

## RIT VEXU Core API

Generated by Doxygen 1.9.8



<b>1 Core</b>	<b>1</b>
1.1 Getting Started	1
1.2 Features	1
<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 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Class Documentation</b>	<b>9</b>
5.1 AutoChooser Class Reference	9
5.1.1 Detailed Description	9
5.1.2 Constructor & Destructor Documentation	9
5.1.2.1 AutoChooser()	9
5.1.3 Member Function Documentation	10
5.1.3.1 add()	10
5.1.3.2 get_choice()	10
5.1.3.3 render()	10
5.1.4 Member Data Documentation	11
5.1.4.1 brain	11
5.1.4.2 choice	11
5.1.4.3 list	11
5.2 AutoCommand Class Reference	11
5.2.1 Detailed Description	12
5.2.2 Member Function Documentation	13
5.2.2.1 on_timeout()	13
5.2.2.2 run()	13
5.2.3 Member Data Documentation	13
5.2.3.1 timeout_seconds	13
5.3 CommandController Class Reference	13
5.3.1 Detailed Description	14
5.3.2 Member Function Documentation	14
5.3.2.1 add() [1/3]	14
5.3.2.2 add() [2/3]	14
5.3.2.3 add() [3/3]	15
5.3.2.4 add_delay()	15
5.3.2.5 last_command_timed_out()	15
5.3.2.6 run()	16
5.4 CustomEncoder Class Reference	16
5.4.1 Detailed Description	16

5.4.2 Constructor & Destructor Documentation	16
5.4.2.1 CustomEncoder()	16
5.4.3 Member Function Documentation	17
5.4.3.1 position()	17
5.4.3.2 rotation()	17
5.4.3.3 setPosition()	17
5.4.3.4 setRotation()	18
5.4.3.5 velocity()	18
5.5 DelayCommand Class Reference	18
5.5.1 Detailed Description	19
5.5.2 Constructor & Destructor Documentation	19
5.5.2.1 DelayCommand()	19
5.5.3 Member Function Documentation	19
5.5.3.1 run()	19
5.6 DriveForwardCommand Class Reference	20
5.6.1 Detailed Description	20
5.6.2 Constructor & Destructor Documentation	21
5.6.2.1 DriveForwardCommand()	21
5.6.3 Member Function Documentation	21
5.6.3.1 on_timeout()	21
5.6.3.2 run()	22
5.7 DriveStopCommand Class Reference	22
5.7.1 Detailed Description	23
5.7.2 Constructor & Destructor Documentation	23
5.7.2.1 DriveStopCommand()	23
5.7.3 Member Function Documentation	23
5.7.3.1 on_timeout()	23
5.7.3.2 run()	23
5.8 DriveToPointCommand Class Reference	24
5.8.1 Detailed Description	24
5.8.2 Constructor & Destructor Documentation	24
5.8.2.1 DriveToPointCommand() [1/2]	24
5.8.2.2 DriveToPointCommand() [2/2]	25
5.8.3 Member Function Documentation	25
5.8.3.1 run()	25
5.9 AutoChooser::entry_t Struct Reference	26
5.9.1 Detailed Description	26
5.9.2 Member Data Documentation	26
5.9.2.1 height	26
5.9.2.2 name	26
5.9.2.3 width	26
5.9.2.4 x	26

5.9.2.5 y . . . . .	27
5.10 Feedback Class Reference . . . . .	27
5.10.1 Detailed Description . . . . .	27
5.10.2 Member Function Documentation . . . . .	28
5.10.2.1 get() . . . . .	28
5.10.2.2 init() . . . . .	28
5.10.2.3 is_on_target() . . . . .	28
5.10.2.4 set_limits() . . . . .	28
5.10.2.5 update() . . . . .	29
5.11 FeedForward Class Reference . . . . .	29
5.11.1 Detailed Description . . . . .	30
5.11.2 Constructor & Destructor Documentation . . . . .	30
5.11.2.1 FeedForward() . . . . .	30
5.11.3 Member Function Documentation . . . . .	30
5.11.3.1 calculate() . . . . .	30
5.12 FeedForward::ff_config_t Struct Reference . . . . .	31
5.12.1 Detailed Description . . . . .	31
5.12.2 Member Data Documentation . . . . .	31
5.12.2.1 kA . . . . .	31
5.12.2.2 kG . . . . .	32
5.12.2.3 kS . . . . .	32
5.12.2.4 kV . . . . .	32
5.13 Flywheel Class Reference . . . . .	32
5.13.1 Detailed Description . . . . .	33
5.13.2 Constructor & Destructor Documentation . . . . .	33
5.13.2.1 Flywheel() [ 1 / 4 ] . . . . .	33
5.13.2.2 Flywheel() [ 2 / 4 ] . . . . .	33
5.13.2.3 Flywheel() [ 3 / 4 ] . . . . .	33
5.13.2.4 Flywheel() [ 4 / 4 ] . . . . .	34
5.13.3 Member Function Documentation . . . . .	34
5.13.3.1 getDesiredRPM() . . . . .	34
5.13.3.2 getFeedforwardValue() . . . . .	34
5.13.3.3 getMotors() . . . . .	35
5.13.3.4 getPID() . . . . .	35
5.13.3.5 getPIDValue() . . . . .	35
5.13.3.6 getRPM() . . . . .	35
5.13.3.7 getTBHGain() . . . . .	35
5.13.3.8 isTaskRunning() . . . . .	36
5.13.3.9 measureRPM() . . . . .	36
5.13.3.10 setPIDTarget() . . . . .	36
5.13.3.11 spin_manual() . . . . .	36
5.13.3.12 spin_raw() . . . . .	37

5.13.3.13 spinRPM()	37
5.13.3.14 stop()	37
5.13.3.15 stopMotors()	37
5.13.3.16 stopNonTasks()	38
5.13.3.17 updatePID()	38
5.14 FlywheelStopCommand Class Reference	38
5.14.1 Detailed Description	39
5.14.2 Constructor & Destructor Documentation	39
5.14.2.1 FlywheelStopCommand()	39
5.14.3 Member Function Documentation	39
5.14.3.1 run()	39
5.15 FlywheelStopMotorsCommand Class Reference	40
5.15.1 Detailed Description	40
5.15.2 Constructor & Destructor Documentation	40
5.15.2.1 FlywheelStopMotorsCommand()	40
5.15.3 Member Function Documentation	41
5.15.3.1 run()	41
5.16 FlywheelStopNonTasksCommand Class Reference	41
5.16.1 Detailed Description	42
5.17 GenericAuto Class Reference	42
5.17.1 Detailed Description	42
5.17.2 Member Function Documentation	42
5.17.2.1 add()	42
5.17.2.2 add_async()	43
5.17.2.3 add_delay()	43
5.17.2.4 run()	43
5.18 GraphDrawer Class Reference	44
5.18.1 Constructor & Destructor Documentation	44
5.18.1.1 GraphDrawer()	44
5.18.2 Member Function Documentation	44
5.18.2.1 add_sample()	44
5.18.2.2 draw()	45
5.19 PurePursuit::hermite_point Struct Reference	45
5.19.1 Detailed Description	46
5.20 Lift< T > Class Template Reference	46
5.20.1 Detailed Description	46
5.20.2 Constructor & Destructor Documentation	47
5.20.2.1 Lift()	47
5.20.3 Member Function Documentation	47
5.20.3.1 control_continuous()	47
5.20.3.2 control_manual()	47
5.20.3.3 control_setpoints()	48

5.20.3.4 get_async()	48
5.20.3.5 get_setpoint()	48
5.20.3.6 hold()	48
5.20.3.7 home()	49
5.20.3.8 set_async()	49
5.20.3.9 set_position()	49
5.20.3.10 set_sensor_function()	49
5.20.3.11 set_sensor_reset()	50
5.20.3.12 set_setpoint()	50
5.21 Lift< T >::lift_cfg_t Struct Reference	50
5.21.1 Detailed Description	50
5.22 Logger Class Reference	51
5.22.1 Detailed Description	51
5.22.2 Constructor & Destructor Documentation	51
5.22.2.1 Logger()	51
5.22.3 Member Function Documentation	52
5.22.3.1 Log() [1/2]	52
5.22.3.2 Log() [2/2]	52
5.22.3.3 Logf() [1/2]	52
5.22.3.4 Logf() [2/2]	53
5.22.3.5 LogIn() [1/2]	53
5.22.3.6 LogIn() [2/2]	53
5.23 MotionController::m_profile_cfg_t Struct Reference	53
5.23.1 Detailed Description	54
5.24 MecanumDrive Class Reference	54
5.24.1 Detailed Description	54
5.24.2 Constructor & Destructor Documentation	55
5.24.2.1 MecanumDrive()	55
5.24.3 Member Function Documentation	55
5.24.3.1 auto_drive()	55
5.24.3.2 auto_turn()	56
5.24.3.3 drive()	56
5.24.3.4 drive_raw()	57
5.25 MecanumDrive::mecanumdrive_config_t Struct Reference	57
5.25.1 Detailed Description	58
5.26 motion_t Struct Reference	58
5.26.1 Detailed Description	58
5.27 MotionController Class Reference	58
5.27.1 Detailed Description	59
5.27.2 Constructor & Destructor Documentation	60
5.27.2.1 MotionController()	60
5.27.3 Member Function Documentation	61

5.27.3.1 <code>get()</code> . . . . .	61
5.27.3.2 <code>get_motion()</code> . . . . .	61
5.27.3.3 <code>init()</code> . . . . .	61
5.27.3.4 <code>is_on_target()</code> . . . . .	61
5.27.3.5 <code>set_limits()</code> . . . . .	62
5.27.3.6 <code>tune_feedforward()</code> . . . . .	62
5.27.3.7 <code>update()</code> . . . . .	63
5.28 MovingAverage Class Reference . . . . .	63
5.28.1 Detailed Description . . . . .	63
5.28.2 Constructor & Destructor Documentation . . . . .	64
5.28.2.1 <code>MovingAverage()</code> [1/2] . . . . .	64
5.28.2.2 <code>MovingAverage()</code> [2/2] . . . . .	64
5.28.3 Member Function Documentation . . . . .	64
5.28.3.1 <code>add_entry()</code> . . . . .	64
5.28.3.2 <code>get_average()</code> . . . . .	64
5.28.3.3 <code>get_size()</code> . . . . .	65
5.29 Odometry3Wheel Class Reference . . . . .	65
5.29.1 Detailed Description . . . . .	66
5.29.2 Constructor & Destructor Documentation . . . . .	67
5.29.2.1 <code>Odometry3Wheel()</code> . . . . .	67
5.29.3 Member Function Documentation . . . . .	67
5.29.3.1 <code>tune()</code> . . . . .	67
5.29.3.2 <code>update()</code> . . . . .	67
5.30 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference . . . . .	68
5.30.1 Detailed Description . . . . .	68
5.30.2 Member Data Documentation . . . . .	68
5.30.2.1 <code>off_axis_center_dist</code> . . . . .	68
5.30.2.2 <code>wheel_diam</code> . . . . .	68
5.30.2.3 <code>wheelbase_dist</code> . . . . .	68
5.31 OdometryBase Class Reference . . . . .	69
5.31.1 Detailed Description . . . . .	70
5.31.2 Constructor & Destructor Documentation . . . . .	70
5.31.2.1 <code>OdometryBase()</code> . . . . .	70
5.31.3 Member Function Documentation . . . . .	70
5.31.3.1 <code>background_task()</code> . . . . .	70
5.31.3.2 <code>end_async()</code> . . . . .	71
5.31.3.3 <code>get_accel()</code> . . . . .	71
5.31.3.4 <code>get_angular_accel_deg()</code> . . . . .	71
5.31.3.5 <code>get_angular_speed_deg()</code> . . . . .	71
5.31.3.6 <code>get_position()</code> . . . . .	71
5.31.3.7 <code>get_speed()</code> . . . . .	72
5.31.3.8 <code>pos_diff()</code> . . . . .	72



5.31.3.9 rot_diff()	72
5.31.3.10 set_position()	73
5.31.3.11 smallest_angle()	73
5.31.3.12 update()	73
5.31.4 Member Data Documentation	74
5.31.4.1 accel	74
5.31.4.2 ang_accel_deg	74
5.31.4.3 ang_speed_deg	74
5.31.4.4 current_pos	74
5.31.4.5 handle	74
5.31.4.6 mut	74
5.31.4.7 speed	74
5.31.4.8 zero_pos	75
5.32 OdometryTank Class Reference	75
5.32.1 Detailed Description	76
5.32.2 Constructor & Destructor Documentation	76
5.32.2.1 OdometryTank() [1/2]	76
5.32.2.2 OdometryTank() [2/2]	77
5.32.3 Member Function Documentation	77
5.32.3.1 set_position()	77
5.32.3.2 update()	77
5.33 OdomSetPosition Class Reference	78
5.33.1 Detailed Description	78
5.33.2 Constructor & Destructor Documentation	78
5.33.2.1 OdomSetPosition()	78
5.33.3 Member Function Documentation	79
5.33.3.1 run()	79
5.34 PID Class Reference	79
5.34.1 Detailed Description	80
5.34.2 Member Enumeration Documentation	80
5.34.2.1 ERROR_TYPE	80
5.34.3 Constructor & Destructor Documentation	80
5.34.3.1 PID()	80
5.34.4 Member Function Documentation	81
5.34.4.1 get()	81
5.34.4.2 get_error()	81
5.34.4.3 get_target()	81
5.34.4.4 get_type()	81
5.34.4.5 init()	81
5.34.4.6 is_on_target()	82
5.34.4.7 reset()	82
5.34.4.8 set_limits()	82

5.34.4.9 set_target()	83
5.34.4.10 update()	83
5.35 PID::pid_config_t Struct Reference	83
5.35.1 Detailed Description	84
5.36 PIDFF Class Reference	84
5.36.1 Member Function Documentation	85
5.36.1.1 get()	85
5.36.1.2 init()	85
5.36.1.3 is_on_target()	85
5.36.1.4 set_limits()	86
5.36.1.5 set_target()	86
5.36.1.6 update() [1/2]	86
5.36.1.7 update() [2/2]	86
5.37 point_t Struct Reference	87
5.37.1 Detailed Description	87
5.37.2 Member Function Documentation	87
5.37.2.1 dist()	87
5.37.2.2 operator+()	88
5.37.2.3 operator-()	88
5.38 pose_t Struct Reference	88
5.38.1 Detailed Description	89
5.39 robot_specs_t Struct Reference	89
5.39.1 Detailed Description	89
5.40 Serializer Class Reference	90
5.40.1 Detailed Description	90
5.40.2 Constructor & Destructor Documentation	90
5.40.2.1 Serializer()	90
5.40.3 Member Function Documentation	91
5.40.3.1 bool_or()	91
5.40.3.2 double_or()	91
5.40.3.3 int_or()	91
5.40.3.4 save_to_disk()	93
5.40.3.5 set_bool()	93
5.40.3.6 set_double()	93
5.40.3.7 set_int()	93
5.40.3.8 set_string()	94
5.40.3.9 string_or()	94
5.41 SpinRPMCommand Class Reference	95
5.41.1 Detailed Description	95
5.41.2 Constructor & Destructor Documentation	95
5.41.2.1 SpinRPMCommand()	95
5.41.3 Member Function Documentation	96

5.41.3.1 run()	96
5.42 PurePursuit::spline Struct Reference	96
5.42.1 Detailed Description	97
5.43 TankDrive Class Reference	97
5.43.1 Detailed Description	97
5.43.2 Constructor & Destructor Documentation	97
5.43.2.1 TankDrive()	97
5.43.3 Member Function Documentation	98
5.43.3.1 drive_arcade()	98
5.43.3.2 drive_forward() [1/2]	98
5.43.3.3 drive_forward() [2/2]	99
5.43.3.4 drive_tank()	99
5.43.3.5 drive_to_point() [1/2]	100
5.43.3.6 drive_to_point() [2/2]	101
5.43.3.7 modify_inputs()	101
5.43.3.8 pure_pursuit()	102
5.43.3.9 reset_auto()	102
5.43.3.10 stop()	103
5.43.3.11 turn_degrees() [1/2]	103
5.43.3.12 turn_degrees() [2/2]	103
5.43.3.13 turn_to_heading() [1/2]	104
5.43.3.14 turn_to_heading() [2/2]	105
5.44 TrapezoidProfile Class Reference	105
5.44.1 Detailed Description	106
5.44.2 Constructor & Destructor Documentation	106
5.44.2.1 TrapezoidProfile()	106
5.44.3 Member Function Documentation	106
5.44.3.1 calculate()	106
5.44.3.2 get_movement_time()	107
5.44.3.3 set_accel()	107
5.44.3.4 set_endpts()	107
5.44.3.5 set_max_v()	108
5.45 TurnDegreesCommand Class Reference	108
5.45.1 Detailed Description	109
5.45.2 Constructor & Destructor Documentation	109
5.45.2.1 TurnDegreesCommand()	109
5.45.3 Member Function Documentation	109
5.45.3.1 on_timeout()	109
5.45.3.2 run()	109
5.46 TurnToHeadingCommand Class Reference	110
5.46.1 Detailed Description	110
5.46.2 Constructor & Destructor Documentation	110

5.46.2.1 TurnToHeadingCommand()	110
5.46.3 Member Function Documentation	111
5.46.3.1 on_timeout()	111
5.46.3.2 run()	111
5.47 Vector2D Class Reference	111
5.47.1 Detailed Description	112
5.47.2 Constructor & Destructor Documentation	112
5.47.2.1 Vector2D() [1/2]	112
5.47.2.2 Vector2D() [2/2]	112
5.47.3 Member Function Documentation	113
5.47.3.1 get_dir()	113
5.47.3.2 get_mag()	113
5.47.3.3 get_x()	113
5.47.3.4 get_y()	113
5.47.3.5 normalize()	114
5.47.3.6 operator*()	114
5.47.3.7 operator+()	114
5.47.3.8 operator-()	114
5.47.3.9 point()	115
5.48 WaitUntilUpToSpeedCommand Class Reference	115
5.48.1 Detailed Description	116
5.48.2 Constructor & Destructor Documentation	116
5.48.2.1 WaitUntilUpToSpeedCommand()	116
5.48.3 Member Function Documentation	116
5.48.3.1 run()	116
<b>6 File Documentation</b>	<b>119</b>
6.1 robot_specs.h	119
6.2 custom_encoder.h	119
6.3 flywheel.h	120
6.4 lift.h	121
6.5 mecanum_drive.h	123
6.6 odometry_3wheel.h	124
6.7 odometry_base.h	125
6.8 odometry_tank.h	125
6.9 screen.h	126
6.10 tank_drive.h	126
6.11 auto_chooser.h	127
6.12 auto_command.h	128
6.13 command_controller.h	128
6.14 delay_command.h	128
6.15 drive_commands.h	129

---

6.16 flywheel_commands.h . . . . .	130
6.17 feedback_base.h . . . . .	131
6.18 feedforward.h . . . . .	131
6.19 generic_auto.h . . . . .	132
6.20 geometry.h . . . . .	132
6.21 graph_drawer.h . . . . .	133
6.22 logger.h . . . . .	133
6.23 math_util.h . . . . .	134
6.24 motion_controller.h . . . . .	134
6.25 moving_average.h . . . . .	135
6.26 pid.h . . . . .	136
6.27 pidff.h . . . . .	136
6.28 pure_pursuit.h . . . . .	137
6.29 serializer.h . . . . .	138
6.30 trapezoid_profile.h . . . . .	138
6.31 vector2d.h . . . . .	139
<b>Index</b>	<b>141</b>



# Chapter 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

In order to simply use this repo, you can either clone it into your VEXcode project folder, or download the .zip and place it into a core/ subfolder. Then follow the instructions for setting up compilation at [Wiki/BuildSystem](#)

If you wish to contribute, follow the instructions at [Wiki/ProjectSetup](#)

### 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
- [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)





## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

AutoChooser . . . . .	9
AutoCommand . . . . .	11
DelayCommand . . . . .	18
DriveForwardCommand . . . . .	20
DriveStopCommand . . . . .	22
DriveToPointCommand . . . . .	24
FlywheelStopCommand . . . . .	38
FlywheelStopMotorsCommand . . . . .	40
FlywheelStopNonTasksCommand . . . . .	41
OdomSetPosition . . . . .	78
SpinRPMCommand . . . . .	95
TurnDegreesCommand . . . . .	108
TurnToHeadingCommand . . . . .	110
WaitUntilUpToSpeedCommand . . . . .	115
CommandController . . . . .	13
vex::encoder	
CustomEncoder . . . . .	16
AutoChooser::entry_t . . . . .	26
Feedback . . . . .	27
MotionController . . . . .	58
PID . . . . .	79
PIDFF . . . . .	84
FeedForward . . . . .	29
FeedForward::ff_config_t . . . . .	31
Flywheel . . . . .	32
GenericAuto . . . . .	42
GraphDrawer . . . . .	44
PurePursuit::hermite_point . . . . .	45
Lift< T > . . . . .	46
Lift< T >::lift_cfg_t . . . . .	50
Logger . . . . .	51
MotionController::m_profile_cfg_t . . . . .	53
MecanumDrive . . . . .	54
MecanumDrive::mecanumdrive_config_t . . . . .	57
motion_t . . . . .	58

