, do\_log >::MaybeMessage, 53
- MecanumDrive, 54
  - auto\_drive, 55
  - auto\_turn, 55
  - drive, 56
  - drive\_raw, 57
  - MecanumDrive, 55
- MecanumDrive::mecanumdrive\_config\_t, 57
- message
  - StateMachine< System, IDType, Message, delay\_ms, do\_log >::MaybeMessage, 54
- modify\_inputs
  - TankDrive, 128

- motion\_t, [57](#)
- MotionController, [58](#)
  - get, [59](#)
  - get\_motion, [59](#)
  - init, [60](#)
  - is\_on\_target, [60](#)
  - MotionController, [59](#)
  - set\_limits, [60](#)
  - tune\_feedforward, [61](#)
  - update, [61](#)
- MotionController::m\_profile\_cfg\_t, [52](#)
- MovingAverage, [62](#)
  - add\_entry, [63](#)
  - get\_size, [63](#)
  - get\_value, [63](#)
- MovingAverage, [62](#)
- mut
  - OdometryBase, [72](#)
- name
  - AutoChooser::entry\_t, [26](#)
- None
  - TankDrive, [123](#)
- norm
  - Translation2d, [141](#)
- normalize
  - Translation2d, [141](#)
- Odometry3Wheel, [64](#)
  - Odometry3Wheel, [65](#)
  - tune, [65](#)
  - update, [66](#)
- Odometry3Wheel::odometry3wheel\_cfg\_t, [66](#)
  - off\_axis\_center\_dist, [66](#)
  - wheel\_diam, [66](#)
  - wheelbase\_dist, [67](#)
- OdometryBase, [67](#)
  - accel, [71](#)
  - ang\_accel\_deg, [71](#)
  - ang\_speed\_deg, [71](#)
  - background\_task, [68](#)
  - current\_pos, [71](#)
  - end\_async, [69](#)
  - get\_accel, [69](#)
  - get\_angular\_accel\_deg, [69](#)
  - get\_angular\_speed\_deg, [69](#)
  - get\_position, [69](#)
  - get\_speed, [70](#)
  - handle, [72](#)
  - mut, [72](#)
  - OdometryBase, [68](#)
  - set\_position, [70](#)
  - smallest\_angle, [70](#)
  - speed, [72](#)
  - update, [71](#)
- OdometryPage
  - screen::OdometryPage, [73](#)
- OdometrySerial, [74](#)
  - cobs\_decode, [75](#)
  - cobs\_encode, [76](#)
  - get\_pose2d, [76](#)
  - get\_position, [76](#)
  - OdometrySerial, [75](#)
  - receive\_cobs\_packet, [77](#)
  - send\_config, [77](#)
  - set\_position, [77](#)
  - update, [78](#)
- OdometryTank, [78](#)
  - OdometryTank, [79, 80](#)
  - set\_position, [81](#)
  - update, [81](#)
- OdomSetPosition, [81](#)
  - OdomSetPosition, [81](#)
  - run, [82](#)
- off\_axis\_center\_dist
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [66](#)
- on\_target\_time
  - PID::pid\_config\_t, [90](#)
- on\_timeout
  - DriveForwardCommand, [23](#)
  - PurePursuitCommand, [100](#)
  - TurnDegreesCommand, [150](#)
  - TurnToHeadingCommand, [151](#)
- operator<<
  - Pose2d, [99](#)
  - Rotation2d, [108](#)
  - Transform2d, [139](#)
  - Translation2d, [146](#)
  - Twist2d, [154](#)
- operator+
  - Pose2d, [96](#)
  - Rotation2d, [104](#)
  - Translation2d, [142](#)
- operator-
  - Pose2d, [96](#)
  - Rotation2d, [105](#)
  - Transform2d, [137](#)
  - Translation2d, [143](#)
- operator/
  - Pose2d, [96](#)
  - Rotation2d, [105](#)
  - Transform2d, [137](#)
  - Translation2d, [143](#)
  - Twist2d, [153](#)
- operator==
  - Pose2d, [96](#)
  - Rotation2d, [106](#)
