

RIT VEXU Core API

Generated by Doxygen 1.9.8

1 Core	1
1.1 Getting Started	1
1.2 Features	1
2 Hierarchical Index	3
2.1 Class Hierarchy	3
3 Class Index	5
3.1 Class List	5
4 File Index	7
4.1 File List	7
5 Class Documentation	9
5.1 Async Class Reference	9
5.1.1 Detailed Description	10
5.1.2 Member Function Documentation	10
5.1.2.1 run()	10
5.2 AutoChooser Class Reference	10
5.2.1 Detailed Description	11
5.2.2 Constructor & Destructor Documentation	11
5.2.2.1 AutoChooser()	11
5.2.3 Member Function Documentation	11
5.2.3.1 add()	11
5.2.3.2 get_choice()	11
5.2.3.3 render()	12
5.2.4 Member Data Documentation	12
5.2.4.1 brain	12
5.2.4.2 choice	12
5.2.4.3 list	12
5.3 AutoCommand Class Reference	13
5.3.1 Detailed Description	13
5.3.2 Member Function Documentation	14
5.3.2.1 on_timeout()	14
5.3.2.2 run()	14
5.3.3 Member Data Documentation	14
5.3.3.1 timeout_seconds	14
5.4 Branch Class Reference	15
5.4.1 Detailed Description	15
5.4.2 Member Function Documentation	15
5.4.2.1 on_timeout()	15
5.4.2.2 run()	16
5.5 CommandController Class Reference	16
5.5.1 Detailed Description	16

5.5.2 Constructor & Destructor Documentation	16
5.5.2.1 CommandController()	16
5.5.3 Member Function Documentation	17
5.5.3.1 add() [1/3]	17
5.5.3.2 add() [2/3]	17
5.5.3.3 add() [3/3]	17
5.5.3.4 add_cancel_func()	19
5.5.3.5 add_delay()	19
5.5.3.6 last_command_timed_out()	19
5.5.3.7 run()	20
5.6 Condition Class Reference	20
5.6.1 Detailed Description	20
5.7 CustomEncoder Class Reference	20
5.7.1 Detailed Description	21
5.7.2 Constructor & Destructor Documentation	21
5.7.2.1 CustomEncoder()	21
5.7.3 Member Function Documentation	21
5.7.3.1 position()	21
5.7.3.2 rotation()	22
5.7.3.3 setPosition()	22
5.7.3.4 setRotation()	22
5.7.3.5 velocity()	22
5.8 DelayCommand Class Reference	23
5.8.1 Detailed Description	24
5.8.2 Constructor & Destructor Documentation	24
5.8.2.1 DelayCommand()	24
5.8.3 Member Function Documentation	24
5.8.3.1 run()	24
5.9 DriveForwardCommand Class Reference	24
5.9.1 Detailed Description	25
5.9.2 Constructor & Destructor Documentation	25
5.9.2.1 DriveForwardCommand()	25
5.9.3 Member Function Documentation	26
5.9.3.1 on_timeout()	26
5.9.3.2 run()	26
5.10 DriveStopCommand Class Reference	26
5.10.1 Detailed Description	27
5.10.2 Constructor & Destructor Documentation	27
5.10.2.1 DriveStopCommand()	27
5.10.3 Member Function Documentation	27
5.10.3.1 on_timeout()	27
5.10.3.2 run()	28

5.11 DriveToPointCommand Class Reference	28
5.11.1 Detailed Description	29
5.11.2 Constructor & Destructor Documentation	29
5.11.2.1 DriveToPointCommand() [1/2]	29
5.11.2.2 DriveToPointCommand() [2/2]	29
5.11.3 Member Function Documentation	30
5.11.3.1 run()	30
5.12 AutoChooser::entry_t Struct Reference	30
5.12.1 Detailed Description	30
5.12.2 Member Data Documentation	31
5.12.2.1 height	31
5.12.2.2 name	31
5.12.2.3 width	31
5.12.2.4 x	31
5.12.2.5 y	31
5.13 Feedback Class Reference	31
5.13.1 Detailed Description	32
5.13.2 Member Function Documentation	32
5.13.2.1 get()	32
5.13.2.2 init()	32
5.13.2.3 is_on_target()	33
5.13.2.4 set_limits()	33
5.13.2.5 update()	33
5.14 FeedForward Class Reference	34
5.14.1 Detailed Description	34
5.14.2 Constructor & Destructor Documentation	34
5.14.2.1 FeedForward()	34
5.14.3 Member Function Documentation	35
5.14.3.1 calculate()	35
5.15 FeedForward::ff_config_t Struct Reference	35
5.15.1 Detailed Description	35
5.15.2 Member Data Documentation	36
5.15.2.1 kA	36
5.15.2.2 kG	36
5.15.2.3 kS	36
5.15.2.4 kV	36
5.16 Flywheel Class Reference	36
5.16.1 Detailed Description	37
5.16.2 Constructor & Destructor Documentation	37
5.16.2.1 Flywheel() [1/4]	37
5.16.2.2 Flywheel() [2/4]	38
5.16.2.3 Flywheel() [3/4]	38