MovingAverage . . . . .	63
Odometry3Wheel::odometry3wheel_cfg_t . . . . .	68
OdometryBase . . . . .	69
Odometry3Wheel . . . . .	65
OdometryTank . . . . .	75
PID::pid_config_t . . . . .	83
point_t . . . . .	87
pose_t . . . . .	88
robot_specs_t . . . . .	89
Serializer . . . . .	90
PurePursuit::spline . . . . .	96
TankDrive . . . . .	97
TrapezoidProfile . . . . .	105
Vector2D . . . . .	111

## Chapter 3

# Class Index

### 3.1 Class List

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

AutoChooser	9
AutoCommand	11
CommandController	13
CustomEncoder	16
DelayCommand	18
DriveForwardCommand	20
DriveStopCommand	22
DriveToPointCommand	24
AutoChooser::entry_t	26
Feedback	27
FeedForward	29
FeedForward::ff_config_t	31
Flywheel	32
FlywheelStopCommand	38
FlywheelStopMotorsCommand	40
FlywheelStopNonTasksCommand	41
GenericAuto	42
GraphDrawer	44
PurePursuit::hermite_point	45
Lift< T >	46
Lift< T >::lift_cfg_t	50
Logger	
Class to simplify writing to files	51
MotionController::m_profile_cfg_t	53
MecanumDrive	54
MecanumDrive::mecanumdrive_config_t	57
motion_t	58
MotionController	58
MovingAverage	63
Odometry3Wheel	65
Odometry3Wheel::odometry3wheel_cfg_t	68
OdometryBase	69
OdometryTank	75
OdomSetPosition	78
PID	79

PID::pid_config_t . . . . .	83
PIDFF . . . . .	84
point_t . . . . .	87
pose_t . . . . .	88
robot_specs_t . . . . .	89
Serializer	
Serializes Arbitrary data to a file on the SD Card . . . . .	90
SpinRPMCommand . . . . .	95
PurePursuit::spline . . . . .	96
TankDrive . . . . .	97
TrapezoidProfile . . . . .	105
TurnDegreesCommand . . . . .	108
TurnToHeadingCommand . . . . .	110
Vector2D . . . . .	111
WaitUntilUpToSpeedCommand . . . . .	115

# Chapter 4

## File Index

### 4.1 File List

Here is a list of all documented files with brief descriptions:

include/robot_specs.h . . . . .	119
include/subsystems/custom_encoder.h . . . . .	119
include/subsystems/flywheel.h . . . . .	120
include/subsystems/lift.h . . . . .	121
include/subsystems/mecanum_drive.h . . . . .	123
include/subsystems/screen.h . . . . .	126
include/subsystems/tank_drive.h . . . . .	126
include/subsystems/odometry/odometry_3wheel.h . . . . .	124
include/subsystems/odometry/odometry_base.h . . . . .	125
include/subsystems/odometry/odometry_tank.h . . . . .	125
include/utls/auto_chooser.h . . . . .	127
include/utls/feedback_base.h . . . . .	131
include/utls/feedforward.h . . . . .	131
include/utls/generic_auto.h . . . . .	132
include/utls/geometry.h . . . . .	132
include/utls/graph_drawer.h . . . . .	133
include/utls/logger.h . . . . .	133
include/utls/math_util.h . . . . .	134
include/utls/motion_controller.h . . . . .	134
include/utls/moving_average.h . . . . .	135
include/utls/pid.h . . . . .	136
include/utls/pidff.h . . . . .	136
include/utls/pure_pursuit.h . . . . .	137
include/utls/serializer.h . . . . .	138
include/utls/trapezoid_profile.h . . . . .	138
include/utls/vector2d.h . . . . .	139
include/utls/command_structure/auto_command.h . . . . .	128
include/utls/command_structure/command_controller.h . . . . .	128
include/utls/command_structure/delay_command.h . . . . .	128
include/utls/command_structure/drive_commands.h . . . . .	129
include/utls/command_structure/flywheel_commands.h . . . . .	130



## Chapter 5

# Class Documentation

### 5.1 AutoChooser Class Reference

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

#### Classes

- struct [entry\\_t](#)

#### Public Member Functions

- [AutoChooser](#) (vex::brain &[brain](#))
- void [add](#) (std::string name)
- std::string [get\\_choice](#) ()

#### Protected Member Functions

- void [render](#) ([entry\\_t](#) \*selected)

#### Protected Attributes

- std::string [choice](#)
- std::vector< [entry\\_t](#) > [list](#)
- vex::brain & [brain](#)

#### 5.1.1 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.

#### 5.1.2 Constructor & Destructor Documentation

##### 5.1.2.1 AutoChooser()

```
AutoChooser::AutoChooser (
    vex::brain & brain )
```

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
--------------	--

## 5.1.3 Member Function Documentation

### 5.1.3.1 add()

```
void AutoChooser::add (
    std::string name )
```

Add an auto path to the chooser

**Parameters**

<i>name</i>	The name of the path. This should be used as an human readable identifier to the auto path
-------------	--

Add a new autonomous option. There are 3 options per row.

### 5.1.3.2 get\_choice()

```
std::string AutoChooser::get_choice ( )
```

Get the currently selected auto choice

**Returns**

the identifier to the auto path

Return the selected autonomous

### 5.1.3.3 render()

```
void AutoChooser::render (
    entry_t * selected ) [protected]
```

Place all the autonomous choices on the screen. If one is selected, change it's color

**Parameters**

<i>selected</i>	the choice that is currently selected
-----------------	---------------------------------------



## 5.1.4 Member Data Documentation

### 5.1.4.1 brain

```
vex::brain& AutoChooser::brain [protected]
```

the brain to show the choices on

### 5.1.4.2 choice

```
std::string AutoChooser::choice [protected]
```

the current choice of auto

### 5.1.4.3 list

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

< a list of all possible auto choices

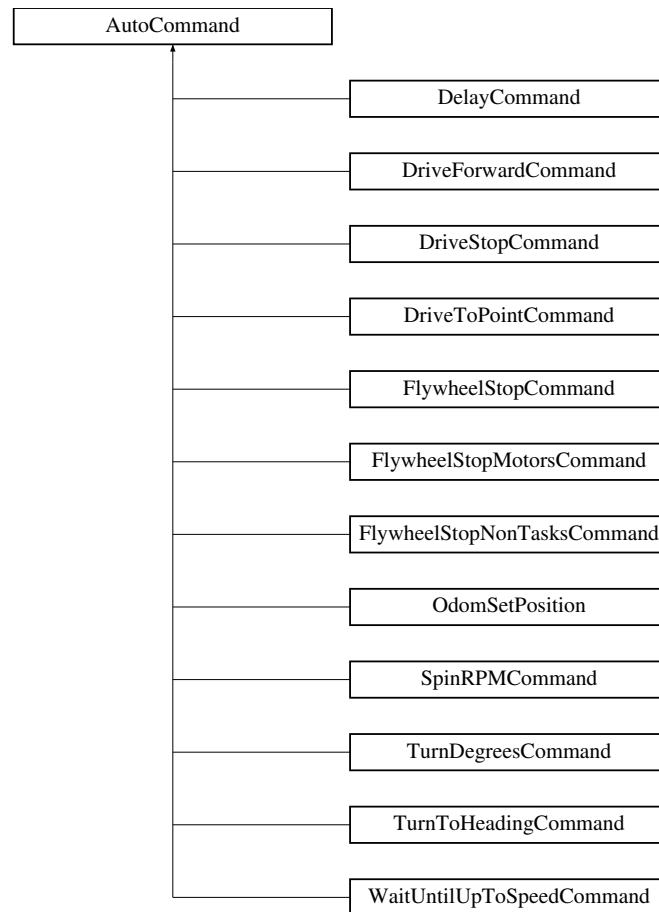
The documentation for this class was generated from the following files:

- include/utils/auto\_chooser.h
- src/utils/auto\_chooser.cpp

## 5.2 AutoCommand Class Reference

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

Inheritance diagram for AutoCommand:



### Public Member Functions

- virtual bool `run()`
- virtual void `on_timeout()`
- `AutoCommand * withTimeout` (double t\_seconds)

### Public Attributes

- double `timeout_seconds` = default\_timeout

### Static Public Attributes

- static constexpr double `default_timeout` = 10.0

## 5.2.1 Detailed Description

File: [auto\\_command.h](#) Desc: Interface for module-specific commands

## 5.2.2 Member Function Documentation

### 5.2.2.1 on\_timeout()

```
virtual void AutoCommand::on_timeout ( ) [inline], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented in [DriveForwardCommand](#), [TurnDegreesCommand](#), [TurnToHeadingCommand](#), and [DriveStopCommand](#).

### 5.2.2.2 run()

```
virtual bool AutoCommand::run ( ) [inline], [virtual]
```

Executes the command Overridden by child classes

#### Returns

true when the command is finished, false otherwise

Reimplemented in [DelayCommand](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [DriveToPointCommand](#), [TurnToHeadingCommand](#), [DriveStopCommand](#), [OdomSetPosition](#), [SpinRPMCommand](#), [WaitUntilUpToSpeedCommand](#), [FlywheelStopCommand](#), and [FlywheelStopMotorsCommand](#).

## 5.2.3 Member Data Documentation

### 5.2.3.1 timeout\_seconds

```
double AutoCommand::timeout_seconds = default_timeout
```

How long to run until we cancel this command. If the command is cancelled, [on\\_timeout\(\)](#) is called to allow any cleanup from the function. If the timeout\_seconds  $\leq 0$ , no timeout will be applied and this command will run forever A timeout can come in handy for some commands that can not reach the end due to some physical limitation such as

- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

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

- include/utils/command\_structure/auto\_command.h

## 5.3 CommandController Class Reference

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

## Public Member Functions

- void [add](#) ([AutoCommand](#) \*cmd, double timeout\_seconds=10.0)
- void [add](#) (std::vector< [AutoCommand](#) \* > cmds)
- void [add](#) (std::vector< [AutoCommand](#) \* > cmds, double timeout\_sec)
- void [add\\_delay](#) (int ms)
- void [run](#) ()
- bool [last\\_command\\_timed\\_out](#) ()

### 5.3.1 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.

### 5.3.2 Member Function Documentation

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

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

Adds a command to the queue

##### Parameters

<i>cmd</i>	the <a href="#">AutoCommand</a> 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

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 <a href="#">AutoCommand</a> 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

#### 5.3.2.2 [add\(\)](#) [2/3]

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

Add multiple commands to the queue. No timeout here.

## Parameters

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

**5.3.2.3 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
<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

**5.3.2.4 add\_delay()**

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

Adds a command that will delay progression of the queue

## Parameters

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

**5.3.2.5 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

### 5.3.2.6 run()

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

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

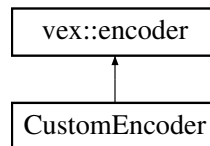
The documentation for this class was generated from the following files:

- include/utls/command\_structure/command\_controller.h
- src/utls/command\_structure/command\_controller.cpp

## 5.4 CustomEncoder Class Reference

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

Inheritance diagram for CustomEncoder:



### 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)

### 5.4.1 Detailed Description

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

### 5.4.2 Constructor & Destructor Documentation

#### 5.4.2.1 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

## 5.4.3 Member Function Documentation

### 5.4.3.1 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

### 5.4.3.2 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

### 5.4.3.3 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 val

**5.4.3.4 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 val

**5.4.3.5 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

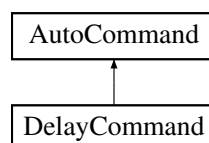
The documentation for this class was generated from the following files:

- include/subsystems/custom\_encoder.h
- src/subsystems/custom\_encoder.cpp

**5.5 DelayCommand Class Reference**

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

Inheritance diagram for DelayCommand:





### Public Member Functions

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

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

## 5.5.1 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

## 5.5.2 Constructor & Destructor Documentation

### 5.5.2.1 DelayCommand()

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

Construct a delay command

#### Parameters

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

## 5.5.3 Member Function Documentation

### 5.5.3.1 run()

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

Delays for the amount of milliseconds stored in the command Overrides run from [AutoCommand](#)

**Returns**

true when complete

Reimplemented from [AutoCommand](#).

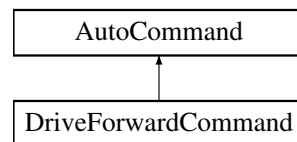
The documentation for this class was generated from the following file:

- include/utils/command\_structure/delay\_command.h

## 5.6 DriveForwardCommand Class Reference

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

Inheritance diagram for DriveForwardCommand:

**Public Member Functions**

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

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

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

**Additional Inherited Members****Public Attributes inherited from [AutoCommand](#)**

- double [timeout\\_seconds](#) = default\_timeout

**Static Public Attributes inherited from [AutoCommand](#)**

- static constexpr double [default\\_timeout](#) = 10.0

### 5.6.1 Detailed Description

[AutoCommand](#) wrapper class for the drive\_forward function in the [TankDrive](#) class

## 5.6.2 Constructor & Destructor Documentation

### 5.6.2.1 DriveForwardCommand()

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

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

## 5.6.3 Member Function Documentation

### 5.6.3.1 on\_timeout()

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

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

### 5.6.3.2 run()

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

Run drive\_forward Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

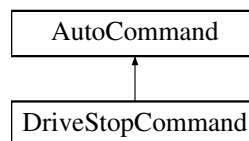
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.7 DriveStopCommand Class Reference

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

Inheritance diagram for DriveStopCommand:



#### Public Member Functions

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

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

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

#### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

#### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.7.1 Detailed Description

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

### 5.7.2 Constructor & Destructor Documentation

#### 5.7.2.1 DriveStopCommand()

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

Construct a DriveStop Command

Parameters

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

### 5.7.3 Member Function Documentation

#### 5.7.3.1 on\_timeout()

```
void DriveStopCommand::on_timeout ( ) [override], [virtual]
```

What to do if we timeout instead of finishing. timeout is specified by the timeout seconds in the constructor

Reimplemented from [AutoCommand](#).

#### 5.7.3.2 run()

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

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

Reimplemented from [AutoCommand](#).

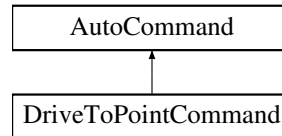
The documentation for this class was generated from the following files:

- include/utls/command\_structure/drive\_commands.h
- src/utls/command\_structure/drive\_commands.cpp

## 5.8 DriveToPointCommand Class Reference

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

Inheritance diagram for DriveToPointCommand:



### Public Member Functions

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

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

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.8.1 Detailed Description

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

## 5.8.2 Constructor & Destructor Documentation

### 5.8.2.1 DriveToPointCommand() [1/2]

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

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

## 5.8.2.2 DriveToPointCommand() [2/2]

```
DriveToPointCommand::DriveToPointCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    point_t point,
    directionType dir,
    double max_speed = 1 )
```

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>point</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

## 5.8.3 Member Function Documentation

## 5.8.3.1 run()

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

Run drive\_to\_point Overrides run from [AutoCommand](#)

## Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utls/command\_structure/drive\_commands.h
- src/utls/command\_structure/drive\_commands.cpp

## 5.9 AutoChooser::entry\_t Struct Reference

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

### Public Attributes

- int [x](#)
- int [y](#)
- int [width](#)
- int [height](#)
- std::string [name](#)

### 5.9.1 Detailed Description

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

### 5.9.2 Member Data Documentation

#### 5.9.2.1 height

```
int AutoChooser::entry_t::height
```

height of the block

#### 5.9.2.2 name

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

name of the auto represented by the block

#### 5.9.2.3 width

```
int AutoChooser::entry_t::width
```

width of the block

#### 5.9.2.4 x

```
int AutoChooser::entry_t::x
```

screen x position of the block



### 5.9.2.5 y

```
int AutoChooser::entry_t::y
```

screen y position of the block

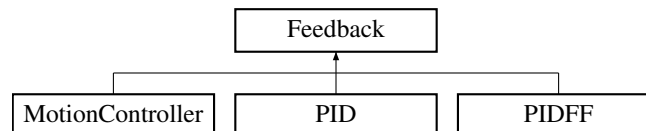
The documentation for this struct was generated from the following file:

- include/utils/auto\_chooser.h

## 5.10 Feedback Class Reference

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

Inheritance diagram for Feedback:



### Public Types

- enum **FeedbackType** { **PIDType** , **FeedforwardType** , **OtherType** }

### 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
- virtual Feedback::FeedbackType **get\_type** ()

### 5.10.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

#### Author

Ryan McGee

#### Date

9/25/2022

## 5.10.2 Member Function Documentation

### 5.10.2.1 `get()`

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

#### Returns

the last saved result from the feedback controller

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

### 5.10.2.2 `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

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

### 5.10.2.3 `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 [PIDFF](#).

### 5.10.2.4 `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 [PIDFF](#).

**5.10.2.5 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 [PIDFF](#).

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

- `include/utlis/feedback_base.h`

## 5.11 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.*

### 5.11.1 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

### 5.11.2 Constructor & Destructor Documentation

#### 5.11.2.1 FeedForward()

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

Creates a [FeedForward](#) object.

#### Parameters

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

### 5.11.3 Member Function Documentation

#### 5.11.3.1 calculate()

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

Perform the feedforward calculation.

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

## Parameters

<i>v</i>	Requested velocity of system
<i>a</i>	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:

- include/utls/feedforward.h

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

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

## Public Attributes

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

### 5.12.1 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

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

### 5.12.2 Member Data Documentation

#### 5.12.2.1 [kA](#)

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

[kA](#) - Acceleration coefficient: the power required to change the mechanism's speed. Multiplied by the requested acceleration.

### 5.12.2.2 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.

### 5.12.2.3 kS

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

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

### 5.12.2.4 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:

- include/utlis/feedforward.h

## 5.13 Flywheel Class Reference

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

### Public Member Functions

- [Flywheel](#) (motor\_group &motors, [PID::pid\\_config\\_t](#) &pid\_config, [FeedForward::ff\\_config\\_t](#) &ff\_config, const double ratio)
- [Flywheel](#) (motor\_group &motors, [FeedForward::ff\\_config\\_t](#) &ff\_config, const double ratio)
- [Flywheel](#) (motor\_group &motors, double tbh\_gain, const double ratio)
- [Flywheel](#) (motor\_group &motors, const double ratio)
- double [getDesiredRPM](#) ()
- bool [isTaskRunning](#) ()
- motor\_group \* [getMotors](#) ()
- double [measureRPM](#) ()
- double [getRPM](#) ()
- [PID](#) \* [getPID](#) ()
- double [getPIDValue](#) ()
- double [getFeedforwardValue](#) ()
- double [getTBHGain](#) ()
- void [setPIDTarget](#) (double value)
- void [updatePID](#) (double value)
- void [spin\\_raw](#) (double speed, directionType dir=fwd)
- void [spin\\_manual](#) (double speed, directionType dir=fwd)
- void [spinRPM](#) (int rpm)
- void [stop](#) ()
- void [stopMotors](#) ()
- void [stopNonTasks](#) ()

### 5.13.1 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 `spinRPM` or `stop` on it at any time and it will take all necessary steps to accomplish this.

### 5.13.2 Constructor & Destructor Documentation

#### 5.13.2.1 [Flywheel\(\)](#) [1/4]

```
Flywheel::Flywheel (
    motor_group & motors,
    PID::pid_config_t & pid_config,
    FeedForward::ff_config_t & ff_config,
    const double ratio )
```

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

##### Parameters

<i>motors</i>	pointer to the motors on the fly wheel
<i>pid_config</i>	pointer the pid config to use
<i>ff_config</i>	the feedforward config to use
<i>ratio</i>	ratio of the whatever just multiplies the velocity

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

#### 5.13.2.2 [Flywheel\(\)](#) [2/4]

```
Flywheel::Flywheel (
    motor_group & motors,
    FeedForward::ff_config_t & ff_config,
    const double ratio )
```

Create the [Flywheel](#) object using only feedforward for control

##### Parameters

<i>motors</i>	the motors on the fly wheel
<i>ff_config</i>	the feedforward config to use
<i>ratio</i>	ratio of the whatever just multiplies the velocity

Create the [Flywheel](#) object using only feedforward for control

#### 5.13.2.3 [Flywheel\(\)](#) [3/4]

```
Flywheel::Flywheel (
    motor_group & motors,
```

```
double tbh_gain,
const double ratio )
```

Create the [Flywheel](#) object using Take Back Half for control

#### Parameters

<i>motors</i>	the motors on the fly wheel
<i>tbh_gain</i>	the TBH control paramater
<i>ratio</i>	ratio of the whatever just multiplies the velocity

Create the [Flywheel](#) object using Take Back Half for control

#### 5.13.2.4 Flywheel() [4/4]

```
Flywheel::Flywheel (
    motor_group & motors,
    const double ratio )
```

Create the [Flywheel](#) object using Bang Bang for control

#### Parameters

<i>motors</i>	the motors on the fly wheel
<i>ratio</i>	ratio of the whatever just multiplies the velocity

Create the [Flywheel](#) object using Bang Bang for control

### 5.13.3 Member Function Documentation

#### 5.13.3.1 getDesiredRPM()