  - Transform2d, [138](#)
  - Translation2d, [144](#)
  - Twist2d, [154](#)
- operator\*
  - Pose2d, [95](#)
  - Rotation2d, [104](#)
  - Transform2d, [137](#)
  - Translation2d, [142](#)
  - Twist2d, [153](#)
- Page

- Flywheel, 35
- Parallel, 83
- Path
  - PurePursuit::Path, 84
- PID, 85
  - config, 89
  - ERROR\_TYPE, 86
  - get, 87
  - get\_error, 87
  - get\_sensor\_val, 87
  - get\_target, 87
  - init, 87
  - is\_on\_target, 88
  - PID, 86
  - reset, 88
  - set\_limits, 88
  - set\_target, 88
  - update, 89
- PID::pid\_config\_t, 90
  - error\_method, 90
  - on\_target\_time, 90
- PIDPage
  - screen::PIDPage, 91
- Pose2d, 92
  - exp, 95
  - log, 95
  - operator<<, 99
  - operator+, 96
  - operator-, 96
  - operator/, 96
  - operator==, 96
  - operator\*, 95
  - Pose2d, 93, 94
  - relative\_to, 97
  - rotation, 97
  - setRotationDeg, 97
  - setRotationRad, 97
  - setX, 97
  - setY, 98
  - transform\_by, 98
  - translation, 98
  - x, 98
  - y, 98
- position
  - CustomEncoder, 19
- pure\_pursuit
  - TankDrive, 128
- PurePursuit::hermite\_point, 43
- PurePursuit::Path, 84
  - get\_points, 84
  - get\_radius, 84
  - is\_valid, 85
  - Path, 84
- PurePursuit::spline, 116
- PurePursuitCommand, 99
  - on\_timeout, 100
  - PurePursuitCommand, 99
  - run, 100
- radians
  - Rotation2d, 106
- radius
  - tracking\_wheel\_cfg\_t, 133
- receive\_cobs\_packet
  - OdometrySerial, 77
- Rect, 100
- relative\_to
  - Pose2d, 97
- reset
  - PID, 88
- reset\_auto
  - TankDrive, 129
- revolutions
  - Rotation2d, 106
- robot\_specs\_t, 100
  - drive\_correction\_cutoff, 101
- rotate\_around
  - Translation2d, 144
- rotate\_by
  - Translation2d, 144
- rotation
  - CustomEncoder, 20
  - Pose2d, 97
  - Transform2d, 138
- Rotation2d, 101
  - degrees, 103
  - f\_cos, 103
  - f\_sin, 104
  - f\_tan, 104
  - operator<<, 108
  - operator+, 104
  - operator-, 105
  - operator/, 105
  - operator==, 106
  - operator\*, 104
  - radians, 106
  - revolutions, 106
  - Rotation2d, 102, 103
  - rotation\_matrix, 106
  - setDeg, 107
  - setRad, 107
  - wrapped\_degrees\_180, 107
  - wrapped\_degrees\_360, 107
  - wrapped\_radians\_180, 107
  - wrapped\_radians\_360, 108
  - wrapped\_revolutions\_180, 108
  - wrapped\_revolutions\_360, 108
- rotation\_matrix
  - Rotation2d, 106
- run
  - BasicSolenoidSet, 11
  - BasicSpinCommand, 12
  - BasicStopCommand, 13
  - CommandController, 18
  - DelayCommand, 22
  - DriveForwardCommand, 23
  - DriveStopCommand, 24



- DriveToPointCommand, 25
- FlywheelStopCommand, 38
- FlywheelStopMotorsCommand, 39
- GenericAuto, 43
- OdomSetPosition, 82
- PurePursuitCommand, 100
- SpinRPMCommand, 116
- TurnDegreesCommand, 150
- TurnToHeadingCommand, 151
- WaitUntilUpToSpeedCommand, 155
- save\_to\_disk
  - Serializer, 112
- screen::ButtonWidget, 14
  - ButtonWidget, 14, 15
  - update, 15
- screen::FunctionPage, 40
  - draw, 41
  - FunctionPage, 41
  - update, 41
- screen::OdometryPage, 72