```
double Flywheel::getDesiredRPM ( )
```

Return the RPM that the flywheel is currently trying to achieve

#### Returns

RPM the target rpm

Return the current value that the RPM should be set to

#### 5.13.3.2 getFeedforwardValue()

```
double Flywheel::getFeedforwardValue ( )
```

returns the current OUT value of the [PID](#) - the value that the [PID](#) would set the motors to

returns the current OUT value of the Feedforward - the value that the Feedforward would set the motors to

#### Returns

the voltage that feedforward wants the motors at to achieve the target RPM



### 5.13.3.3 getMotors()

```
motor_group * Flywheel::getMotors ( )
```

Returns a POINTER to the motors

Returns a POINTER TO the motors; not currently used.

#### Returns

motorPointer -pointer to the motors

### 5.13.3.4 getPID()

```
PID * Flywheel::getPID ( )
```

Returns a POINTER to the [PID](#).

Returns a POINTER TO the [PID](#); not currently used.

#### Returns

pidPointer -pointer to the [PID](#)

### 5.13.3.5 getPIDValue()

```
double Flywheel::getPIDValue ( )
```

returns the current OUT value of the [PID](#) - the value that the [PID](#) would set the motors to

returns the current OUT value of the [PID](#) - the value that the [PID](#) would set the motors to

#### Returns

the voltage that [PID](#) wants the motors at to achieve the target RPM

### 5.13.3.6 getRPM()

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

return the current smoothed velocity of the flywheel motors, in RPM

### 5.13.3.7 getTBHGain()

```
double Flywheel::getTBHGain ( )
```

get the gain used for TBH control

get the gain used for TBH control

#### Returns

the gain used in TBH control

#### 5.13.3.8 isTaskRunning()

```
bool Flywheel::isTaskRunning ( )
```

Checks if the background RPM controlling task is running

##### Returns

true if the task is running

Checks if the background RPM controlling task is running

##### Returns

taskRunning - If the task is running

#### 5.13.3.9 measureRPM()

```
double Flywheel::measureRPM ( )
```

make a measurement of the current RPM of the flywheel motor and return a smoothed version

return the current velocity of the flywheel motors, in RPM

##### Returns

the measured velocity of the flywheel

#### 5.13.3.10 setPIDTarget()

```
void Flywheel::setPIDTarget (
    double value )
```

Sets the value of the [PID](#) target

##### Parameters

<i>value</i>	- desired value of the <a href="#">PID</a>
--------------	--

#### 5.13.3.11 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 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

**5.13.3.12 spin\_raw()**

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

Spin motors using voltage; defaults forward at 12 volts FOR USE BY TASKS ONLY

## 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

**5.13.3.13 spinRPM()**

```
void Flywheel::spinRPM (
    int inputRPM )
```

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

## Parameters

<i>rpm</i>	- the 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>inputRPM</i>	- set the current RPM
-----------------	-----------------------

**5.13.3.14 stop()**

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

stop the RPM thread and the wheel

**5.13.3.15 stopMotors()**

```
void Flywheel::stopMotors ( )
```

stop only the motors; exclusively for BANG BANG use

### 5.13.3.16 stopNonTasks()

```
void Flywheel::stopNonTasks ( )
```

Stop the motors if the task isn't running - stop manual control

### 5.13.3.17 updatePID()

```
void Flywheel::updatePID (
    double value )
```

updates the value of the [PID](#)

#### Parameters

<i>value</i>	- value to update the <a href="#">PID</a> with
--------------	--

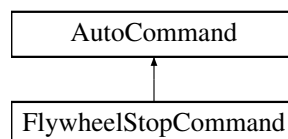
The documentation for this class was generated from the following files:

- include/subsystems/flywheel.h
- src/subsystems/flywheel.cpp

## 5.14 FlywheelStopCommand Class Reference

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

Inheritance diagram for FlywheelStopCommand:



#### Public Member Functions

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

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

## Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

## 5.14.1 Detailed Description

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

## 5.14.2 Constructor & Destructor Documentation

### 5.14.2.1 FlywheelStopCommand()

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

Construct a [FlywheelStopCommand](#)

#### Parameters

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

## 5.14.3 Member Function Documentation

### 5.14.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

#### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

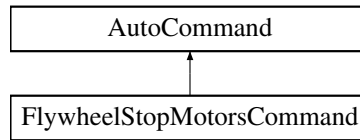
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.15 FlywheelStopMotorsCommand Class Reference

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

Inheritance diagram for FlywheelStopMotorsCommand:



### Public Member Functions

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

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

#### 5.15.1 Detailed Description

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

#### 5.15.2 Constructor & Destructor Documentation

##### 5.15.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheelStopMotors Command

## Parameters

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

### 5.15.3 Member Function Documentation

#### 5.15.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

## Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

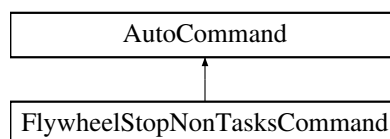
The documentation for this class was generated from the following files:

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp

## 5.16 FlywheelStopNonTasksCommand Class Reference

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

Inheritance diagram for FlywheelStopNonTasksCommand:



#### Additional Inherited Members

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

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

- double [timeout\\_seconds](#) = default\_timeout

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.16.1 Detailed Description

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

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

- include/utlis/command\_structure/flywheel\_commands.h
- src/utlis/command\_structure/flywheel\_commands.cpp

## 5.17 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)

### 5.17.1 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

### 5.17.2 Member Function Documentation

#### 5.17.2.1 [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
------------------	---------------------



### 5.17.2.2 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
--------------------	---------------------

### 5.17.2.3 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
-----------	----------------------------------

### 5.17.2.4 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:

- include/utils/generic\_auto.h
- src/utils/generic\_auto.cpp

## 5.18 GraphDrawer Class Reference

### Public Member Functions

- [GraphDrawer](#) (vex::brain::lcd &screen, int num\_samples, std::string x\_label, std::string y\_label, vex::color col, bool draw\_border, double lower\_bound, double upper\_bound)  
*a helper class to graph values on the brain screen*
- void [add\\_sample](#) ([point\\_t](#) sample)
- void [draw](#) (int x, int y, int width, int height)

### 5.18.1 Constructor & Destructor Documentation

#### 5.18.1.1 GraphDrawer()

```
GraphDrawer::GraphDrawer (
    vex::brain::lcd & screen,
    int num_samples,
    std::string x_label,
    std::string y_label,
    vex::color col,
    bool draw_border,
    double lower_bound,
    double upper_bound )
```

a helper class to graph values on the brain screen

Construct a [GraphDrawer](#)

#### Parameters

<i>screen</i>	a reference to Brain.screen we can save for later
<i>num_samples</i>	the graph works on a fixed window and will plot the last <code>num_samples</code> before the history is forgotten. Larger values give more context but may slow down if you have many graphs or an exceptionally high
<i>x_label</i>	the name of the x axis (currently unused)
<i>y_label</i>	the name of the y axis (currently unused)
<i>draw_border</i>	whether to draw the border around the graph. can be turned off if there are multiple graphs in the same space ie. a graph of error and output
<i>lower_bound</i>	the bottom of the window to graph. if <code>lower_bound == upperbound</code> , the graph will scale to it's datapoints
<i>upper_bound</i>	the top of the window to graph. if <code>lower_bound == upperbound</code> , the graph will scale to it's datapoints

### 5.18.2 Member Function Documentation

#### 5.18.2.1 add\_sample()

```
void GraphDrawer::add_sample (
    point\_t sample )
```

`add_sample` adds a point to the graph, removing one from the back

## Parameters

<i>sample</i>	an x, y coordinate of the next point to graph
---------------	---

## 5.18.2.2 draw()

```
void GraphDrawer::draw (
    int x,
    int y,
    int width,
    int height )
```

draws the graph to the screen in the constructor

## Parameters

<i>x</i>	x position of the top left of the graphed region
<i>y</i>	y position of the top left of the graphed region
<i>width</i>	the width of the graphed region
<i>height</i>	the height of the graphed region

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

- include/utls/graph\_drawer.h
- src/utls/graph\_drawer.cpp

## 5.19 PurePursuit::hermite\_point Struct Reference

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

## Public Member Functions

- [point\\_t](#) `getPoint ()`
- [Vector2D](#) `getTangent ()`

## Public Attributes

- double `x`
- double `y`
- double `dir`
- double `mag`

### 5.19.1 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:

- include/utls/pure\_pursuit.h

## 5.20 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))

### 5.20.1 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

## 5.20.2 Constructor & Destructor Documentation

### 5.20.2.1 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>	<a href="#">Lift</a> characterization information; <a href="#">PID</a> 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

## 5.20.3 Member Function Documentation

### 5.20.3.1 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

### 5.20.3.2 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

**5.20.3.3 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).

**5.20.3.4 get\_async()**

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

## Returns

whether or not the background thread is running the lift

**5.20.3.5 get\_setpoint()**

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

## Returns

The current setpoint for the lift

**5.20.3.6 hold()**

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

Target the class's setpoint. Calculate the PID output and set the lift motors accordingly.

### 5.20.3.7 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.

### 5.20.3.8 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
------------	--

### 5.20.3.9 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

### 5.20.3.10 set\_sensor\_function()

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

### 5.20.3.11 set\_sensor\_reset()

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

### 5.20.3.12 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

<i>val</i>	<a href="#">Lift</a> setpoint, in motor revolutions or sensor units defined by <a href="#">get_sensor</a> . 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:

- `include/subsystems/lift.h`

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

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

#### Public Attributes

- double **up\_speed**
- double **down\_speed**
- double **softstop\_up**
- double **softstop\_down**
- [PID::pid\\_config\\_t](#) **lift\_pid\_cfg**

### 5.21.1 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:

- `include/subsystems/lift.h`



## 5.22 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.*

### Public Attributes

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

### 5.22.1 Detailed Description

Class to simplify writing to files.

### 5.22.2 Constructor & Destructor Documentation

#### 5.22.2.1 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
-----------------	---------------------

### 5.22.3 Member Function Documentation

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

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

Write a string to the log.

## Parameters

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

#### 5.22.3.2 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

#### 5.22.3.3 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

**5.22.3.4 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

**5.22.3.5 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
----------	---------------------

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

```
void Logger::Logln (
    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:

- include/utils/logger.h
- src/utils/logger.cpp

**5.23 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*

### 5.23.1 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:

- include/Utils/motion\_controller.h

## 5.24 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)

### 5.24.1 Detailed Description

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

## 5.24.2 Constructor & Destructor Documentation

### 5.24.2.1 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

## 5.24.3 Member Function Documentation

### 5.24.3.1 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.

**5.24.3.2 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

**5.24.3.3 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)

## 5.24.3.4 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:

- include/subsystems/mecanum\_drive.h
- src/subsystems/mecanum\_drive.cpp

## 5.25 MecanumDrive::mecanumdrive\_config\_t Struct Reference

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

### Public Attributes

- [PID::pid\\_config\\_t](#) **drive\_pid\_conf**
- [PID::pid\\_config\\_t](#) **drive\_gyro\_pid\_conf**
- [PID::pid\\_config\\_t](#) **turn\_pid\_conf**
- double **drive\_wheel\_diam**
- double **lateral\_wheel\_diam**
- double **wheelbase\_width**

#### 5.25.1 Detailed Description

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

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

- include/subsystems/mecanum\_drive.h

## 5.26 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*

#### 5.26.1 Detailed Description

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

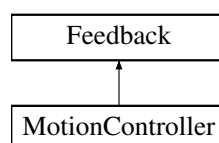
The documentation for this struct was generated from the following file:

- include/utls/trapezoid\_profile.h

## 5.27 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](#) ()

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

- virtual [Feedback::FeedbackType](#) [get\\_type](#) ()

## Static Public Member Functions

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

## Additional Inherited Members

## Public Types inherited from [Feedback](#)

- enum [FeedbackType](#) { [PIDType](#) , [FeedforwardType](#) , [OtherType](#) }

### 5.27.1 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

## 5.27.2 Constructor & Destructor Documentation

### 5.27.2.1 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
---------------	--

## 5.27.3 Member Function Documentation

### 5.27.3.1 get()

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

## Returns

the last saved result from the feedback controller

Implements [Feedback](#).

### 5.27.3.2 get\_motion()

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

## Returns

The current position, velocity and acceleration setpoints

### 5.27.3.3 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](#).

### 5.27.3.4 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](#).

**5.27.3.5 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](#).

**5.27.3.6 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

**5.27.3.7 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:

- include/utils/motion\_controller.h
- src/utils/motion\_controller.cpp

## 5.28 MovingAverage Class Reference

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

**Public Member Functions**

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

### 5.28.1 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).

## 5.28.2 Constructor & Destructor Documentation

### 5.28.2.1 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
--------------------	---

### 5.28.2.2 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

## 5.28.3 Member Function Documentation

### 5.28.3.1 add\_entry()

```
void MovingAverage::add_entry (
    double n )
```

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.
----------	--

### 5.28.3.2 get\_average()

```
double MovingAverage::get_average ( )
```

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

**5.28.3.3 get\_size()**

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

How many samples the average is made from

**Returns**

the number of samples used to calculate this average

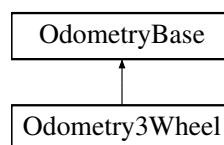
The documentation for this class was generated from the following files:

- `include/utils/moving_average.h`
- `src/utils/moving_average.cpp`

## 5.29 Odometry3Wheel Class Reference

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

Inheritance diagram for Odometry3Wheel:

**Classes**

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

**Public Member Functions**

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

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

- [OdometryBase](#) (bool is\_async)
- [pose\\_t](#) [get\\_position](#) (void)
- virtual void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- 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 [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- 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*

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

- static constexpr [pose\\_t](#) [zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

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

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

## 5.29.1 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



## 5.29.2 Constructor & Destructor Documentation

### 5.29.2.1 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

## 5.29.3 Member Function Documentation

### 5.29.3.1 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

### 5.29.3.2 update()

```
pose_t 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:

- include/subsystems/odometry/odometry\_3wheel.h
- src/subsystems/odometry/odometry\_3wheel.cpp

## 5.30 Odometry3Wheel::odometry3wheel\_cfg\_t Struct Reference

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

**Public Attributes**

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

### 5.30.1 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

### 5.30.2 Member Data Documentation

#### 5.30.2.1 [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

#### 5.30.2.2 [wheel\\_diam](#)

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

the diameter of the tracking wheel

#### 5.30.2.3 [wheelbase\\_dist](#)

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

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

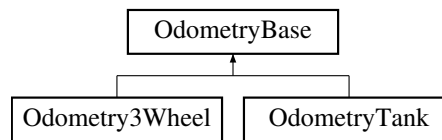
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry\_3wheel.h

## 5.31 OdometryBase Class Reference

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

Inheritance diagram for OdometryBase:



### Public Member Functions

- [OdometryBase](#) (bool is\_async)
- [pose\\_t get\\_position](#) (void)
- virtual void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#))
- virtual [pose\\_t update](#) ()=0
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- 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 [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- 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*

### Static Public Attributes

- static constexpr [pose\\_t zero\\_pos](#) = {.x=0.0L, .y=0.0L, .rot=90.0L}

### Protected Attributes

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 5.31.1 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

### 5.31.2 Constructor & Destructor Documentation

#### 5.31.2.1 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
-----------------	---

### 5.31.3 Member Function Documentation

#### 5.31.3.1 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.

### 5.31.3.2 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.

### 5.31.3.3 get\_accel()

```
double OdometryBase::get_accel ( )
```

Get the current acceleration

#### Returns

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

### 5.31.3.4 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>)

### 5.31.3.5 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)

### 5.31.3.6 get\_position()

```
pose_t OdometryBase::get_position (
    void )
```

Gets the current position and rotation

#### Returns

the position that the odometry believes the robot is at

Gets the current position and rotation

### 5.31.3.7 get\_speed()

```
double OdometryBase::get_speed ( )
```

Get the current speed

#### Returns

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

### 5.31.3.8 pos\_diff()

```
double OdometryBase::pos_diff (
    pose_t start_pos,
    pose_t end_pos ) [static]
```

Get the distance between two points

#### Parameters

<i>start_pos</i>	distance from this point
<i>end_pos</i>	to this point

#### Returns

the euclidean distance between start\_pos and end\_pos

### 5.31.3.9 rot\_diff()

```
double OdometryBase::rot_diff (
    pose_t pos1,
    pose_t pos2 ) [static]
```

Get the change in rotation between two points

#### Parameters

<i>pos1</i>	position with initial rotation
<i>pos2</i>	position with final rotation

#### Returns

change in rotation between pos1 and pos2

Get the change in rotation between two points

**5.31.3.10 set\_position()**

```
void OdometryBase::set_position (
    const pose\_t & 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 [OdometryTank](#).

**5.31.3.11 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>	intitial 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.

**5.31.3.12 update()**

```
virtual pose\_t 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](#), and [OdometryTank](#).

### 5.31.4 Member Data Documentation

#### 5.31.4.1 accel

`double OdometryBase::accel [protected]`

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

#### 5.31.4.2 ang\_accel\_deg

`double OdometryBase::ang_accel_deg [protected]`

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

#### 5.31.4.3 ang\_speed\_deg

`double OdometryBase::ang_speed_deg [protected]`

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

#### 5.31.4.4 current\_pos

`pose_t OdometryBase::current_pos [protected]`

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

#### 5.31.4.5 handle

`vex::task* OdometryBase::handle [protected]`

handle to the vex task that is running the odometry code

#### 5.31.4.6 mut

`vex::mutex OdometryBase::mut [protected]`

Mutex to control multithreading

#### 5.31.4.7 speed

`double OdometryBase::speed [protected]`

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



### 5.31.4.8 zero\_pos

```
constexpr pose_t OdometryBase::zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L} [inline], [static], [constexpr]
```

Zeroed position. X=0, Y=0, Rotation= 90 degrees

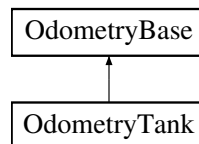
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry\_base.h
- src/subsystems/odometry/odometry\_base.cpp

## 5.32 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\_enc, [CustomEncoder](#) &right\_enc, [robot\\_specs\\_t](#) &config, vex::inertial \*imu=NULL, bool is\_async=true)
- [pose\\_t update](#) () override
- void [set\\_position](#) (const [pose\\_t](#) &newpos=[zero\\_pos](#)) override

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

- [OdometryBase](#) (bool is\_async)
- [pose\\_t get\\_position](#) (void)
- void [end\\_async](#) ()
- double [get\\_speed](#) ()
- 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 [pos\\_diff](#) ([pose\\_t](#) start\_pos, [pose\\_t](#) end\_pos)
- static double [rot\\_diff](#) ([pose\\_t](#) pos1, [pose\\_t](#) pos2)
- 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*

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

- static constexpr [pose\\_t](#) **zero\_pos** = {.x=0.0L, .y=0.0L, .rot=90.0L}

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

- vex::task \* [handle](#)
- vex::mutex [mut](#)
- [pose\\_t](#) [current\\_pos](#)
- double [speed](#)
- double [accel](#)
- double [ang\\_speed\\_deg](#)
- double [ang\\_accel\\_deg](#)

### 5.32.1 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.

### 5.32.2 Constructor & Destructor Documentation

#### 5.32.2.1 [OdometryTank\(\)](#) [1/2]

```
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> .

### 5.32.2.2 OdometryTank() [2/2]

```
OdometryTank::OdometryTank (
    CustomEncoder & left_enc,
    CustomEncoder & right_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_enc</i>	The left motors
<i>right_enc</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> .

## 5.32.3 Member Function Documentation

### 5.32.3.1 set\_position()

```
void OdometryTank::set_position (
    const pose_t & 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](#).

### 5.32.3.2 update()

```
pose_t 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](#).

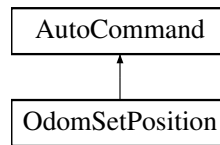
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry\_tank.h
- src/subsystems/odometry/odometry\_tank.cpp

## 5.33 OdomSetPosition Class Reference

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

Inheritance diagram for OdomSetPosition:



### Public Member Functions

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

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.33.1 Detailed Description

[AutoCommand](#) wrapper class for the `set_position` function in the `Odometry` class

### 5.33.2 Constructor & Destructor Documentation

#### 5.33.2.1 OdomSetPosition()

```

OdomSetPosition::OdomSetPosition (
    OdometryBase & odom,
    const pose\_t & 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

### 5.33.3 Member Function Documentation

#### 5.33.3.1 run()

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

Run set\_position Overrides run from [AutoCommand](#)

## Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

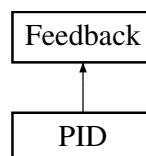
The documentation for this class was generated from the following files:

- include/utils/command\_structure/drive\_commands.h
- src/utils/command\_structure/drive\_commands.cpp

## 5.34 PID Class Reference

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

Inheritance diagram for PID:



## Classes

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

**Public Types**

- enum [ERROR\\_TYPE](#) { [LINEAR](#) , [ANGULAR](#) }

**Public Types inherited from [Feedback](#)**

- enum [FeedbackType](#) { [PIDType](#) , [FeedforwardType](#) , [OtherType](#) }

**Public Member Functions**

- [PID](#) ([pid\\_config\\_t](#) &[config](#))
- void [init](#) (double start\_pt, double set\_pt) override
- double [update](#) (double sensor\_val) override
- double [get](#) () override
- void [set\\_limits](#) (double lower, double upper) override
- bool [is\\_on\\_target](#) () override
- void [reset](#) ()
- double [get\\_error](#) ()
- double [get\\_target](#) ()
- void [set\\_target](#) (double target)
- [Feedback::FeedbackType](#) [get\\_type](#) () override

**Public Attributes**

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

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

**5.34.1 Detailed Description****[PID](#) Class**

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

out = kP\*error + kI\*integral(d Error) + kD\*(dError/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

**5.34.2 Member Enumeration Documentation****5.34.2.1 [ERROR\\_TYPE](#)**

enum [PID::ERROR\\_TYPE](#)

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

**5.34.3 Constructor & Destructor Documentation****5.34.3.1 [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

## 5.34.4 Member Function Documentation

### 5.34.4.1 `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](#).

### 5.34.4.2 `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

### 5.34.4.3 `get_target()`

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

Get the [PID](#)'s target

## Returns

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

### 5.34.4.4 `get_type()`

```
Feedback::FeedbackType PID::get_type ( ) [override], [virtual]
```

Reimplemented from [Feedback](#).

### 5.34.4.5 `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

Implements [Feedback](#).

**5.34.4.6 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](#).

**5.34.4.7 reset()**

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

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

**5.34.4.8 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](#).



## 5.34.4.9 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

## 5.34.4.10 update()

```
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](#).

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

- include/utils/pid.h
- src/utils/pid.cpp

## 5.35 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**

- derivative coefficient  $d * derivative(error)$*
- double **deadband**  
*at what threshold are we close enough to be finished*
- double **on\_target\_time**  
*the time in seconds that we have to be on target for to say we are officially at the target*
- **ERROR\_TYPE error\_method**  
*Linear or angular. wheter to do error as a simple subtraction or to wrap.*

### 5.35.1 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:

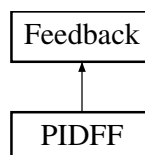
- deadband -
- on\_target\_time - for how long do we have to be at the target to stop As well, [pid\\_config\\_t](#) holds an error type which determines whether errors should be calculated as if the sensor position is a measure of distance or an angle

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

- include/utlis/pid.h

## 5.36 PIDFF Class Reference

Inheritance diagram for PIDFF:



### Public Member Functions

- **PIDFF** ([PID::pid\\_config\\_t](#) &pid\_cfg, [FeedForward::ff\\_config\\_t](#) &ff\_cfg)
- void **init** (double start\_pt, double set\_pt) override
- void **set\_target** (double set\_pt)
- double **update** (double val) override
- double **update** (double val, double vel\_setpt, double a\_setpt=0)
- double **get** () override
- void **set\_limits** (double lower, double upper) override
- bool **is\_on\_target** () override

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

- virtual [Feedback::FeedbackType](#) **get\_type** ()

**Public Attributes**

- [PID](#) pid

**Additional Inherited Members****Public Types inherited from [Feedback](#)**

- enum **FeedbackType** { **PIDType** , **FeedforwardType** , **OtherType** }

**5.36.1 Member Function Documentation****5.36.1.1 get()**

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

**Returns**

the last saved result from the feedback controller

Implements [Feedback](#).

**5.36.1.2 init()**

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

Implements [Feedback](#).

**5.36.1.3 is\_on\_target()**

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

**Returns**

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

**5.36.1.4 set\_limits()**

```
void PIDFF::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](#).

**5.36.1.5 set\_target()**

```
void PIDFF::set_target (
    double set_pt )
```

Set the target of the [PID](#) loop

**Parameters**

<i>set<sub>←</sub> _pt</i>	Setpoint / target value
--------------------------------	-------------------------

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

```
double PIDFF::update (
    double val ) [override], [virtual]
```

Iterate the feedback loop once with an updated sensor value. Only kS for feedforward will be applied.

**Parameters**

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

**Returns**

feedback loop result

Implements [Feedback](#).

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

```
double PIDFF::update (
    double val,
```

```
double vel_setpt,
double a_setpt = 0 )
```

Iterate the feedback loop once with an updated sensor value

#### Parameters

<i>val</i>	value from the sensor
<i>vel_setpt</i>	Velocity for feedforward
<i>a_setpt</i>	Acceleration for feedforward

#### Returns

feedback loop result

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

- include/utls/pidff.h
- src/utls/pidff.cpp

## 5.37 point\_t Struct Reference

```
#include <geometry.h>
```

#### Public Member Functions

- double [dist](#) (const [point\\_t](#) other)
- [point\\_t operator+](#) (const [point\\_t](#) &other)
- [point\\_t operator-](#) (const [point\\_t](#) &other)

#### Public Attributes

- double **x**  
*the x position in space*
- double **y**  
*the y position in space*

### 5.37.1 Detailed Description

Data structure representing an X,Y coordinate

### 5.37.2 Member Function Documentation

#### 5.37.2.1 dist()

```
double point_t::dist (
    const point\_t other ) [inline]
```

dist calculates the euclidian distance between this point and another point using the pythagorean theorem

**Parameters**

<i>other</i>	the point to measure the distance from
--------------	--

**Returns**

the euclidian distance between this and other

**5.37.2.2 operator+()**

```
point_t point_t::operator+ (
    const point_t & other ) [inline]
```

[Vector2D](#) addition operation on points

**Parameters**

<i>other</i>	the point to add on to this
--------------	-----------------------------

**Returns**

this + other (this.x + other.x, this.y + other.y)

**5.37.2.3 operator-()**

```
point_t point_t::operator- (
    const point_t & other ) [inline]
```

[Vector2D](#) subtraction operation on points

**Parameters**

<i>other</i>	the <a href="#">point_t</a> to subtract from this
--------------	---

**Returns**

this - other (this.x - other.x, this.y - other.y)

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

- include/utils/geometry.h

**5.38 pose\_t Struct Reference**

```
#include <geometry.h>
```

**Public Attributes**

- double **x**  
*x position in the world*
- double **y**  
*y position in the world*
- double **rot**  
*rotation in the world*

**5.38.1 Detailed Description**

Describes a single position and rotation

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

- include/utils/geometry.h

**5.39 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**  
*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*
- [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*

**5.39.1 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.

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

- include/robot\_specs.h

## 5.40 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*

### 5.40.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

### 5.40.2 Constructor & Destructor Documentation

#### 5.40.2.1 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>

### 5.40.3 Member Function Documentation

#### 5.40.3.1 `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

#### 5.40.3.2 `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

#### 5.40.3.3 `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

**5.40.3.4 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

**5.40.3.5 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

**5.40.3.6 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

**5.40.3.7 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

#### 5.40.3.8 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

#### 5.40.3.9 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`

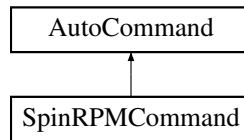
The documentation for this class was generated from the following files:

- `include/utils/serializer.h`
- `src/utils/serializer.cpp`

## 5.41 SpinRPMCommand Class Reference

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

Inheritance diagram for SpinRPMCommand:



### Public Member Functions

- [SpinRPMCommand](#) ([Flywheel](#) &flywheel, int rpm)
- bool [run](#) () override

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

#### 5.41.1 Detailed Description

File: [flywheel\\_commands.h](#) Desc: [insert meaningful desc] [AutoCommand](#) wrapper class for the spinRPM function in the [Flywheel](#) class

#### 5.41.2 Constructor & Destructor Documentation

##### 5.41.2.1 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]

### 5.41.3 Member Function Documentation

#### 5.41.3.1 run()

```
bool SpinRPMCommand::run ( ) [override], [virtual]
```

Run spin\_manual Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utls/command\_structure/flywheel\_commands.h
- src/utls/command\_structure/flywheel\_commands.cpp

## 5.42 PurePursuit::spline Struct Reference

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

**Public Member Functions**

- double **getY** (double x)

**Public Attributes**

- double **a**
- double **b**
- double **c**
- double **d**
- double **x\_start**
- double **x\_end**

### 5.42.1 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:

- `include/utis/pure_pursuit.h`

## 5.43 TankDrive Class Reference

```
#include <tank_drive.h>
```

### Public Member Functions

- [TankDrive](#) (`motor_group &left_motors`, `motor_group &right_motors`, [robot\\_specs\\_t](#) &config, [OdometryBase](#) \*odom=NULL)
- void [stop](#) ()
- void [drive\\_tank](#) (double left, double right, int power=1, bool isdriver=false)
- void [drive\\_arcade](#) (double forward\_back, double left\_right, int power=1)
- bool [drive\\_forward](#) (double inches, `directionType` dir, [Feedback](#) &feedback, double max\_speed=1)
- bool [drive\\_forward](#) (double inches, `directionType` dir, double max\_speed=1)
- bool [turn\\_degrees](#) (double degrees, [Feedback](#) &feedback, double max\_speed=1)
- bool [turn\\_degrees](#) (double degrees, double max\_speed=1)
- bool [drive\\_to\\_point](#) (double x, double y, `vex::directionType` dir, [Feedback](#) &feedback, double max\_speed=1)
- bool [drive\\_to\\_point](#) (double x, double y, `vex::directionType` dir, double max\_speed=1)
- bool [turn\\_to\\_heading](#) (double heading\_deg, [Feedback](#) &feedback, double max\_speed=1)
- bool [turn\\_to\\_heading](#) (double heading\_deg, double max\_speed=1)
- void [reset\\_auto](#) ()
- bool [pure\\_pursuit](#) (`std::vector`< [PurePursuit::hermite\\_point](#) > path, `directionType` dir, double radius, double res, [Feedback](#) &feedback, double max\_speed=1)

### Static Public Member Functions

- static double [modify\\_inputs](#) (double input, int power=2)

### 5.43.1 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

### 5.43.2 Constructor & Destructor Documentation

#### 5.43.2.1 TankDrive()

```
TankDrive::TankDrive (
    motor_group & left_motors,
    motor_group & right_motors,
    robot_specs_t & config,
    OdometryBase * odom = NULL )
```

Create the [TankDrive](#) object

## 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

### 5.43.3 Member Function Documentation

#### 5.43.3.1 drive\_arcade()

```
void TankDrive::drive_arcade (
    double forward_back,
    double left_right,
    int power = 1 )
```

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 $\text{left}^{\text{power}}$ , $\text{right}^{\text{power}}$

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

#### 5.43.3.2 drive\_forward() [1/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    double max_speed = 1 )
```

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

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

## Returns

true if we have finished driving to our point

## 5.43.3.3 drive\_forward() [2/2]

```
bool TankDrive::drive_forward (
    double inches,
    directionType dir,
    Feedback & feedback,
    double max_speed = 1 )
```

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

## 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

## 5.43.3.4 drive\_tank()

```
void TankDrive::drive_tank (
```

```
double left,
double right,
int power = 1,
bool isdriver = false )
```

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>isdriver</i>	default false. if true uses motor percentage. if false uses plain percentage of maximum voltage

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

#### 5.43.3.5 drive\_to\_point() [1/2]

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    double max_speed = 1 )
```

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

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

**Returns**

true if we have reached our target point

**5.43.3.6 drive\_to\_point() [2/2]**

```
bool TankDrive::drive_to_point (
    double x,
    double y,
    vex::directionType dir,
    Feedback & feedback,
    double max_speed = 1 )
```

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

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

**Returns**

true if we have reached our target point

**5.43.3.7 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

**5.43.3.8 pure\_pursuit()**

```
bool TankDrive::pure_pursuit (
    std::vector< PurePursuit::hermite_point > path,
    directionType dir,
    double radius,
    double res,
    Feedback & feedback,
    double max_speed = 1 )
```

Follow a hermite curve using the pure pursuit algorithm.

**Parameters**

<i>path</i>	The hermite curve for the robot to take. Must have 2 or more points.
<i>dir</i>	Whether the robot should move forward or backwards
<i>radius</i>	How the pure pursuit radius, in inches, for finding the lookahead point
<i>res</i>	The number of points to use along the path; the hermite curve is split up into "res" individual points.
<i>feedback</i>	The feedback controller to use
<i>max_speed</i>	Robot's maximum speed throughout the path, between 0 and 1.0

**Returns**

true when we reach the end of the path

**5.43.3.9 reset\_auto()**

```
void TankDrive::reset_auto ( )
```

Reset the initialization for autonomous drive functions

**5.43.3.10 stop()**

```
void TankDrive::stop ( )
```

Stops rotation of all the motors using their "brake mode"

**5.43.3.11 turn\_degrees() [1/2]**

```
bool TankDrive::turn_degrees (
    double degrees,
    double max_speed = 1 )
```

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

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

**Returns**

true if we turned te target number of degrees

**5.43.3.12 turn\_degrees() [2/2]**

```
bool TankDrive::turn_degrees (
    double degrees,
    Feedback & feedback,
    double max_speed = 1 )
```

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

**Returns**

true if we have turned our target number of degrees

**5.43.3.13 turn\_to\_heading() [1/2]**

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    double max_speed = 1 )
```

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

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

**Returns**

true if we have reached our target heading

**5.43.3.14 turn\_to\_heading() [2/2]**

```
bool TankDrive::turn_to_heading (
    double heading_deg,
    Feedback & feedback,
    double max_speed = 1 )
```

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

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

**Returns**

true if we have reached our target heading

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

- include/subsystems/tank\_drive.h
- src/subsystems/tank\_drive.cpp

**5.44 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](#) ()

### 5.44.1 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

### 5.44.2 Constructor & Destructor Documentation

#### 5.44.2.1 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

### 5.44.3 Member Function Documentation

#### 5.44.3.1 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

**5.44.3.2 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

**5.44.3.3 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
--------------	--------------------------------

**5.44.3.4 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

### 5.44.3.5 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_v</i>	the maximum velocity the robot can travel at
--------------	--

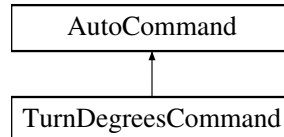
The documentation for this class was generated from the following files:

- include/utils/trapezoid\_profile.h
- src/utils/trapezoid\_profile.cpp

## 5.45 TurnDegreesCommand Class Reference

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

Inheritance diagram for TurnDegreesCommand:



#### Public Member Functions

- [TurnDegreesCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double degrees, double max\_speed=1)
- bool [run](#) () override
- void [on\\_timeout](#) () override

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

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

#### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double **default\_timeout** = 10.0

### 5.45.1 Detailed Description

[AutoCommand](#) wrapper class for the turn\_degrees function in the [TankDrive](#) class

### 5.45.2 Constructor & Destructor Documentation

#### 5.45.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double degrees,
    double max_speed = 1 )
```

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

### 5.45.3 Member Function Documentation

#### 5.45.3.1 on\_timeout()

```
void TurnDegreesCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we timeout

Reimplemented from [AutoCommand](#).

#### 5.45.3.2 run()

```
bool TurnDegreesCommand::run ( ) [override], [virtual]
```

Run turn\_degrees Overrides run from [AutoCommand](#)

##### Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

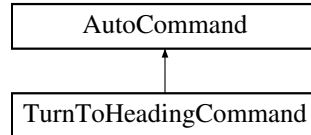
The documentation for this class was generated from the following files:

- include/utls/command\_structure/drive\_commands.h
- src/utls/command\_structure/drive\_commands.cpp

## 5.46 TurnToHeadingCommand Class Reference

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

Inheritance diagram for TurnToHeadingCommand:



### Public Member Functions

- [TurnToHeadingCommand](#) ([TankDrive](#) &drive\_sys, [Feedback](#) &feedback, double heading\_deg, double speed=1)
- bool [run](#) () override
- void [on\\_timeout](#) () override

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

- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

### Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

### Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

#### 5.46.1 Detailed Description

[AutoCommand](#) wrapper class for the [turn\\_to\\_heading\(\)](#) function in the [TankDrive](#) class

#### 5.46.2 Constructor & Destructor Documentation

##### 5.46.2.1 TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    double heading_deg,
    double max_speed = 1 )
```

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

### 5.46.3 Member Function Documentation

#### 5.46.3.1 on\_timeout()

```
void TurnToHeadingCommand::on_timeout ( ) [override], [virtual]
```

Cleans up drive system if we time out before finishing

reset the drive system if we don't hit our target

Reimplemented from [AutoCommand](#).

#### 5.46.3.2 run()

```
bool TurnToHeadingCommand::run ( ) [override], [virtual]
```

Run turn\_to\_heading Overrides run from [AutoCommand](#)

## Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utls/command\_structure/drive\_commands.h
- src/utls/command\_structure/drive\_commands.cpp

## 5.47 Vector2D Class Reference

```
#include <vector2d.h>
```

## Public Member Functions

- [Vector2D](#) (double dir, double mag)
- [Vector2D](#) ([point\\_t](#) p)
- double [get\\_dir](#) () const
- double [get\\_mag](#) () const
- double [get\\_x](#) () const
- double [get\\_y](#) () const
- [Vector2D](#) [normalize](#) ()
- [point\\_t](#) [point](#) ()
- [Vector2D](#) [operator\\*](#) (const double &x)
- [Vector2D](#) [operator+](#) (const [Vector2D](#) &other)
- [Vector2D](#) [operator-](#) (const [Vector2D](#) &other)

### 5.47.1 Detailed Description

[Vector2D](#) is an x,y pair Used to represent 2D locations on the field. It can also be treated as a direction and magnitude

### 5.47.2 Constructor & Destructor Documentation

#### 5.47.2.1 [Vector2D\(\)](#) [1/2]

```
Vector2D::Vector2D (
    double dir,
    double mag )
```

Construct a vector object.

#### Parameters

<i>dir</i>	Direction, in radians. 'foward' is 0, clockwise positive when viewed from the top.
<i>mag</i>	Magnitude.

#### 5.47.2.2 [Vector2D\(\)](#) [2/2]

```
Vector2D::Vector2D (
    point\_t p )
```

Construct a vector object from a cartesian point.

#### Parameters

<i>p</i>	<a href="#">point_t.x</a> , <a href="#">point_t.y</a>
----------	---

### 5.47.3 Member Function Documentation

#### 5.47.3.1 `get_dir()`

```
double Vector2D::get_dir ( ) const
```

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use `r2d()` to convert.

##### Returns

the direction of the vector in radians

Get the direction of the vector, in radians. '0' is forward, clockwise positive when viewed from the top.

Use `r2d()` to convert.

#### 5.47.3.2 `get_mag()`

```
double Vector2D::get_mag ( ) const
```

##### Returns

the magnitude of the vector

Get the magnitude of the vector

#### 5.47.3.3 `get_x()`

```
double Vector2D::get_x ( ) const
```

##### Returns

the X component of the vector; positive to the right.

Get the X component of the vector; positive to the right.

#### 5.47.3.4 `get_y()`

```
double Vector2D::get_y ( ) const
```

##### Returns

the Y component of the vector, positive forward.

Get the Y component of the vector, positive forward.