  - draw, 73
  - OdometryPage, 73
  - update, 73
- screen::Page, 82
  - draw, 83
  - update, 83
- screen::PIDPage, 91
  - draw, 92
  - PIDPage, 91
  - update, 92
- screen::ScreenData, 109
- screen::SliderWidget, 113
  - SliderWidget, 114
  - update, 114
- screen::StatsPage, 119
  - draw, 119
  - StatsPage, 119
  - update, 119
- send\_config
  - OdometrySerial, 77
- send\_message
  - StateMachine< System, IDType, Message, delay\_ms, do\_log >, 118
- Serializer, 109
  - bool\_or, 111
  - double\_or, 111
  - int\_or, 111
  - save\_to\_disk, 112
  - Serializer, 110
  - set\_bool, 112
  - set\_double, 112
  - set\_int, 112
  - set\_string, 113
  - string\_or, 113
- set\_accel
  - TrapezoidProfile, 148
- set\_async
  - Lift< T >, 47
- set\_bool
  - Serializer, 112
- set\_double
  - Serializer, 112
- set\_endpts
  - TrapezoidProfile, 148
- set\_int
  - Serializer, 112
- set\_limits
  - Feedback, 30
  - MotionController, 60
  - PID, 88
  - TakeBackHalf, 121
- set\_max\_v
  - TrapezoidProfile, 148
- set\_position
  - Lift< T >, 48
  - OdometryBase, 70
  - OdometrySerial, 77
  - OdometryTank, 81
- set\_sensor\_function
  - Lift< T >, 48
- set\_sensor\_reset
  - Lift< T >, 48
- set\_setpoint
  - Lift< T >, 48
- set\_string
  - Serializer, 113
- set\_target
  - PID, 88
- setDeg
  - Rotation2d, 107
- setPosition
  - CustomEncoder, 20
- setRad
  - Rotation2d, 107
- setRotation
  - CustomEncoder, 20
- setRotationDeg
  - Pose2d, 97
- setRotationRad
  - Pose2d, 97
- setX
  - Pose2d, 97
  - Translation2d, 145
- setY
  - Pose2d, 98
  - Translation2d, 145
- SliderWidget
  - screen::SliderWidget, 114
- smallest\_angle
  - OdometryBase, 70
- Smart
  - TankDrive, 123
- speed
  - OdometryBase, 72
- spin\_manual
  - Flywheel, 35

- spin\_rpm
  - Flywheel, 36
- SpinRpmCmd
  - Flywheel, 36
- SpinRPMCommand, 115
  - run, 116
  - SpinRPMCommand, 115
- spinRPMTask
  - Flywheel, 37
- StateMachine
  - StateMachine< System, IDType, Message, delay\_ms, do\_log >, 118
- StateMachine< System, IDType, Message, delay\_ms, do\_log >, 117
  - current\_state, 118
  - send\_message, 118
  - StateMachine, 118
- StateMachine< System, IDType, Message, delay\_ms, do\_log >::MaybeMessage, 53
  - has\_message, 54
  - MaybeMessage, 53
  - message, 54
- StateMachine< System, IDType, Message, delay\_ms, do\_log >::State, 116
- StatsPage
  - screen::StatsPage, 119
- stop
  - Flywheel, 36
  - TankDrive, 129
- string\_or
  - Serializer, 113
- TakeBackHalf, 120
  - get, 121
  - init, 121
  - is\_on\_target, 121
  - set\_limits, 121
  - update, 122
- TankDrive, 122
  - BrakeType, 123
  - drive\_arcade, 124
  - drive\_forward, 124, 125
  - drive\_tank, 125
  - drive\_tank\_raw, 126
  - drive\_to\_point, 126, 127
  - get\_position, 128
  - modify\_inputs, 128
  - None, 123
  - pure\_pursuit, 128
  - reset\_auto, 129
  - Smart, 123
  - stop, 129
  - TankDrive, 123
  - turn\_degrees, 129, 130
  - turn\_to\_heading, 131
  - ZeroVelocity, 123