### 5.47.3.5 normalize()

```
Vector2D Vector2D::normalize ( )
```

Changes the magnitude of the vector to 1

#### Returns

the normalized vector

Changes the magnetude of the vector to 1

### 5.47.3.6 operator\*()

```
Vector2D Vector2D::operator* (
    const double & x )
```

Scales a [Vector2D](#) by a scalar with the \* operator

#### Parameters

<i>x</i>	the value to scale the vector by
----------	----------------------------------

#### Returns

the this [Vector2D](#) scaled by x

### 5.47.3.7 operator+()

```
Vector2D Vector2D::operator+ (
    const Vector2D & other )
```

Add the components of two vectors together [Vector2D](#) + [Vector2D](#) = (this.x + other.x, this.y + other.y)

#### Parameters

<i>other</i>	the vector to add to this
--------------	---------------------------

#### Returns

the sum of the vectors

### 5.47.3.8 operator-()

```
Vector2D Vector2D::operator- (
    const Vector2D & other )
```

Subtract the components of two vectors together [Vector2D](#) - [Vector2D](#) = (this.x - other.x, this.y - other.y)



## Parameters

<i>other</i>	the vector to subtract from this
--------------	----------------------------------

## Returns

the difference of the vectors

## 5.47.3.9 point()

```
point_t Vector2D::point ( )
```

Returns a point from the vector

## Returns

the point represented by the vector

Convert a direction and magnitude representation to an x, y representation

## Returns

the x, y representation of the vector

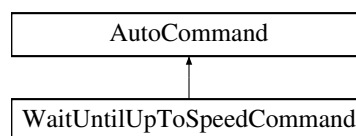
The documentation for this class was generated from the following files:

- include/utils/vector2d.h
- src/utils/vector2d.cpp

## 5.48 WaitUntilUpToSpeedCommand Class Reference

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

Inheritance diagram for WaitUntilUpToSpeedCommand:



## Public Member Functions

- [WaitUntilUpToSpeedCommand](#) ([Flywheel](#) &flywheel, int threshold\_rpm)
- bool [run](#) () override

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

- virtual void [on\\_timeout](#) ()
- [AutoCommand](#) \* [withTimeout](#) (double t\_seconds)

## Additional Inherited Members

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

- double [timeout\\_seconds](#) = default\_timeout

## Static Public Attributes inherited from [AutoCommand](#)

- static constexpr double [default\\_timeout](#) = 10.0

### 5.48.1 Detailed Description

[AutoCommand](#) that listens to the [Flywheel](#) and waits until it is at its target speed +/- the specified threshold

### 5.48.2 Constructor & Destructor Documentation

#### 5.48.2.1 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

### 5.48.3 Member Function Documentation

#### 5.48.3.1 run()

```
bool WaitUntilUpToSpeedCommand::run ( ) [override], [virtual]
```

Run spin\_manual Overrides run from [AutoCommand](#)

**Returns**

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utils/command\_structure/flywheel\_commands.h
- src/utils/command\_structure/flywheel\_commands.cpp



# Chapter 6

## File Documentation

### 6.1 robot\_specs.h

```
00001 #pragma once
00002 #include "../core/include/utils/pid.h"
00003 #include "../core/include/utils/feedback_base.h"
00004
00011 typedef struct
00012 {
00013     double robot_radius;
00014
00015     double odom_wheel_diam;
00016     double odom_gear_ratio;
00017     double dist_between_wheels;
00018
00019     double drive_correction_cutoff;
00020
00021     Feedback *drive_feedback;
00022     Feedback *turn_feedback;
00023     PID::pid_config_t correction_pid;
00024
00025 } robot_specs_t;
```

### 6.2 custom\_encoder.h

```
00001 #pragma once
00002 #include "vex.h"
00003
00008 class CustomEncoder : public vex::encoder
00009 {
00010     typedef vex::encoder super;
00011
00012     public:
00018     CustomEncoder(vex::triport::port &port, double ticks_per_rev);
00019
00025     void setRotation(double val, vex::rotationUnits units);
00026
00032     void setPosition(double val, vex::rotationUnits units);
00033
00039     double rotation(vex::rotationUnits units);
00040
00046     double position(vex::rotationUnits units);
00047
00053     double velocity(vex::velocityUnits units);
00054
00055     private:
00056     double tick_scalar;
00057 };
00058
```

## 6.3 flywheel.h

```

00001 #pragma once
00002 /*****
00003 *
00004 *   File:      Flywheel.h
00005 *   Purpose:   Generalized flywheel class for Core.
00006 *   Author:    Chris Nokes
00007 *
00008 *****/
00009 * EDIT HISTORY
00010 *****/
00011 * 09/23/2022 <CRN> Reorganized, added documentation.
00012 * 09/23/2022 <CRN> Added functions elaborated on in .cpp.
00013 *****/
00014 #include "../core/include/utils/feedforward.h"
00015 #include "vex.h"
00016 #include "../core/include/robot_specs.h"
00017 #include "../core/include/utils/pid.h"
00018 #include <atomic>
00019
00020 using namespace vex;
00021
00022 class Flywheel{
00023     enum FlywheelControlStyle{
00024         PID_Feedforward,
00025         Feedforward,
00026         Take_Back_Half,
00027         Bang_Bang,
00028     };
00029 public:
00030     // CONSTRUCTORS, GETTERS, AND SETTERS
00031     Flywheel(motor_group &motors, PID::pid_config_t &pid_config, FeedForward::ff_config_t &ff_config,
00032             const double ratio);
00033
00034     Flywheel(motor_group &motors, FeedForward::ff_config_t &ff_config, const double ratio);
00035
00036     Flywheel(motor_group &motors, double tbh_gain, const double ratio);
00037
00038     Flywheel(motor_group &motors, const double ratio);
00039
00040     double getDesiredRPM();
00041
00042     bool isTaskRunning();
00043
00044     motor_group* getMotors();
00045
00046     double measureRPM();
00047
00048     double getRPM();
00049     PID* getPID();
00050
00051     double getPIDValue();
00052
00053     double getFeedforwardValue();
00054
00055     double getTBHGain();
00056
00057     void setPIDTarget(double value);
00058
00059     void updatePID(double value);
00060
00061     // SPINNERS AND STOPPERS
00062
00063     void spin_raw(double speed, directionType dir=fwd);
00064
00065     void spin_manual(double speed, directionType dir=fwd);
00066
00067     void spinRPM(int rpm);
00068
00069     void stop();
00070
00071     void stopMotors();
00072
00073     void stopNonTasks();
00074
00075 private:
00076     motor_group &motors;           // motors that make up the flywheel
00077     bool taskRunning = false;      // is the task (thread but not) currently running?
00078     PID pid;                       // PID on the flywheel
00079     FeedForward ff;               // FF constants for the flywheel
00080     double TBH_gain;              // TBH gain parameter for the flywheel
00081     double ratio;                 // multiplies the velocity by this value
00082     std::atomic<double> RPM;       // Desired RPM of the flywheel.

```

```

00179     task rpmTask;                                // task (thread but not) that handles spinning the wheel at a
given RPM
00180     FlywheelControlStyle control_style; // how the flywheel should be controlled
00181     double smoothedRPM;
00182     MovingAverage RPM_avger;
00183 };

```

## 6.4 lift.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/pid.h"
00005 #include <iostream>
00006 #include <map>
00007 #include <atomic>
00008 #include <vector>
00009
00010 using namespace vex;
00011 using namespace std;
00012
00020 template <typename T>
00021 class Lift
00022 {
00023     public:
00024
00031     struct lift_cfg_t
00032     {
00033         double up_speed, down_speed;
00034         double softstop_up, softstop_down;
00035
00036         PID::pid_config_t lift_pid_cfg;
00037     };
00038
00060     Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit
*homing_switch=NULL)
00061     : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg), setpoint_map(setpoint_map),
homing_switch(homing_switch)
00062     {
00063
00064         is_async = true;
00065         setpoint = 0;
00066
00067         // Create a background task that is constantly updating the lift PID, if requested.
00068         // Set once, and forget.
00069         task t([](void* ptr){
00070             Lift &lift = *((Lift*) ptr);
00071
00072             while(true)
00073             {
00074                 if(lift.get_async())
00075                     lift.hold();
00076
00077                 vexDelay(50);
00078             }
00079
00080             return 0;
00081         }, this);
00082     }
00083
00084
00093     void control_continuous(bool up_ctrl, bool down_ctrl)
00094     {
00095         static timer tmr;
00096
00097         double cur_pos = 0;
00098
00099         // Check if there's a hook for a custom sensor. If not, use the motors.
00100         if(get_sensor == NULL)
00101             cur_pos = lift_motors.position(rev);
00102         else
00103             cur_pos = get_sensor();
00104
00105         if(up_ctrl && cur_pos < cfg.softstop_up)
00106         {
00107             lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
00108             setpoint = cur_pos + .3;
00109
00110             // std::cout << "DEBUG OUT: UP " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
"\n";
00111
00112             // Disable the PID while going UP.
00113             is_async = false;

```

```

00114     } else if(down_ctrl && cur_pos > cfg.softstop_down)
00115     {
00116         // Lower the lift slowly, at a rate defined by down_speed
00117         if(setpoint > cfg.softstop_down)
00118             setpoint = setpoint - (tmr.time(sec) * cfg.down_speed);
00119         // std::cout << "DEBUG OUT: DOWN " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
"\n";
00120         is_async = true;
00121     } else
00122     {
00123         // Hold the lift at the last setpoint
00124         is_async = true;
00125     }
00126
00127     tmr.reset();
00128 }
00129
00130 void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down)
00131 {
00132     static bool down_hold = false;
00133     static bool init = true;
00134
00135     // Allow for setting position while still calling this function
00136     if(init || up_btn || down_btn)
00137     {
00138         init = false;
00139         is_async = false;
00140     }
00141
00142     double rev = lift_motors.position(rotationUnits::rev);
00143
00144     if(rev < cfg.softstop_down && down_btn)
00145         down_hold = true;
00146     else if( !down_btn )
00147         down_hold = false;
00148
00149     if(up_btn && rev < cfg.softstop_up)
00150         lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
00151     else if(down_btn && rev > cfg.softstop_down && !down_hold)
00152         lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
00153     else
00154         lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);
00155 }
00156
00157 void control_setpoints(bool up_step, bool down_step, vector<T> pos_list)
00158 {
00159     // Make sure inputs are only processed on the rising edge of the button
00160     static bool up_last = up_step, down_last = down_step;
00161
00162     bool up_rising = up_step && !up_last;
00163     bool down_rising = down_step && !down_last;
00164
00165     up_last = up_step;
00166     down_last = down_step;
00167
00168     static int cur_index = 0;
00169
00170     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00171     if(cur_index >= pos_list.size())
00172         cur_index = pos_list.size() - 1;
00173
00174     // Increment or decrement the index of the list, bringing it up or down.
00175     if(up_rising && cur_index < (pos_list.size() - 1))
00176         cur_index++;
00177     else if(down_rising && cur_index > 0)
00178         cur_index--;
00179
00180     // Set the lift to hold the position in the background with the PID loop
00181     set_position(pos_list[cur_index]);
00182     is_async = true;
00183 }
00184
00185 bool set_position(T pos)
00186 {
00187     this->setpoint = setpoint_map[pos];
00188     is_async = true;
00189
00190     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00191 }
00192
00193 bool set_setpoint(double val)
00194 {
00195     this->setpoint = val;
00196     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00197 }
00198
00199
00200
00201
00202
00203
00204
00205
00206
00207
00208
00209
00210
00211
00212
00213
00214
00215
00216
00217
00218
00219
00220
00221
00222
00223
00224
00225
00226
00227
00228
00229
00230
00231
00232

```



```

00233
00237 double get_setpoint()
00238 {
00239     return this->setpoint;
00240 }
00241
00246 void hold()
00247 {
00248     lift_pid.set_target(setpoint);
00249     // std::cout << "DEBUG OUT: SETPOINT " << setpoint << "\n";
00250
00251     if(get_sensor != NULL)
00252         lift_pid.update(get_sensor());
00253     else
00254         lift_pid.update(lift_motors.position(rev));
00255
00256     // std::cout << "DEBUG OUT: ROTATION " << lift_motors.rotation(rev) << "\n\n";
00257
00258     lift_motors.spin(fwd, lift_pid.get(), volt);
00259 }
00260
00265 void home()
00266 {
00267     static timer tmr;
00268     tmr.reset();
00269
00270     while(tmr.time(sec) < 3)
00271     {
00272         lift_motors.spin(directionType::rev, 6, volt);
00273
00274         if (homing_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
00275             break;
00276         else if (homing_switch != NULL && homing_switch->pressing())
00277             break;
00278     }
00279
00280     if(reset_sensor != NULL)
00281         reset_sensor();
00282
00283     lift_motors.resetPosition();
00284     lift_motors.stop();
00285
00286 }
00287
00291 bool get_async()
00292 {
00293     return is_async;
00294 }
00295
00301 void set_async(bool val)
00302 {
00303     this->is_async = val;
00304 }
00305
00315 void set_sensor_function(double (*fn_ptr) (void))
00316 {
00317     this->get_sensor = fn_ptr;
00318 }
00319
00326 void set_sensor_reset(void (*fn_ptr) (void))
00327 {
00328     this->reset_sensor = fn_ptr;
00329 }
00330
00331 private:
00332
00333     motor_group &lift_motors;
00334     lift_cfg_t &cfg;
00335     PID lift_pid;
00336     map<T, double> &setpoint_map;
00337     limit *homing_switch;
00338
00339     atomic<double> setpoint;
00340     atomic<bool> is_async;
00341
00342     double (*get_sensor)(void) = NULL;
00343     void (*reset_sensor)(void) = NULL;
00344
00345
00346 };

```

## 6.5 mecanum\_drive.h

```
00001 #pragma once
```

```

00002
00003 #include "vex.h"
00004 #include "../core/include/utils/pid.h"
00005
00006 #ifndef PI
00007 #define PI 3.141592654
00008 #endif
00009
00014 class MecanumDrive
00015 {
00016
00017     public:
00018
00022     struct mecanumdrive_config_t
00023     {
00024         // PID configurations for autonomous driving
00025         PID::pid_config_t drive_pid_conf;
00026         PID::pid_config_t drive_gyro_pid_conf;
00027         PID::pid_config_t turn_pid_conf;
00028
00029         // Diameter of the mecanum wheels
00030         double drive_wheel_diam;
00031
00032         // Diameter of the perpendicular undriven encoder wheel
00033         double lateral_wheel_diam;
00034
00035         // Width between the center of the left and right wheels
00036         double wheelbase_width;
00037     };
00038
00039     MecanumDrive(vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor
00040 &right_rear,
00041                 vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, mecanumdrive_config_t
00042 *config=NULL);
00043
00044     void drive_raw(double direction_deg, double magnitude, double rotation);
00045
00046     void drive(double left_y, double left_x, double right_x, int power=2);
00047
00048     bool auto_drive(double inches, double direction, double speed, bool gyro_correction=true);
00049
00050     bool auto_turn(double degrees, double speed, bool ignore_imu=false);
00051
00052     private:
00053
00054     vex::motor &left_front, &right_front, &left_rear, &right_rear;
00055
00056     mecanumdrive_config_t *config;
00057     vex::rotation *lateral_wheel;
00058     vex::inertial *imu;
00059
00060     PID *drive_pid = NULL;
00061     PID *drive_gyro_pid = NULL;
00062     PID *turn_pid = NULL;
00063
00064     bool init = true;
00065
00066 };

```

## 6.6 odometry\_3wheel.h

```

00001 #pragma once
00002 #include "../core/include/subsystems/odometry/odometry_base.h"
00003 #include "../core/include/subsystems/tank_drive.h"
00004 #include "../core/include/subsystems/custom_encoder.h"
00005
00006 class Odometry3Wheel : public OdometryBase
00007 {
00008     public:
00009
00010     typedef struct
00011     {
00012         double wheelbase_dist;
00013         double off_axis_center_dist;
00014         double wheel_diam;
00015     } odometry3wheel_cfg_t;
00016
00017     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
00018                   odometry3wheel_cfg_t &cfg, bool is_async=true);
00019
00020     pose_t update() override;
00021
00022 };

```

```

00075     void tune(vex::controller &con, TankDrive &drive);
00076
00077     private:
00078
00091     static pose_t calculate_new_pos(double lside_delta_deg, double rside_delta_deg, double
offax_delta_deg, pose_t old_pos, odometry3wheel_cfg_t cfg);
00092
00093     CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00094     odometry3wheel_cfg_t &cfg;
00095
00096
00097 };

```

## 6.7 odometry\_base.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/robot_specs.h"
00006
00007 #ifndef PI
00008 #define PI 3.141592654
00009 #endif
00010
00011
00012
00025 class OdometryBase
00026 {
00027 public:
00028
00034     OdometryBase(bool is_async);
00035
00040     pose_t get_position(void);
00041
00046     virtual void set_position(const pose_t& newpos=zero_pos);
00047
00052     virtual pose_t update() = 0;
00053
00061     static int background_task(void* ptr);
00062
00068     void end_async();
00069
00076     static double pos_diff(pose_t start_pos, pose_t end_pos);
00077
00084     static double rot_diff(pose_t pos1, pose_t pos2);
00085
00094     static double smallest_angle(double start_deg, double end_deg);
00095
00097     bool end_task = false;
00098
00103     double get_speed();
00104
00109     double get_accel();
00110
00115     double get_angular_speed_deg();
00116
00121     double get_angular_accel_deg();
00122
00126     inline static constexpr pose_t zero_pos = {.x=0.0L, .y=0.0L, .rot=90.0L};
00127
00128 protected:
00132     vex::task *handle;
00133
00137     vex::mutex mut;
00138
00142     pose_t current_pos;
00143
00144     double speed;
00145     double accel;
00146     double ang_speed_deg;
00147     double ang_accel_deg;
00148 };

```

## 6.8 odometry\_tank.h

```

00001 #pragma once
00002
00003 #include "../core/include/subsystems/odometry/odometry_base.h"

```

```

00004 #include "../core/include/subsystems/custom_encoder.h"
00005 #include "../core/include/utils/geometry.h"
00006 #include "../core/include/utils/vector2d.h"
00007 #include "../core/include/robot_specs.h"
00008
00009 static int background_task(void* odom_obj);
00010
00011
00018 class OdometryTank : public OdometryBase
00019 {
00020 public:
00029     OdometryTank(vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config,
vex::inertial *imu=NULL, bool is_async=true);
00030
00040     OdometryTank(CustomEncoder &left_enc, CustomEncoder &right_enc, robot_specs_t &config,
vex::inertial *imu=NULL, bool is_async=true);
00041
00046     pose_t update() override;
00047
00052     void set_position(const pose_t &newpos=zero_pos) override;
00053
00054
00055
00056 private:
00060     static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff,
double rside_diff, double angle_deg);
00061
00062     vex::motor_group *left_side, *right_side;
00063     CustomEncoder *left_enc, *right_enc;
00064     vex::inertial *imu;
00065     robot_specs_t &config;
00066
00067     double rotation_offset = 0;
00068
00069 };

```

## 6.9 screen.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <vector>
00004
00009
00010 typedef void (*screenFunc)(vex::brain::lcd &screen, int x, int y, int width, int height, bool
first_run);
00011
00012 void draw_mot_header(vex::brain::lcd &screen, int x, int y, int width);
00013 // name should be no longer than 15 characters
00014 void draw_mot_stats(vex::brain::lcd &screen, int x, int y, int width, const char *name, vex::motor
&motor, int animation_tick);
00015 void draw_dev_stats(vex::brain::lcd &screen, int x, int y, int width, const char *name, vex::device
&dev, int animation_tick);
00016
00017 void draw_battery_stats(vex::brain::lcd &screen, int x, int y, double voltage, double percentage);
00018
00019
00020
00021 void draw_lr_arrows(vex::brain::lcd &screen, int bar_width, int width, int height);
00022
00023 int handle_screen_thread(vex::brain::lcd &screen, std::vector<screenFunc> pages, int first_page);
00024 void StartScreen(vex::brain::lcd &screen, std::vector<screenFunc> pages, int first_page = 0);

```

## 6.10 tank\_drive.h

```

00001 #pragma once
00002
00003 #ifndef PI
00004 #define PI 3.141592654
00005 #endif
00006
00007 #include "vex.h"
00008 #include "../core/include/subsystems/odometry/odometry_tank.h"
00009 #include "../core/include/utils/pid.h"
00010 #include "../core/include/utils/feedback_base.h"
00011 #include "../core/include/robot_specs.h"
00012 #include "../core/src/utils/pure_pursuit.cpp"
00013 #include <vector>
00014
00015

```

```

00016 using namespace vex;
00017
00022 class TankDrive
00023 {
00024 public:
00025
00033 TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
*odom=NULL);
00034
00038 void stop();
00039
00050 void drive_tank(double left, double right, int power=1, bool isdriver=false);
00051
00062 void drive_arcade(double forward_back, double left_right, int power=1);
00063
00074 bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed=1);
00075
00084 bool drive_forward(double inches, directionType dir, double max_speed=1);
00085
00096 bool turn_degrees(double degrees, Feedback &feedback, double max_speed=1);
00097
00107 bool turn_degrees(double degrees, double max_speed=1);
00108
00120 bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double
max_speed=1);
00121
00133 bool drive_to_point(double x, double y, vex::directionType dir, double max_speed=1);
00134
00143 bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed=1);
00151 bool turn_to_heading(double heading_deg, double max_speed=1);
00152
00156 void reset_auto();
00157
00166 static double modify_inputs(double input, int power=2);
00167
00179 bool pure_pursuit(std::vector<PurePursuit::hermite_point> path, directionType dir, double radius,
double res, Feedback &feedback, double max_speed=1);
00180
00181 private:
00182 motor_group &left_motors;
00183 motor_group &right_motors;
00184
00185 PID correction_pid;
00186 Feedback *drive_default_feedback = NULL;
00187 Feedback *turn_default_feedback = NULL;
00188
00189 OdometryBase *odometry;
00190
00191 robot_specs_t &config;
00192
00193 bool func_initialized = false;
00194 bool is_pure_pursuit = false;
00195 };

```

## 6.11 auto\_chooser.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <string>
00004 #include <vector>
00005
00006
00015 class AutoChooser
00016 {
00017 public:
00023 AutoChooser(vex::brain &brain);
00024
00029 void add(std::string name);
00030
00035 std::string get_choice();
00036
00037 protected:
00038
00042 struct entry_t
00043 {
00044     int x;
00045     int y;
00046     int width;
00047     int height;
00048     std::string name;
00049 };
00050
00051 void render(entry_t *selected);

```

```

00052
00053     std::string choice;
00054     std::vector<entry_t> list ;
00055     vex::brain &brain;
00058 };

```

## 6.12 auto\_command.h

```

00001
00007 #pragma once
00008
00009 #include "vex.h"
00010
00011 class AutoCommand {
00012 public:
00013     static constexpr double default_timeout = 10.0;
00019     virtual bool run() { return true; }
00023     virtual void on_timeout(){}
00024     AutoCommand* withTimeout(double t_seconds){
00025         this->timeout_seconds = t_seconds;
00026         return this;
00027     }
00037     double timeout_seconds = default_timeout;
00038
00039 };

```

## 6.13 command\_controller.h

```

00001
00010 #pragma once
00011 #include <vector>
00012 #include <queue>
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014
00015 class CommandController
00016 {
00017 public:
00023     void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00024
00029     void add(std::vector<AutoCommand *> cmds);
00030
00036     void add(std::vector<AutoCommand *> cmds, double timeout_sec);
00043     void add_delay(int ms);
00044
00049     void run();
00055     bool last_command_timed_out();
00056
00057 private:
00058     std::queue<AutoCommand *> command_queue;
00059     bool command_timed_out = false;
00060 };

```

## 6.14 delay\_command.h

```

00001
00008 #pragma once
00009
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00012 class DelayCommand: public AutoCommand {
00013 public:
00018     DelayCommand(int ms): ms(ms) {}
00019
00025     bool run() override {
00026         vexDelay(ms);
00027         return true;
00028     }
00029
00030 private:
00031     // amount of milliseconds to wait
00032     int ms;
00033 };

```

## 6.15 drive\_commands.h

```

00001
00019 #pragma once
00020
00021 #include "vex.h"
00022 #include "../core/include/utils/geometry.h"
00023 #include "../core/include/utils/command_structure/auto_command.h"
00024 #include "../core/include/subsystems/tank_drive.h"
00025
00026 using namespace vex;
00027
00028
00029 // ==== DRIVING ====
00030
00036 class DriveForwardCommand: public AutoCommand {
00037     public:
00038         DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
00039                             double max_speed=1);
00039
00045         bool run() override;
00049         void on_timeout() override;
00050
00051     private:
00052         // drive system to run the function on
00053         TankDrive &drive_sys;
00054
00055         // feedback controller to use
00056         Feedback &feedback;
00057
00058         // parameters for drive_forward
00059         double inches;
00060         directionType dir;
00061         double max_speed;
00062 };
00063
00068 class TurnDegreesCommand: public AutoCommand {
00069     public:
00070         TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed =
00071                             1);
00077
00077         bool run() override;
00081         void on_timeout() override;
00082
00083
00084     private:
00085         // drive system to run the function on
00086         TankDrive &drive_sys;
00087
00088         // feedback controller to use
00089         Feedback &feedback;
00090
00091         // parameters for turn_degrees
00092         double degrees;
00093         double max_speed;
00094 };
00095
00100 class DriveToPointCommand: public AutoCommand {
00101     public:
00102         DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType
00103                             dir, double max_speed = 1);
00103         DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
00104                             double max_speed=1);
00110
00110         bool run() override;
00111
00112     private:
00113         // drive system to run the function on
00114         TankDrive &drive_sys;
00115
00119         void on_timeout() override;
00120
00121
00122         // feedback controller to use
00123         Feedback &feedback;
00124
00125         // parameters for drive_to_point
00126         double x;
00127         double y;
00128         directionType dir;
00129         double max_speed;
00130
00131 };
00132
00138 class TurnToHeadingCommand: public AutoCommand {
00139     public:
00140         TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed =

```

```

1);
00141
00147     bool run() override;
00151     void on_timeout() override;
00152
00153
00154 private:
00155     // drive system to run the function on
00156     TankDrive &drive_sys;
00157
00158     // feedback controller to use
00159     Feedback &feedback;
00160
00161     // parameters for turn_to_heading
00162     double heading_deg;
00163     double max_speed;
00164 };
00165
00170 class DriveStopCommand: public AutoCommand {
00171 public:
00172     DriveStopCommand(TankDrive &drive_sys);
00173
00179     bool run() override;
00180     void on_timeout() override;
00181
00182 private:
00183     // drive system to run the function on
00184     TankDrive &drive_sys;
00185 };
00186
00187
00188 // ==== ODOMETRY ====
00189
00194 class OdomSetPosition: public AutoCommand {
00195 public:
00201     OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);
00202
00208     bool run() override;
00209
00210 private:
00211     // drive system with an odometry config
00212     OdometryBase &odom;
00213     pose_t newpos;
00214 };

```

## 6.16 flywheel\_commands.h

```

00001
00007 #pragma once
00008
00009 #include "../core/include/subsystems/flywheel.h"
00010 #include "../core/include/utils/command_structure/auto_command.h"
00011
00017 class SpinRPMCommand: public AutoCommand {
00018 public:
00024     SpinRPMCommand(Flywheel &flywheel, int rpm);
00025
00031     bool run() override;
00032
00033 private:
00034     // Flywheel instance to run the function on
00035     Flywheel &flywheel;
00036
00037     // parameters for spinRPM
00038     int rpm;
00039 };
00040
00045 class WaitUntilUpToSpeedCommand: public AutoCommand {
00046 public:
00052     WaitUntilUpToSpeedCommand(Flywheel &flywheel, int threshold_rpm);
00053
00059     bool run() override;
00060
00061 private:
00062     // Flywheel instance to run the function on
00063     Flywheel &flywheel;
00064
00065     // if the actual speed is equal to the desired speed +/- this value, we are ready to fire
00066     int threshold_rpm;
00067 };
00068
00074 class FlywheelStopCommand: public AutoCommand {
00075 public:

```



```

00080     FlywheelStopCommand(Flywheel &flywheel);
00081
00082     bool run() override;
00083
00084 private:
00085     // Flywheel instance to run the function on
00086     Flywheel &flywheel;
00087 };
00088
00089 class FlywheelStopMotorsCommand: public AutoCommand {
00090 public:
00091     FlywheelStopMotorsCommand(Flywheel &flywheel);
00092
00093     bool run() override;
00094
00095 private:
00096     // Flywheel instance to run the function on
00097     Flywheel &flywheel;
00098 };
00099
00100 class FlywheelStopNonTasksCommand: public AutoCommand {
00101 public:
00102     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00103
00104     bool run() override;
00105
00106 private:
00107     // Flywheel instance to run the function on
00108     Flywheel &flywheel;
00109 };
00110
00111 class FlywheelStopNonTasksCommand: public AutoCommand {
00112 public:
00113     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00114
00115     bool run() override;
00116
00117 private:
00118     // Flywheel instance to run the function on
00119     Flywheel &flywheel;
00120 };
00121
00122 class FlywheelStopNonTasksCommand: public AutoCommand {
00123 public:
00124     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00125
00126     bool run() override;
00127
00128 private:
00129     // Flywheel instance to run the function on
00130     Flywheel &flywheel;
00131 };
00132
00133 class FlywheelStopNonTasksCommand: public AutoCommand {
00134 public:
00135     FlywheelStopNonTasksCommand(Flywheel &flywheel);
00136
00137     bool run() override;
00138
00139 private:
00140     // Flywheel instance to run the function on
00141     Flywheel &flywheel;
00142 };

```

## 6.17 feedback\_base.h

```

00001 #pragma once
00002
00003 class Feedback
00004 {
00005 public:
00006     enum FeedbackType
00007     {
00008         PIDType,
00009         FeedforwardType,
00010         OtherType,
00011     };
00012
00013     virtual void init(double start_pt, double set_pt) = 0;
00014
00015     virtual double update(double val) = 0;
00016
00017     virtual double get() = 0;
00018
00019     virtual void set_limits(double lower, double upper) = 0;
00020
00021     virtual bool is_on_target() = 0;
00022
00023     virtual FeedbackType get_type()
00024     {
00025         return FeedbackType::OtherType;
00026     }
00027 };
00028
00029 class Feedback
00030 {
00031 public:
00032     enum FeedbackType
00033     {
00034         PIDType,
00035         FeedforwardType,
00036         OtherType,
00037     };
00038
00039     virtual void init(double start_pt, double set_pt) = 0;
00040
00041     virtual double update(double val) = 0;
00042
00043     virtual double get() = 0;
00044
00045     virtual void set_limits(double lower, double upper) = 0;
00046
00047     virtual bool is_on_target() = 0;
00048
00049     virtual FeedbackType get_type()
00050     {
00051         return FeedbackType::OtherType;
00052     }
00053 };
00054
00055 class Feedback
00056 {
00057 public:
00058     enum FeedbackType
00059     {
00060         PIDType,
00061         FeedforwardType,
00062         OtherType,
00063     };
00064
00065     virtual void init(double start_pt, double set_pt) = 0;
00066
00067     virtual double update(double val) = 0;
00068
00069     virtual double get() = 0;
00070
00071     virtual void set_limits(double lower, double upper) = 0;
00072
00073     virtual bool is_on_target() = 0;
00074
00075     virtual FeedbackType get_type()
00076     {
00077         return FeedbackType::OtherType;
00078     }
00079 };
00080
00081 class Feedback
00082 {
00083 public:
00084     enum FeedbackType
00085     {
00086         PIDType,
00087         FeedforwardType,
00088         OtherType,
00089     };
00090
00091     virtual void init(double start_pt, double set_pt) = 0;
00092
00093     virtual double update(double val) = 0;
00094
00095     virtual double get() = 0;
00096
00097     virtual void set_limits(double lower, double upper) = 0;
00098
00099     virtual bool is_on_target() = 0;
00100
00101     virtual FeedbackType get_type()
00102     {
00103         return FeedbackType::OtherType;
00104     }
00105 };
00106
00107 class Feedback
00108 {
00109 public:
00110     enum FeedbackType
00111     {
00112         PIDType,
00113         FeedforwardType,
00114         OtherType,
00115     };
00116
00117     virtual void init(double start_pt, double set_pt) = 0;
00118
00119     virtual double update(double val) = 0;
00120
00121     virtual double get() = 0;
00122
00123     virtual void set_limits(double lower, double upper) = 0;
00124
00125     virtual bool is_on_target() = 0;
00126
00127     virtual FeedbackType get_type()
00128     {
00129         return FeedbackType::OtherType;
00130     }
00131 };
00132
00133 class Feedback
00134 {
00135 public:
00136     enum FeedbackType
00137     {
00138         PIDType,
00139         FeedforwardType,
00140         OtherType,
00141     };
00142
00143     virtual void init(double start_pt, double set_pt) = 0;
00144
00145     virtual double update(double val) = 0;
00146
00147     virtual double get() = 0;
00148
00149     virtual void set_limits(double lower, double upper) = 0;
00150
00151     virtual bool is_on_target() = 0;
00152
00153     virtual FeedbackType get_type()
00154     {
00155         return FeedbackType::OtherType;
00156     }
00157 };
00158
00159 class Feedback
00160 {
00161 public:
00162     enum FeedbackType
00163     {
00164         PIDType,
00165         FeedforwardType,
00166         OtherType,
00167     };
00168
00169     virtual void init(double start_pt, double set_pt) = 0;
00170
00171     virtual double update(double val) = 0;
00172
00173     virtual double get() = 0;
00174
00175     virtual void set_limits(double lower, double upper) = 0;
00176
00177     virtual bool is_on_target() = 0;
00178
00179     virtual FeedbackType get_type()
00180     {
00181         return FeedbackType::OtherType;
00182     }
00183 };
00184
00185 class Feedback
00186 {
00187 public:
00188     enum FeedbackType
00189     {
00190         PIDType,
00191         FeedforwardType,
00192         OtherType,
00193     };
00194
00195     virtual void init(double start_pt, double set_pt) = 0;
00196
00197     virtual double update(double val) = 0;
00198
00199     virtual double get() = 0;
00200
00201     virtual void set_limits(double lower, double upper) = 0;
00202
00203     virtual bool is_on_target() = 0;
00204
00205     virtual FeedbackType get_type()
00206     {
00207         return FeedbackType::OtherType;
00208     }
00209 };
00210
00211 class Feedback
00212 {
00213 public:
00214     enum FeedbackType
00215     {
00216         PIDType,
00217         FeedforwardType,
00218         OtherType,
00219     };
00220
00221     virtual void init(double start_pt, double set_pt) = 0;
00222
00223     virtual double update(double val) = 0;
00224
00225     virtual double get() = 0;
00226
00227     virtual void set_limits(double lower, double upper) = 0;
00228
00229     virtual bool is_on_target() = 0;
00230
00231     virtual FeedbackType get_type()
00232     {
00233         return FeedbackType::OtherType;
00234     }
00235 };
00236
00237 class Feedback
00238 {
00239 public:
00240     enum FeedbackType
00241     {
00242         PIDType,
00243         FeedforwardType,
00244         OtherType,
00245     };
00246
00247     virtual void init(double start_pt, double set_pt) = 0;
00248
00249     virtual double update(double val) = 0;
00250
00251     virtual double get() = 0;
00252
00253     virtual void set_limits(double lower, double upper) = 0;
00254
00255     virtual bool is_on_target() = 0;
00256
00257     virtual FeedbackType get_type()
00258     {
00259         return FeedbackType::OtherType;
00260     }
00261 };
00262
00263 class Feedback
00264 {
00265 public:
00266     enum FeedbackType
00267     {
00268         PIDType,
00269         FeedforwardType,
00270         OtherType,
00271     };
00272
00273     virtual void init(double start_pt, double set_pt) = 0;
00274
00275     virtual double update(double val) = 0;
00276
00277     virtual double get() = 0;
00278
00279     virtual void set_limits(double lower, double upper) = 0;
00280
00281     virtual bool is_on_target() = 0;
00282
00283     virtual FeedbackType get_type()
00284     {
00285         return FeedbackType::OtherType;
00286     }
00287 };
00288
00289 class Feedback
00290 {
00291 public:
00292     enum FeedbackType
00293     {
00294         PIDType,
00295         FeedforwardType,
00296         OtherType,
00297     };
00298
00299     virtual void init(double start_pt, double set_pt) = 0;
00300
00301     virtual double update(double val) = 0;
00302
00303     virtual double get() = 0;
00304
00305     virtual void set_limits(double lower, double upper) = 0;
00306
00307     virtual bool is_on_target() = 0;
00308
00309     virtual FeedbackType get_type()
00310     {
00311         return FeedbackType::OtherType;
00312     }
00313 };
00314
00315 class Feedback
00316 {
00317 public:
00318     enum FeedbackType
00319     {
00320         PIDType,
00321         FeedforwardType,
00322         OtherType,
00323     };
00324
00325     virtual void init(double start_pt, double set_pt) = 0;
00326
00327     virtual double update(double val) = 0;
00328
00329     virtual double get() = 0;
00330
00331     virtual void set_limits(double lower, double upper) = 0;
00332
00333     virtual bool is_on_target() = 0;
00334
00335     virtual FeedbackType get_type()
00336     {
00337         return FeedbackType::OtherType;
00338     }
00339 };
00340
00341 class Feedback
00342 {
00343 public:
00344     enum FeedbackType
00345     {
00346         PIDType,
00347         FeedforwardType,
00348         OtherType,
00349     };
00350
00351     virtual void init(double start_pt, double set_pt) = 0;
00352
00353     virtual double update(double val) = 0;
00354
00355     virtual double get() = 0;
00356
00357     virtual void set_limits(double lower, double upper) = 0;
00358
00359     virtual bool is_on_target() = 0;
00360
00361     virtual FeedbackType get_type()
00362     {
00363         return FeedbackType::OtherType;
00364     }
00365 };
00366
00367 class Feedback
00368 {
00369 public:
00370     enum FeedbackType
00371     {
00372         PIDType,
00373         FeedforwardType,
00374         OtherType,
00375     };
00376
00377     virtual void init(double start_pt, double set_pt) = 0;
00378
00379     virtual double update(double val) = 0;
00380
00381     virtual double get() = 0;
00382
00383     virtual void set_limits(double lower, double upper) = 0;
00384
00385     virtual bool is_on_target() = 0;
00386
00387     virtual FeedbackType get_type()
00388     {
00389         return FeedbackType::OtherType;
00390     }
00391 };
00392
00393 class Feedback
00394 {
00395 public:
00396     enum FeedbackType
00397     {
00398         PIDType,
00399         FeedforwardType,
00400         OtherType,
00401     };
00402
00403     virtual void init(double start_pt, double set_pt) = 0;
00404
00405     virtual double update(double val) = 0;
00406
00407     virtual double get() = 0;
00408
00409     virtual void set_limits(double lower, double upper) = 0;
00410
00411     virtual bool is_on_target() = 0;
00412
00413     virtual FeedbackType get_type()
00414     {
00415         return FeedbackType::OtherType;
00416     }
00417 };
00418
00419 class Feedback
00420 {
00421 public:
00422     enum FeedbackType
00423     {
00424         PIDType,
00425         FeedforwardType,
00426         OtherType,
00427     };
00428
00429     virtual void init(double start_pt, double set_pt) = 0;
00430
00431     virtual double update(double val) = 0;
00432
00433     virtual double get() = 0;
00434
00435     virtual void set_limits(double lower, double upper) = 0;
00436
00437     virtual bool is_on_target() = 0;
00438
00439     virtual FeedbackType get_type()
00440     {
00441         return FeedbackType::OtherType;
00442     }
00443 };
00444
00445 class Feedback
00446 {
00447 public:
00448     enum FeedbackType
00449     {
00450         PIDType,
00451         FeedforwardType,
00452         OtherType,
00453     };
00454
00455     virtual void init(double start_pt, double set_pt) = 0;
00456
00457     virtual double update(double val) = 0;
00458
00459     virtual double get() = 0;
00460
00461     virtual void set_limits(double lower, double upper) = 0;
00462
00463     virtual bool is_on_target() = 0;
00464
00465     virtual FeedbackType get_type()
00466     {
00467         return FeedbackType::OtherType;
00468     }
00469 };
00470
00471 class Feedback
00472 {
00473 public:
00474     enum FeedbackType
00475     {
00476         PIDType,
00477         FeedforwardType,
00478         OtherType,
00479     };
00480
00481     virtual void init(double start_pt, double set_pt) = 0;
00482
00483     virtual double update(double val) = 0;
00484
00485     virtual double get() = 0;
00486
00487     virtual void set_limits(double lower, double upper) = 0;
00488
00489     virtual bool is_on_target() = 0;
00490
00491     virtual FeedbackType get_type()
00492     {
00493         return FeedbackType::OtherType;
00494     }
00495 };
00496
00497 class Feedback
00498 {
00499 public:
00500     enum FeedbackType
00501     {
00502         PIDType,
00503         FeedforwardType,
00504         OtherType,
00505     };
00506
00507     virtual void init(double start_pt, double set_pt) = 0;
00508
00509     virtual double update(double val) = 0;
00510
00511     virtual double get() = 0;
00512
00513     virtual void set_limits(double lower, double upper) = 0;
00514
00515     virtual bool is_on_target() = 0;
00516
00517     virtual FeedbackType get_type()
00518     {
00519         return FeedbackType::OtherType;
00520     }
00521 };
00522
00523 class Feedback
00524 {
00525 public:
00526     enum FeedbackType
00527     {
00528         PIDType,
00529         FeedforwardType,
00530         OtherType,
00531     };
00532
00533     virtual void init(double start_pt, double set_pt) = 0;
00534
00535     virtual double update(double val) = 0;
00536
00537     virtual double get() = 0;
00538
00539     virtual void set_limits(double lower, double upper) = 0;
00540
00541     virtual bool is_on_target() = 0;
00542
00543     virtual FeedbackType get_type()
00544     {
00545         return FeedbackType::OtherType;
00546     }
00547 };
00548
00549 class Feedback
00550 {
00551 public:
00552     enum FeedbackType
00553     {
00554         PIDType,
00555         FeedforwardType,
00556         OtherType,
00557     };
00558
00559     virtual void init(double start_pt, double set_pt) = 0;
00560
00561     virtual double update(double val) = 0;
00562
00563     virtual double get() = 0;
00564
00565     virtual void set_limits(double lower, double upper) = 0;
00566
00567     virtual bool is_on_target() = 0;
00568
00569     virtual FeedbackType get_type()
00570     {
00571         return FeedbackType::OtherType;
00572     }
00573 };
00574
00575 class Feedback
00576 {
00577 public:
00578     enum FeedbackType
00579     {
00580         PIDType,
00581         FeedforwardType,
00582         OtherType,
00583     };
00584
00585     virtual void init(double start_pt, double set_pt) = 0;
00586
00587     virtual double update(double val) = 0;
00588
00589     virtual double get() = 0;
00590
00591     virtual void set_limits(double lower, double upper) = 0;
00592
00593     virtual bool is_on_target() = 0;
00594
00595     virtual FeedbackType get_type()
00596     {
00597         return FeedbackType::OtherType;
00598     }
00599 };
00600
00601 class Feedback
00602 {
00603 public:
00604     enum FeedbackType
00605     {
00606         PIDType,
00607         FeedforwardType,
00608         OtherType,
00609     };
00610
00611     virtual void init(double start_pt, double set_pt) = 0;
00612
00613     virtual double update(double val) = 0;
00614
00615     virtual double get() = 0;
00616
00617     virtual void set_limits(double lower, double upper) = 0;
00618
00619     virtual bool is_on_target() = 0;
00620
00621     virtual FeedbackType get_type()
00622     {
00623         return FeedbackType::OtherType;
00624     }
00625 };
00626
00627 class Feedback
00628 {
00629 public:
00630     enum FeedbackType
00631     {
00632         PIDType,
00633         FeedforwardType,
00634         OtherType,
00635     };
00636
00637     virtual void init(double start_pt, double set_pt) = 0;
00638
00639     virtual double update(double val) = 0;
00640
00641     virtual double get() = 0;
00642
00643     virtual void set_limits(double lower, double upper) = 0;
00644
00645     virtual bool is_on_target() = 0;
00646
00647     virtual FeedbackType get_type()
00648     {
00649         return FeedbackType::OtherType;
00650     }
00651 };
00652
00653 class Feedback
00654 {
00655 public:
00656     enum FeedbackType
00657     {
00658         PIDType,
00659         FeedforwardType,
00660         OtherType,
00661     };
00662
00663     virtual void init(double start_pt, double set_pt) = 0;
00664
00665     virtual double update(double val) = 0;
00666
00667     virtual double get() = 0;
00668
00669     virtual void set_limits(double lower, double upper) = 0;
00670
00671     virtual bool is_on_target() = 0;
00672
00673     virtual FeedbackType get_type()
00674     {
00675         return FeedbackType::OtherType;
00676     }
00677 };
00678
00679 class Feedback
00680 {
00681 public:
00682     enum FeedbackType
00683     {
00684         PIDType,
00685         FeedforwardType,
00686         OtherType,
00687     };
00688
00689     virtual void init(double start_pt, double set_pt) = 0;
00690
00691     virtual double update(double val) = 0;
00692
00693     virtual double get() = 0;
00694
00695     virtual void set_limits(double lower, double upper) = 0;
00696
00697     virtual bool is_on_target() = 0;
00698
00699     virtual FeedbackType get_type()
00700     {
00701         return FeedbackType::OtherType;
00702     }
00703 };
00704
00705 class Feedback
00706 {
00707 public:
00708     enum FeedbackType
00709     {
00710         PIDType,
00711         FeedforwardType,
00712         OtherType,
00713     };
00714
00715     virtual void init(double start_pt, double set_pt) = 0;
00716
00717     virtual double update(double val) = 0;
00718
00719     virtual double get() = 0;
00720
00721     virtual void set_limits(double lower, double upper) = 0;
00722
00723     virtual bool is_on_target() = 0;
00724
00725     virtual FeedbackType get_type()
00726     {
00727         return FeedbackType::OtherType;
00728     }
00729 };
00730
00731 class Feedback
00732 {
00733 public:
00734     enum FeedbackType
00735     {
00736         PIDType,
00737         FeedforwardType,
00738         OtherType,
00739     };
00740
00741     virtual void init(double start_pt, double set_pt) = 0;
00742
00743     virtual double update(double val) = 0;
00744
00745     virtual double get() = 0;
00746
00747     virtual void set_limits(double lower, double upper) = 0;
00748
00749     virtual bool is_on_target() = 0;
00750
00751     virtual FeedbackType get_type()
00752     {
00753         return FeedbackType::OtherType;
00754     }
00755 };
00756
00757 class Feedback
00758 {
00759 public:
00760     enum FeedbackType
00761     {
00762         PIDType,
00763         FeedforwardType,
00764         OtherType,
00765     };
00766
00767     virtual void init(double start_pt, double set_pt) = 0;
00768
00769     virtual double update(double val) = 0;
00770
00771     virtual double get() = 0;
00772
00773     virtual void set_limits(double lower, double upper) = 0;
00774
00775     virtual bool is_on_target() = 0;
00776
00777     virtual FeedbackType get_type()
00778     {
00779         return FeedbackType::OtherType;
00780     }
00781 };
00782
00783 class Feedback
00784 {
00785 public:
00786     enum FeedbackType
00787     {
00788         PIDType,
00789         FeedforwardType,
00790         OtherType,
00791     };
00792
00793     virtual void init(double start_pt, double set_pt) = 0;
00794
00795     virtual double update(double val) = 0;
00796
00797     virtual double get() = 0;
00798
00799     virtual void set_limits(double lower, double upper) = 0;
00800
00801     virtual bool is_on_target() = 0;
00802
00803     virtual FeedbackType get_type()
00804     {
00805         return FeedbackType::OtherType;
00806     }
00807 };
00808
00809 class Feedback
00810 {
00811 public:
00812     enum FeedbackType
00813     {
00814         PIDType,
00815         FeedforwardType,
00816         OtherType,
00817     };
00818
00819     virtual void init(double start_pt, double set_pt) = 0;
00820
00821     virtual double update(double val) = 0;
00822
00823     virtual double get() = 0;
00824
00825     virtual void set_limits(double lower, double upper) = 0;
00826
00827     virtual bool is_on_target() = 0;
00828
00829     virtual FeedbackType get_type()
00830     {
00831         return FeedbackType::OtherType;
00832     }
00833 };
00834
00835 class Feedback
00836 {
00837 public:
00838     enum FeedbackType
00839     {
00840         PIDType,
00841         FeedforwardType,
00842         OtherType,
00843     };
00844
00845     virtual void init(double start_pt, double set_pt) = 0;
00846
00847     virtual double update(double val) = 0;
00848
00849     virtual double get() = 0;
00850
00851     virtual void set_limits(double lower, double upper) = 0;
00852
00853     virtual bool is_on_target() = 0;
00854
00855     virtual FeedbackType get_type()
00856     {
00857         return FeedbackType::OtherType;
00858     }
00859 };
00860
00861 class Feedback
00862 {
00863 public:
00864     enum FeedbackType
00865     {
00866         PIDType,
00867         FeedforwardType,
00868         OtherType,
00869     };
00870
00871     virtual void init(double start_pt, double set_pt) = 0;
00872
00873     virtual double update(double val) = 0;
00874
00875     virtual double get() = 0;
00876
00877     virtual void set_limits(double lower, double upper) = 0;
00878
00879     virtual bool is_on_target() = 0;
00880
00881     virtual FeedbackType get_type()
00882     {
00883         return FeedbackType::OtherType;
00884     }
00885 };
00886
00887 class Feedback
00888 {
00889 public:
00890     enum FeedbackType
00891     {
00892         PIDType,
00893         FeedforwardType,
00894         OtherType,
00895     };
00896
00897     virtual void init(double start_pt, double set_pt) = 0;
00898
00899     virtual double update(double val) = 0;
00900
00901     virtual double get() = 0;
00902
00903     virtual void set_limits(double lower, double upper) = 0;
00904
00905     virtual bool is_on_target() = 0;
00906
00907     virtual FeedbackType get_type()
00908     {
00909         return FeedbackType::OtherType;
00910     }
00911 };
00912
00913 class Feedback
00914 {
00915 public:
00916     enum FeedbackType
00917     {
00918         PIDType,
00919         FeedforwardType,
00920         OtherType,
00921     };
00922
00923     virtual void init(double start_pt, double set_pt) = 0;
00924
00925     virtual double update(double val) = 0;
00926
00927     virtual double get() = 0;
00928
00929     virtual void set_limits(double lower, double upper) = 0;
00930
00931     virtual bool is_on_target() = 0;
00932
00933     virtual FeedbackType get_type()
00934     {
00935         return FeedbackType::OtherType;
00936     }
00937 };
00938
00939 class Feedback
00940 {
00941 public:
00942     enum FeedbackType
00943     {
00944         PIDType,
00945         FeedforwardType,
00946         OtherType,
00947     };
00948
00949     virtual void init(double start_pt, double set_pt) = 0;
00950
00951     virtual double update(double val) = 0;
00952
00953     virtual double get() = 0;
00954
00955     virtual void set_limits(double lower, double upper) = 0;
00956
00957     virtual bool is_on_target() = 0;
00958
00959     virtual FeedbackType get_type()
00960     {
00961         return FeedbackType::OtherType;
00962     }
00963 };
00964
00965 class Feedback
00966 {
00967 public:
00968     enum FeedbackType
00969     {
00970         PIDType,
00971         FeedforwardType,
00972         OtherType,
00973     };
00974
00975     virtual void init(double start_pt, double set_pt) = 0;
00976
00977     virtual double update(double val) = 0;
00978
00979     virtual double get() = 0;
00980
00981     virtual void set_limits(double lower, double upper) = 0;
00982
00983     virtual bool is_on_target() = 0;
00984
00985     virtual FeedbackType get_type()
00986     {
00987         return FeedbackType::OtherType;
00988     }
00989 };
00990
00991 class Feedback
00992 {
00993 public:
00994     enum FeedbackType
00995     {
00996         PIDType,
00997         FeedforwardType,
00998         OtherType,
00999     };
01000
01001     virtual void init(double start_pt, double set_pt) = 0;
01002
01003     virtual double update(double val) = 0;
01004
01005     virtual double get() = 0;
01006
01007     virtual void set_limits(double lower, double upper) = 0;
01008
01009     virtual bool is_on_target() = 0;
01010
01011     virtual FeedbackType get_type()
01012     {
01013         return FeedbackType::OtherType;
01014     }
01015 };
01016
01017 class Feedback
01018 {
01019 public:
01020     enum FeedbackType
01021     {
01022         PIDType,
01023         FeedforwardType,
01024         OtherType,
01025     };
01026
01027     virtual void init(double start_pt, double set_pt) = 0;
01028
01029     virtual double update(double val) = 0;
01030
01031     virtual double get() = 0;
01032
01033     virtual void set_limits(double lower, double upper) = 0;
01034
01035     virtual bool is_on_target() = 0;
01036
01037     virtual FeedbackType get_type()
01038     {
01039         return FeedbackType::OtherType;
01040     }
01041 };
01042
01043 class Feedback
01044 {
01045 public:
01046     enum FeedbackType
01047     {
01048         PIDType,
01049         FeedforwardType,
01050         OtherType,
01051     };
01052
01053     virtual void init(double start_pt, double set_pt) = 0;
01054
01055     virtual double update(double val) = 0;
01056
01057     virtual double get() = 0;
01058
01059     virtual void set_limits(double lower, double upper) = 0;
01060
01061     virtual bool is_on_target() = 0;
01062
01063     virtual FeedbackType get_type()
01064     {
01065         return FeedbackType::OtherType;
01066     }
01067 };
01068
01069 class Feedback
01070 {
01071 public:
01072     enum FeedbackType
01073     {
01074         PIDType,
01075         FeedforwardType,
01076         OtherType,
01077     };
01078
01079     virtual void init(double start_pt, double set_pt) = 0;
01080
01081     virtual double update(double val) = 0;
01082
01083     virtual double get() = 0;
01084
01085     virtual void set_limits(double lower, double upper) = 0;
01086
01087     virtual bool is_on_target() = 0;
01088
01089     virtual FeedbackType get_type()
01090     {
01091         return FeedbackType::OtherType;
01092     }
01093 };
01094
01095 class Feedback
01096 {
01097 public:
01098     enum FeedbackType
01099     {
01100         PIDType,
01101         FeedforwardType,
01102         OtherType,
01103     };
01104
01105     virtual void init(double start_pt, double set_pt) = 0;
01106
01107     virtual double update(double val) = 0;
01108
01109     virtual double get() = 0;
01110
01111     virtual void set_limits(double lower, double upper) = 0;
01112
01113     virtual bool is_on_target() = 0;
01114
01115     virtual FeedbackType get_type()
01116     {
01117         return FeedbackType::OtherType;
01118     }
01119 };
01120
01121 class Feedback
01122 {
01123 public:
01124     enum FeedbackType
01125     {
01126         PIDType,
01127         FeedforwardType,
01128         OtherType,
01129     };
01130
01131     virtual void init(double start_pt, double set_pt) = 0;
01132
01133     virtual double
```

```

00046     double kG;
00047 } ff_config_t;
00048
00049
00054 FeedForward(ff_config_t &cfg) : cfg(cfg) {}
00055
00066 double calculate(double v, double a, double pid_ref=0.0)
00067 {
00068     double ks_sign = 0;
00069     if(v != 0)
00070         ks_sign = sign(v);
00071     else if(pid_ref != 0)
00072         ks_sign = sign(pid_ref);
00073
00074     return (cfg.kS * ks_sign) + (cfg.kV * v) + (cfg.kA * a) + cfg.kG;
00075 }
00076
00077 private:
00078
00079     ff_config_t &cfg;
00080
00081 };
00082
00083
00091 FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);