- theta
  - Translation2d, 145
- theta\_rad
  - tracking\_wheel\_cfg\_t, 133
- tracking\_wheel\_cfg\_t, 132
  - radius, 133
  - theta\_rad, 133
  - x, 133
  - y, 133
- Transform2d, 133
  - inverse, 137
  - operator<<, 139
  - operator-, 137
  - operator/, 137
  - operator==, 138
  - operator\*, 137
  - rotation, 138
  - Transform2d, 134, 136
  - translation, 138
  - x, 138
  - y, 138
- transform\_by
  - Pose2d, 98
- translation
  - Pose2d, 98
  - Transform2d, 138
- Translation2d, 139
  - as\_vector, 141
  - distance, 141
  - get\_mag, 141
  - norm, 141
  - normalize, 141
  - operator<<, 146
  - operator+, 142
  - operator-, 143
  - operator/, 143
  - operator==, 144
  - operator\*, 142
  - rotate\_around, 144
  - rotate\_by, 144
  - setX, 145
  - setY, 145
  - theta, 145
  - Translation2d, 140
  - x, 145
  - y, 145
- TrapezoidProfile, 146
  - calculate, 147
  - get\_movement\_time, 148
  - set\_accel, 148
  - set\_endpts, 148
  - set\_max\_v, 148
  - TrapezoidProfile, 147
- tune
  - Odometry3Wheel, 65
- tune\_feedforward
  - MotionController, 61
- turn\_degrees
  - TankDrive, 129, 130
- turn\_to\_heading
  - TankDrive, 131

- TurnDegreesCommand, [149](#)
  - on\_timeout, [150](#)
  - run, [150](#)
  - TurnDegreesCommand, [149](#)
- TurnToHeadingCommand, [150](#)
  - on\_timeout, [151](#)
  - run, [151](#)
  - TurnToHeadingCommand, [150](#)
- Twist2d, [151](#)
  - dtheta, [153](#)
  - dx, [153](#)
  - dy, [153](#)
  - operator<<, [154](#)
  - operator/, [153](#)
  - operator==, [154](#)
  - operator\*, [153](#)
  - Twist2d, [152](#)
- update
  - Feedback, [30](#)
  - MotionController, [61](#)
  - Odometry3Wheel, [66](#)
  - OdometryBase, [71](#)
  - OdometrySerial, [78](#)
  - OdometryTank, [81](#)
  - PID, [89](#)
  - screen::ButtonWidget, [15](#)
  - screen::FunctionPage, [41](#)
  - screen::OdometryPage, [73](#)
  - screen::Page, [83](#)
  - screen::PIDPage, [92](#)
  - screen::SliderWidget, [114](#)
  - screen::StatsPage, [119](#)
  - TakeBackHalf, [122](#)
- velocity
  - CustomEncoder, [20](#)
- WaitUntilCondition, [154](#)
- WaitUntilUpToSpeedCmd
  - Flywheel, [37](#)
- WaitUntilUpToSpeedCommand, [155](#)
  - run, [155](#)
  - WaitUntilUpToSpeedCommand, [155](#)
- wheel\_diam
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [66](#)
- wheelbase\_dist
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [67](#)
- wrapped\_degrees\_180
  - Rotation2d, [107](#)
- wrapped\_degrees\_360
  - Rotation2d, [107](#)
- wrapped\_radians\_180
  - Rotation2d, [107](#)
- wrapped\_radians\_360
  - Rotation2d, [108](#)
- wrapped\_revolutions\_180
  - Rotation2d, [108](#)
- wrapped\_revolutions\_360
  - Rotation2d, [108](#)
- x
  - Pose2d, [98](#)
  - tracking\_wheel\_cfg\_t, [133](#)
  - Transform2d, [138](#)
  - Translation2d, [145](#)
- y
  - Pose2d, [98](#)
  - tracking\_wheel\_cfg\_t, [133](#)
  - Transform2d, [138](#)
  - Translation2d, [145](#)
- ZeroVelocity
  - TankDrive, [123](#)