```

## 6.19 generic\_auto.h

```

00001 #pragma once
00002
00003 #include <queue>
00004 #include <map>
00005 #include "vex.h"
00006 #include <functional>
00007
00008 typedef std::function<bool(void)> state_ptr;
00009
00014 class GenericAuto
00015 {
00016     public:
00017
00031     bool run(bool blocking);
00032
00037     void add(state_ptr new_state);
00038
00043     void add_async(state_ptr async_state);
00044
00049     void add_delay(int ms);
00050
00051     private:
00052
00053     std::queue<state_ptr> state_list;
00054
00055 };

```

## 6.20 geometry.h

```

00001 #pragma once
00002 #include <cmath>
00003
00007 struct point_t
00008 {
00009     double x;
00010     double y;
00011
00017     double dist(const point_t other)
00018     {
00019         return std::sqrt(std::pow(this->x - other.x, 2) + pow(this->y - other.y, 2));
00020     }
00021
00027     point_t operator+(const point_t &other)
00028     {
00029         point_t p{
00030             .x = this->x + other.x,
00031             .y = this->y + other.y};
00032         return p;
00033     }
00034
00040     point_t operator-(const point_t &other)

```

```

00041     {
00042         point_t p{
00043             .x = this->x - other.x,
00044             .y = this->y - other.y};
00045         return p;
00046     }
00047 };
00048
00049
00053 typedef struct
00054 {
00055     double x;
00056     double y;
00057     double rot;
00058 } pose_t;

```

## 6.21 graph\_drawer.h

```

00001 #pragma once
00002
00003 #include <string>
00004 #include <stdio.h>
00005 #include <vector>
00006 #include <cmath>
00007 #include "vex.h"
00008 #include "../core/include/utils/geometry.h"
00009 #include "../core/include/utils/vector2d.h"
00010
00011 class GraphDrawer
00012 {
00013 public:
00025     GraphDrawer(vex::brain::lcd &screen, int num_samples, std::string x_label, std::string y_label,
vex::color col, bool draw_border, double lower_bound, double upper_bound);
00030     void add_sample(point_t sample);
00038     void draw(int x, int y, int width, int height);
00039
00040 private:
00041     vex::brain::lcd &Screen;
00042     std::vector<point_t> samples;
00043     int sample_index = 0;
00044     std::string xlabel;
00045     std::string ylabel;
00046     vex::color col = vex::red;
00047     vex::color bgcol = vex::transparent;
00048     bool border;
00049     double upper;
00050     double lower;
00051 };

```

## 6.22 logger.h

```

00001 #pragma once
00002
00003 #include <cstdint>
00004 #include <cstdio>
00005 #include <string>
00006 #include "vex.h"
00007
00009 enum LogLevel
00010 {
00011     DEBUG,
00012     NOTICE,
00013     WARNING,
00014     ERROR,
00015     CRITICAL,
00016     TIME
00017 };
00018
00020 class Logger
00021 {
00022 private:
00023     const std::string filename;
00024     vex::brain::sdcard sd;
00025     void write_level(LogLevel l);
00026
00027 public:
00029     const int MAX_FORMAT_LEN = 512;
00032     explicit Logger(const std::string &filename);
00033

```

```

00035     Logger(const Logger &l) = delete;
00037     Logger &operator=(const Logger &l) = delete;
00038
00039
00042     void Log(const std::string &s);
00043
00047     void Log(LogLevel level, const std::string &s);
00048
00051     void LogIn(const std::string &s);
00052
00056     void LogIn(LogLevel level, const std::string &s);
00057
00061     void Logf(const char *fmt, ...);
00062
00067     void Logf(LogLevel level, const char *fmt, ...);
00068 };

```

## 6.23 math\_util.h

```

00001 #pragma once
00002 #include "math.h"
00003 #include "vex.h"
00004 #include <vector>
00005
00013 double clamp(double value, double low, double high);
00014
00021 double sign(double x);
00022
00023 double wrap_angle_deg(double input);
00024 double wrap_angle_rad(double input);
00025
00026 /*
00027 Calculates the variance of a set of numbers (needed for linear regression)
00028 https://en.wikipedia.org/wiki/Variance
00029 @param values the values for which the variance is taken
00030 @param mean the average of values
00031 */
00032 double variance(std::vector<double> const &values, double mean);
00033
00034
00035 /*
00036 Calculates the average of a vector of doubles
00037 @param values the list of values for which the average is taken
00038 */
00039 double mean(std::vector<double> const &values);
00040
00041 /*
00042 Calculates the covariance of a set of points (needed for linear regression)
00043 https://en.wikipedia.org/wiki/Covariance
00044
00045 @param points the points for which the covariance is taken
00046 @param meanx the mean value of all x coordinates in points
00047 @param meany the mean value of all y coordinates in points
00048 */
00049 double covariance(std::vector<std::pair<double, double> const &points, double meanx, double meany);
00050
00051 /*
00052 Calculates the slope and y intercept of the line of best fit for the data
00053 @param points the points for the data
00054 */
00055 std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double> const
00056 &points);

```

## 6.24 motion\_controller.h

```

00001 #pragma once
00002 #include "../core/include/utils/pid.h"
00003 #include "../core/include/utils/feedforward.h"
00004 #include "../core/include/utils/trapezoid_profile.h"
00005 #include "../core/include/utils/feedback_base.h"
00006 #include "../core/include/subsystems/tank_drive.h"
00007 #include "vex.h"
00008
00025 class MotionController : public Feedback
00026 {
00027     public:
00028
00034     typedef struct

```

```

00035     {
00036         double max_v;
00037         double accel;
00038         PID::pid_config_t pid_cfg;
00039         FeedForward::ff_config_t ff_cfg;
00040     } m_profile_cfg_t;
00041
00051     MotionController(m_profile_cfg_t &config);
00052
00057     void init(double start_pt, double end_pt) override;
00058
00065     double update(double sensor_val) override;
00066
00070     double get() override;
00071
00079     void set_limits(double lower, double upper) override;
00080
00085     bool is_on_target() override;
00086
00090     motion_t get_motion();
00091
00110     static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
pct=0.6, double duration=2);
00111
00112     private:
00113
00114     m_profile_cfg_t config;
00115
00116     PID pid;
00117     FeedForward ff;
00118     TrapezoidProfile profile;
00119
00120     double lower_limit = 0, upper_limit = 0;
00121     double out = 0;
00122     motion_t cur_motion;
00123
00124     vex::timer tmr;
00125
00126 };

```

## 6.25 moving\_average.h

```

00001 #include <vector>
00002
00015 class MovingAverage {
00016 public:
00017     /*
00018      * Create a moving average calculator with 0 as the default value
00019      *
00020      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
reading
00021      */
00022     MovingAverage(int buffer_size);
00023     /*
00024      * Create a moving average calculator with a specified default value
00025      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
reading
00026      * @param starting_value The value that the average will be before any data is added
00027      */
00028     MovingAverage(int buffer_size, double starting_value);
00029
00030     /*
00031      * Add a reading to the buffer
00032      * Before:
00033      * [ 1 1 2 2 3 3] => 2
00034      * ^
00035      * After:
00036      * [ 2 1 2 2 3 3] => 2.16
00037      * ^
00038      * @param n    the sample that will be added to the moving average.
00039      */
00040     void add_entry(double n);
00041
00046     double get_average();
00047
00052     int get_size();
00053
00054
00055     private:
00056         int buffer_index;                //index of the next value to be overridden
00057         std::vector<double> buffer;      //all current data readings we've taken
00058         double current_avg;             //the current value of the data
00059
00060 };

```

## 6.26 pid.h

```

00001 #pragma once
00002
00003 #include <cmath>
00004 #include "vex.h"
00005 #include "../core/include/utils/feedback_base.h"
00006
00007 using namespace vex;
00008
00023 class PID : public Feedback
00024 {
00025 public:
00029     enum ERROR_TYPE{
00030         LINEAR,
00031         ANGULAR // assumes degrees
00032     };
00040     struct pid_config_t
00041     {
00042         double p;
00043         double i;
00044         double d;
00045         double deadband;
00046         double on_target_time;
00047         ERROR_TYPE error_method;
00048     };
00049
00050
00051
00056     PID(pid_config_t &config);
00057
00058
00067     void init(double start_pt, double set_pt) override;
00068
00075     double update(double sensor_val) override;
00076
00081     double get() override;
00082
00089     void set_limits(double lower, double upper) override;
00090
00095     bool is_on_target() override;
00096
00100     void reset();
00101
00106     double get_error();
00107
00112     double get_target();
00113
00118     void set_target(double target);
00119
00120     Feedback::FeedbackType get_type() override;
00121
00122     pid_config_t &config;
00123
00124 private:
00125
00126
00127     double last_error = 0;
00128     double accum_error = 0;
00129
00130     double last_time = 0;
00131     double on_target_last_time = 0;
00132
00133     double lower_limit = 0;
00134     double upper_limit = 0;
00135
00136     double target = 0;
00137     double sensor_val = 0;
00138     double out = 0;
00139
00140     bool is_checking_on_target = false;
00141
00142     timer pid_timer;
00143 };

```

## 6.27 pidff.h

```

00001 #pragma once
00002 #include "../core/include/utils/feedback_base.h"
00003 #include "../core/include/utils/pid.h"
00004 #include "../core/include/utils/feedforward.h"
00005
00006 class PIDFF : public Feedback

```

```

00007 {
00008     public:
00009
00010         PIDFF(PID::pid_config_t &pid_cfg, FeedForward::ff_config_t &ff_cfg);
00011
00018         void init(double start_pt, double set_pt) override;
00019
00024         void set_target(double set_pt);
00025
00033         double update(double val) override;
00034
00043         double update(double val, double vel_setpt, double a_setpt=0);
00044
00048         double get() override;
00049
00056         void set_limits(double lower, double upper) override;
00057
00061         bool is_on_target() override;
00062
00063         PID pid;
00064
00065     private:
00066
00067         FeedForward::ff_config_t &ff_cfg;
00069         FeedForward ff;
00070
00071         double out;
00072         double lower_lim, upper_lim;
00073
00074 };
00075 };

```

## 6.28 pure\_pursuit.h

```

00001 #pragma once
00002
00003 #include <vector>
00004 #include "../core/include/utils/geometry.h"
00005 #include "../core/include/utils/vector2d.h"
00006 #include "vex.h"
00007
00008 using namespace vex;
00009
00010 namespace PurePursuit {
00015     struct spline
00016     {
00017         double a, b, c, d, x_start, x_end;
00018
00019         double getY(double x) {
00020             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00021         }
00022     };
00027     struct hermite_point
00028     {
00029         double x;
00030         double y;
00031         double dir;
00032         double mag;
00033
00034         point_t getPoint() {
00035             return {x, y};
00036         }
00037
00038         Vector2D getTangent() {
00039             return Vector2D(dir, mag);
00040         }
00041     };
00042
00047     static std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
point_t point2);
00051     static point_t get_lookahead(std::vector<point_t> path, point_t robot_loc, double radius);
00052
00056     static std::vector<point_t> inject_path(std::vector<point_t> path, double spacing);
00057
00069     static std::vector<point_t> smooth_path(std::vector<point_t> path, double weight_data, double
weight_smooth, double tolerance);
00070
00071     static std::vector<point_t> smooth_path_cubic(std::vector<point_t> path, double res);
00072
00081     static std::vector<point_t> smooth_path_hermite(std::vector<hermite_point> path, double step);
00082 }

```

## 6.29 serializer.h

```

00001 #pragma once
00002 #include <algorithm>
00003 #include <map>
00004 #include <string>
00005 #include <vector>
00006 #include <stdio.h>
00007
00009 const char serialization_separator = '$';
00011 const std::size_t MAX_FILE_SIZE = 4096;
00012
00014 class Serializer
00015 {
00016 private:
00017     bool flush_always;
00018     std::string filename;
00019     std::map<std::string, int> ints;
00020     std::map<std::string, bool> bools;
00021     std::map<std::string, double> doubles;
00022     std::map<std::string, std::string> strings;
00023
00025     bool read_from_disk();
00026
00027 public:
00029     ~Serializer()
00030     {
00031         save_to_disk();
00032         printf("Saving %s\n", filename.c_str());
00033         fflush(stdout);
00034     }
00035
00039     explicit Serializer(const std::string &filename, bool flush_always = true) :
flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}), strings({}) {
read_from_disk(); }
00040
00042     void save_to_disk() const;
00043
00045
00049     void set_int(const std::string &name, int i);
00050
00054     void set_bool(const std::string &name, bool b);
00055
00059     void set_double(const std::string &name, double d);
00060
00064     void set_string(const std::string &name, std::string str);
00065
00068
00073     int int_or(const std::string &name, int otherwise);
00074
00079     bool bool_or(const std::string &name, bool otherwise);
00080
00085     double double_or(const std::string &name, double otherwise);
00086
00091     std::string string_or(const std::string &name, std::string otherwise);
00092 };

```

## 6.30 trapezoid\_profile.h

```

00001 #pragma once
00002
00006 typedef struct
00007 {
00008     double pos;
00009     double vel;
00010     double accel;
00011 } motion_t;
00012
00013
00034 class TrapezoidProfile
00035 {
00036 public:
00037
00044     TrapezoidProfile(double max_v, double accel);
00045
00052     motion_t calculate(double time_s);
00053
00059     void set_endpts(double start, double end);
00060
00065     void set_accel(double accel);
00066
00072     void set_max_v(double max_v);
00073

```



```
00078     double get_movement_time();
00079
00080     private:
00081     double start, end;
00082     double max_v;
00083     double accel;
00084     double time;
00085
00086
00087 };
```

## 6.31 vector2d.h

```
00001 #pragma once
00002
00003
00004 #include <cmath>
00005 #include "../core/include/utils/geometry.h"
00006
00007 #ifndef PI
00008 #define PI 3.141592654
00009 #endif
00010
00011 class Vector2D
00012 {
00013 public:
00014     Vector2D(double dir, double mag);
00015
00016     Vector2D(point_t p);
00017
00018     double get_dir() const;
00019
00020     double get_mag() const;
00021
00022     double get_x() const;
00023
00024     double get_y() const;
00025
00026     Vector2D normalize();
00027
00028     point_t point();
00029
00030     Vector2D operator*(const double &x);
00031     Vector2D operator+(const Vector2D &other);
00032     Vector2D operator-(const Vector2D &other);
00033
00034 private:
00035     double dir, mag;
00036 };
00037
00038 double deg2rad(double deg);
00039
00040 double rad2deg(double r);
```



# Index

- accel
  - OdometryBase, 74
- add
  - AutoChooser, 10
  - CommandController, 14, 15
  - GenericAuto, 42
- add\_async
  - GenericAuto, 42
- add\_delay
  - CommandController, 15
  - GenericAuto, 43
- add\_entry
  - MovingAverage, 64
- add\_sample
  - GraphDrawer, 44
- ang\_accel\_deg
  - OdometryBase, 74
- ang\_speed\_deg
  - OdometryBase, 74
- auto\_drive
  - MecanumDrive, 55
- auto\_turn
  - MecanumDrive, 56
- AutoChooser, 9
  - add, 10
  - AutoChooser, 9
  - brain, 11
  - choice, 11
  - get\_choice, 10
  - list, 11
  - render, 10
- AutoChooser::entry\_t, 26
  - height, 26
  - name, 26
  - width, 26
  - x, 26
  - y, 26
- AutoCommand, 11
  - on\_timeout, 13
  - run, 13
  - timeout\_seconds, 13
- background\_task
  - OdometryBase, 70
- bool\_or
  - Serializer, 91
- brain
  - AutoChooser, 11
- calculate
  - FeedForward, 30
  - TrapezoidProfile, 106
- choice
  - AutoChooser, 11
- CommandController, 13
  - add, 14, 15
  - add\_delay, 15
  - last\_command\_timed\_out, 15
  - run, 15
- control\_continuous
  - Lift< T >, 47
- control\_manual
  - Lift< T >, 47
- control\_setpoints
  - Lift< T >, 48
- Core, 1
- current\_pos
  - OdometryBase, 74
- CustomEncoder, 16
  - CustomEncoder, 16
  - position, 17
  - rotation, 17
  - setPosition, 17
  - setRotation, 18
  - velocity, 18
- DelayCommand, 18
  - DelayCommand, 19
  - run, 19
- dist
  - point\_t, 87
- double\_or
  - Serializer, 91
- draw
  - GraphDrawer, 45
- drive
  - MecanumDrive, 56
- drive\_arcade
  - TankDrive, 98
- drive\_forward
  - TankDrive, 98, 99
- drive\_raw
  - MecanumDrive, 57
- drive\_tank
  - TankDrive, 99
- drive\_to\_point
  - TankDrive, 100, 101
- DriveForwardCommand, 20
  - DriveForwardCommand, 21
  - on\_timeout, 21

- run, 21
- DriveStopCommand, 22
  - DriveStopCommand, 23
  - on\_timeout, 23
  - run, 23
- DriveToPointCommand, 24
  - DriveToPointCommand, 24, 25
  - run, 25
- end\_async
  - OdometryBase, 70
- ERROR\_TYPE
  - PID, 80
- Feedback, 27
  - get, 28
  - init, 28
  - is\_on\_target, 28
  - set\_limits, 28
  - update, 29
- FeedForward, 29
  - calculate, 30
  - FeedForward, 30
- FeedForward::ff\_config\_t, 31
  - kA, 31
  - kG, 31
  - kS, 32
  - kV, 32
- Flywheel, 32
  - Flywheel, 33, 34
  - getDesiredRPM, 34
  - getFeedforwardValue, 34
  - getMotors, 34
  - getPID, 35
  - getPIDValue, 35
  - getRPM, 35
  - getTBHGain, 35
  - isTaskRunning, 35
  - measureRPM, 36
  - setPIDTarget, 36
  - spin\_manual, 36
  - spin\_raw, 37
  - spinRPM, 37
  - stop, 37
  - stopMotors, 37
  - stopNonTasks, 37
  - updatePID, 38
- FlywheelStopCommand, 38
  - FlywheelStopCommand, 39
  - run, 39
- FlywheelStopMotorsCommand, 40
  - FlywheelStopMotorsCommand, 40
  - run, 41
- FlywheelStopNonTasksCommand, 41
- GenericAuto, 42
  - add, 42
  - add\_async, 42
  - add\_delay, 43
  - run, 43
- get
  - Feedback, 28
  - MotionController, 61
  - PID, 81
  - PIDFF, 85
- get\_accel
  - OdometryBase, 71
- get\_angular\_accel\_deg
  - OdometryBase, 71
- get\_angular\_speed\_deg
  - OdometryBase, 71
- get\_async
  - Lift< T >, 48
- get\_average
  - MovingAverage, 64
- get\_choice
  - AutoChooser, 10
- get\_dir
  - Vector2D, 113
- get\_error
  - PID, 81
- get\_mag
  - Vector2D, 113
- get\_motion
  - MotionController, 61
- get\_movement\_time
  - TrapezoidProfile, 107
- get\_position
  - OdometryBase, 71
- get\_setpoint
  - Lift< T >, 48
- get\_size
  - MovingAverage, 65
- get\_speed
  - OdometryBase, 71
- get\_target
  - PID, 81
- get\_type
  - PID, 81
- get\_x
  - Vector2D, 113
- get\_y
  - Vector2D, 113
- getDesiredRPM
  - Flywheel, 34
- getFeedforwardValue
  - Flywheel, 34
- getMotors
  - Flywheel, 34
- getPID
  - Flywheel, 35
- getPIDValue
  - Flywheel, 35
- getRPM
  - Flywheel, 35
- getTBHGain
  - Flywheel, 35

- GraphDrawer, 44
  - add\_sample, 44
  - draw, 45
  - GraphDrawer, 44
- handle
  - OdometryBase, 74
- height
  - AutoChooser::entry\_t, 26
- hold
  - Lift< T >, 48
- home
  - Lift< T >, 48
- include/robot\_specs.h, 119
- include/subsystems/custom\_encoder.h, 119
- include/subsystems/flywheel.h, 120
- include/subsystems/lift.h, 121
- include/subsystems/mecanum\_drive.h, 123
- include/subsystems/odometry/odometry\_3wheel.h, 124
- include/subsystems/odometry/odometry\_base.h, 125
- include/subsystems/odometry/odometry\_tank.h, 125
- include/subsystems/screen.h, 126
- include/subsystems/tank\_drive.h, 126
- include/utis/auto\_chooser.h, 127
- include/utis/command\_structure/auto\_command.h, 128
- include/utis/command\_structure/command\_controller.h, 128
- include/utis/command\_structure/delay\_command.h, 128
- include/utis/command\_structure/drive\_commands.h, 129
- include/utis/command\_structure/flywheel\_commands.h, 130
- include/utis/feedback\_base.h, 131
- include/utis/feedforward.h, 131
- include/utis/generic\_auto.h, 132
- include/utis/geometry.h, 132
- include/utis/graph\_drawer.h, 133
- include/utis/logger.h, 133
- include/utis/math\_util.h, 134
- include/utis/motion\_controller.h, 134
- include/utis/moving\_average.h, 135
- include/utis/pid.h, 136
- include/utis/pidff.h, 136
- include/utis/pure\_pursuit.h, 137
- include/utis/serializer.h, 138
- include/utis/trapezoid\_profile.h, 138
- include/utis/vector2d.h, 139
- init
  - Feedback, 28
  - MotionController, 61
  - PID, 81
  - PIDFF, 85
- int\_or
  - Serializer, 91
- is\_on\_target
  - Feedback, 28
  - MotionController, 61
- PID, 82
- PIDFF, 85
- isTaskRunning
  - Flywheel, 35
- kA
  - FeedForward::ff\_config\_t, 31
- kG
  - FeedForward::ff\_config\_t, 31
- kS
  - FeedForward::ff\_config\_t, 32
- kV
  - FeedForward::ff\_config\_t, 32
- last\_command\_timed\_out
  - CommandController, 15
- Lift
  - Lift< T >, 47
- Lift< T >, 46
  - control\_continuous, 47
  - control\_manual, 47
  - control\_setpoints, 48
  - get\_async, 48
  - get\_setpoint, 48
  - hold, 48
  - home, 48
  - Lift, 47
  - set\_async, 49
  - set\_position, 49
  - set\_sensor\_function, 49
  - set\_sensor\_reset, 50
  - set\_setpoint, 50
- Lift< T >::lift\_cfg\_t, 50
- list
  - AutoChooser, 11
- Log
  - Logger, 52
- Logf
  - Logger, 52
- Logger, 51
  - Log, 52
  - Logf, 52
  - Logger, 51
  - LogIn, 53
- LogIn
  - Logger, 53
- measureRPM
  - Flywheel, 36
- MecanumDrive, 54
  - auto\_drive, 55
  - auto\_turn, 56
  - drive, 56
  - drive\_raw, 57
  - MecanumDrive, 55
- MecanumDrive::mecanumdrive\_config\_t, 57
- modify\_inputs
  - TankDrive, 101
- motion\_t, 58

- MotionController, 58
  - get, 61
  - get\_motion, 61
  - init, 61
  - is\_on\_target, 61
  - MotionController, 60
  - set\_limits, 62
  - tune\_feedforward, 62
  - update, 63
- MotionController::m\_profile\_cfg\_t, 53
- MovingAverage, 63
  - add\_entry, 64
  - get\_average, 64
  - get\_size, 65
  - MovingAverage, 64
- mut
  - OdometryBase, 74
- name
  - AutoChooser::entry\_t, 26
- normalize
  - Vector2D, 113
- Odometry3Wheel, 65
  - Odometry3Wheel, 67
  - tune, 67
  - update, 67
- Odometry3Wheel::odometry3wheel\_cfg\_t, 68
  - off\_axis\_center\_dist, 68
  - wheel\_diam, 68
  - wheelbase\_dist, 68
- OdometryBase, 69
  - accel, 74
  - ang\_accel\_deg, 74
  - ang\_speed\_deg, 74
  - background\_task, 70
  - current\_pos, 74
  - end\_async, 70
  - get\_accel, 71
  - get\_angular\_accel\_deg, 71
  - get\_angular\_speed\_deg, 71
  - get\_position, 71
  - get\_speed, 71
  - handle, 74
  - mut, 74
  - OdometryBase, 70
  - pos\_diff, 72
  - rot\_diff, 72
  - set\_position, 72
  - smallest\_angle, 73
  - speed, 74
  - update, 73
  - zero\_pos, 74
- OdometryTank, 75
  - OdometryTank, 76
  - set\_position, 77
  - update, 77
- OdomSetPosition, 78
  - OdomSetPosition, 78
  - run, 79
- off\_axis\_center\_dist
  - Odometry3Wheel::odometry3wheel\_cfg\_t, 68
- on\_timeout
  - AutoCommand, 13
  - DriveForwardCommand, 21
  - DriveStopCommand, 23
  - TurnDegreesCommand, 109
  - TurnToHeadingCommand, 111
- operator+
  - point\_t, 88
  - Vector2D, 114
- operator-
  - point\_t, 88
  - Vector2D, 114
- operator\*
  - Vector2D, 114
- PID, 79
  - ERROR\_TYPE, 80
  - get, 81
  - get\_error, 81
  - get\_target, 81
  - get\_type, 81
  - init, 81
  - is\_on\_target, 82
  - PID, 80
  - reset, 82
  - set\_limits, 82
  - set\_target, 82
  - update, 83
- PID::pid\_config\_t, 83
- PIDFF, 84
  - get, 85
  - init, 85
  - is\_on\_target, 85
  - set\_limits, 85
  - set\_target, 86
  - update, 86
- point
  - Vector2D, 115
- point\_t, 87
  - dist, 87
  - operator+, 88
  - operator-, 88
- pos\_diff
  - OdometryBase, 72
- pose\_t, 88
- position
  - CustomEncoder, 17
- pure\_pursuit
  - TankDrive, 102
- PurePursuit::hermite\_point, 45
- PurePursuit::spline, 96
- render
  - AutoChooser, 10
- reset
  - PID, 82

- reset\_auto
  - TankDrive, 102
- robot\_specs\_t, 89
- rot\_diff
  - OdometryBase, 72
- rotation
  - CustomEncoder, 17
- run
  - AutoCommand, 13
  - CommandController, 15
  - DelayCommand, 19
  - DriveForwardCommand, 21
  - DriveStopCommand, 23
  - DriveToPointCommand, 25
  - FlywheelStopCommand, 39
  - FlywheelStopMotorsCommand, 41
  - GenericAuto, 43
  - OdomSetPosition, 79
  - SpinRPMCommand, 96
  - TurnDegreesCommand, 109
  - TurnToHeadingCommand, 111
  - WaitUntilUpToSpeedCommand, 116
- save\_to\_disk
  - Serializer, 93
- Serializer, 90
  - bool\_or, 91
  - double\_or, 91
  - int\_or, 91
  - save\_to\_disk, 93
  - Serializer, 90
  - set\_bool, 93
  - set\_double, 93
  - set\_int, 93
  - set\_string, 94
  - string\_or, 94
- set\_accel
  - TrapezoidProfile, 107
- set\_async
  - Lift< T >, 49
- set\_bool
  - Serializer, 93
- set\_double
  - Serializer, 93
- set\_endpts
  - TrapezoidProfile, 107
- set\_int
  - Serializer, 93
- set\_limits
  - Feedback, 28
  - MotionController, 62
  - PID, 82
  - PIDFF, 85
- set\_max\_v
  - TrapezoidProfile, 107
- set\_position
  - Lift< T >, 49
  - OdometryBase, 72
  - OdometryTank, 77
- set\_sensor\_function
  - Lift< T >, 49
- set\_sensor\_reset
  - Lift< T >, 50
- set\_setpoint
  - Lift< T >, 50
- set\_string
  - Serializer, 94
- set\_target
  - PID, 82
  - PIDFF, 86
- setPIDTarget
  - Flywheel, 36
- setPosition
  - CustomEncoder, 17
- setRotation
  - CustomEncoder, 18
- smallest\_angle
  - OdometryBase, 73
- speed
  - OdometryBase, 74
- spin\_manual
  - Flywheel, 36
- spin\_raw
  - Flywheel, 37
- spinRPM
  - Flywheel, 37
- SpinRPMCommand, 95
  - run, 96
  - SpinRPMCommand, 95
- stop
  - Flywheel, 37
  - TankDrive, 102
- stopMotors
  - Flywheel, 37
- stopNonTasks
  - Flywheel, 37
- string\_or
  - Serializer, 94
- TankDrive, 97
  - drive\_arcade, 98
  - drive\_forward, 98, 99
  - drive\_tank, 99
  - drive\_to\_point, 100, 101
  - modify\_inputs, 101
  - pure\_pursuit, 102
  - reset\_auto, 102
  - stop, 102
  - TankDrive, 97
  - turn\_degrees, 103
  - turn\_to\_heading, 104
- timeout\_seconds
  - AutoCommand, 13
- TrapezoidProfile, 105
  - calculate, 106
  - get\_movement\_time, 107
  - set\_accel, 107
  - set\_endpts, 107

- set\_max\_v, [107](#)
  - TrapezoidProfile, [106](#)
- tune
  - Odometry3Wheel, [67](#)
- tune\_feedforward
  - MotionController, [62](#)
- turn\_degrees
  - TankDrive, [103](#)
- turn\_to\_heading
  - TankDrive, [104](#)
- TurnDegreesCommand, [108](#)
  - on\_timeout, [109](#)
  - run, [109](#)
  - TurnDegreesCommand, [109](#)
- TurnToHeadingCommand, [110](#)
  - on\_timeout, [111](#)
  - run, [111](#)
  - TurnToHeadingCommand, [110](#)
- update
  - Feedback, [29](#)
  - MotionController, [63](#)
  - Odometry3Wheel, [67](#)
  - OdometryBase, [73](#)
  - OdometryTank, [77](#)
  - PID, [83](#)
  - PIDFF, [86](#)
- updatePID
  - Flywheel, [38](#)
- Vector2D, [111](#)
  - get\_dir, [113](#)
  - get\_mag, [113](#)
  - get\_x, [113](#)
  - get\_y, [113](#)
  - normalize, [113](#)
  - operator+, [114](#)
  - operator-, [114](#)
  - operator\*, [114](#)
  - point, [115](#)
  - Vector2D, [112](#)
- velocity
  - CustomEncoder, [18](#)
- WaitUntilUpToSpeedCommand, [115](#)
  - run, [116](#)
  - WaitUntilUpToSpeedCommand, [116](#)
- wheel\_diam
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [68](#)
- wheelbase\_dist
  - Odometry3Wheel::odometry3wheel\_cfg\_t, [68](#)
- width
  - AutoChooser::entry\_t, [26](#)
- x
  - AutoChooser::entry\_t, [26](#)
- y
  - AutoChooser::entry\_t, [26](#)
- zero\_pos
  - OdometryBase, [74](#)