5.16.2.4 Flywheel() [4 / 4]	38
5.16.3 Member Function Documentation	39
5.16.3.1 getDesiredRPM()	39
5.16.3.2 getFeedforwardValue()	39
5.16.3.3 getMotors()	39
5.16.3.4 getPID()	39
5.16.3.5 getPIDValue()	40
5.16.3.6 getRPM()	40
5.16.3.7 getTBHGain()	40
5.16.3.8 isTaskRunning()	40
5.16.3.9 measureRPM()	40
5.16.3.10 setPIDTarget()	40
5.16.3.11 spin_manual()	41
5.16.3.12 spin_raw()	41
5.16.3.13 spinRPM()	41
5.16.3.14 stop()	42
5.16.3.15 stopMotors()	42
5.16.3.16 stopNonTasks()	42
5.16.3.17 updatePID()	42
5.17 FlywheelStopCommand Class Reference	42
5.17.1 Detailed Description	43
5.17.2 Constructor & Destructor Documentation	43
5.17.2.1 FlywheelStopCommand()	43
5.17.3 Member Function Documentation	43
5.17.3.1 run()	43
5.18 FlywheelStopMotorsCommand Class Reference	44
5.18.1 Detailed Description	44
5.18.2 Constructor & Destructor Documentation	44
5.18.2.1 FlywheelStopMotorsCommand()	44
5.18.3 Member Function Documentation	45
5.18.3.1 run()	45
5.19 FlywheelStopNonTasksCommand Class Reference	45
5.19.1 Detailed Description	46
5.20 FunctionCommand Class Reference	46
5.20.1 Detailed Description	47
5.20.2 Member Function Documentation	47
5.20.2.1 run()	47
5.21 FunctionCondition Class Reference	47
5.21.1 Detailed Description	47
5.21.2 Member Function Documentation	48
5.21.2.1 test()	48
5.22 screen::FunctionPage Class Reference	48

5.22.1 Detailed Description	48
5.22.2 Constructor & Destructor Documentation	48
5.22.2.1 FunctionPage()	48
5.22.3 Member Function Documentation	49
5.22.3.1 draw()	49
5.22.3.2 update()	49
5.23 GenericAuto Class Reference	49
5.23.1 Detailed Description	50
5.23.2 Member Function Documentation	50
5.23.2.1 add()	50
5.23.2.2 add_async()	50
5.23.2.3 add_delay()	50
5.23.2.4 run()	50
5.24 GraphDrawer Class Reference	51
5.24.1 Constructor & Destructor Documentation	51
5.24.1.1 GraphDrawer()	51
5.24.2 Member Function Documentation	52
5.24.2.1 add_sample()	52
5.24.2.2 draw()	52
5.25 PurePursuit::hermite_point Struct Reference	53
5.25.1 Detailed Description	53
5.26 IfTimePassed Class Reference	53
5.26.1 Detailed Description	54
5.26.2 Member Function Documentation	54
5.26.2.1 test()	54
5.27 InOrder Class Reference	54
5.27.1 Detailed Description	55
5.27.2 Member Function Documentation	55
5.27.2.1 on_timeout()	55
5.27.2.2 run()	55
5.28 Lift< T > Class Template Reference	55
5.28.1 Detailed Description	56
5.28.2 Constructor & Destructor Documentation	56
5.28.2.1 Lift()	56
5.28.3 Member Function Documentation	57
5.28.3.1 control_continuous()	57
5.28.3.2 control_manual()	57
5.28.3.3 control_setpoints()	57
5.28.3.4 get_async()	58
5.28.3.5 get_setpoint()	58
5.28.3.6 hold()	58
5.28.3.7 home()	58

5.28.3.8 set_async()	58
5.28.3.9 set_position()	59
5.28.3.10 set_sensor_function()	59
5.28.3.11 set_sensor_reset()	59
5.28.3.12 set_setpoint()	59
5.29 Lift< T >::lift_cfg_t Struct Reference	60
5.29.1 Detailed Description	60
5.30 Logger Class Reference	60
5.30.1 Detailed Description	61
5.30.2 Constructor & Destructor Documentation	61
5.30.2.1 Logger()	61
5.30.3 Member Function Documentation	62
5.30.3.1 Log() [1/2]	62
5.30.3.2 Log() [2/2]	62
5.30.3.3 Logf() [1/2]	62
5.30.3.4 Logf() [2/2]	62
5.30.3.5 Logln() [1/2]	63
5.30.3.6 Logln() [2/2]	63
5.31 MotionController::m_profile_cfg_t Struct Reference	63
5.31.1 Detailed Description	64
5.32 Mat2 Struct Reference	64
5.33 MecanumDrive Class Reference	64
5.33.1 Detailed Description	65
5.33.2 Constructor & Destructor Documentation	65
5.33.2.1 MecanumDrive()	65
5.33.3 Member Function Documentation	65
5.33.3.1 auto_drive()	65
5.33.3.2 auto_turn()	66
5.33.3.3 drive()	66
5.33.3.4 drive_raw()	67
5.34 MecanumDrive::mecanumdrive_config_t Struct Reference	68
5.34.1 Detailed Description	68
5.35 motion_t Struct Reference	68
5.35.1 Detailed Description	68
5.36 MotionController Class Reference	69
5.36.1 Detailed Description	70
5.36.2 Constructor & Destructor Documentation	70
5.36.2.1 MotionController()	70
5.36.3 Member Function Documentation	70
5.36.3.1 get()	70
5.36.3.2 get_motion()	71
5.36.3.3 init()	71

5.36.3.4 <code>is_on_target()</code>	71
5.36.3.5 <code>set_limits()</code>	71
5.36.3.6 <code>tune_feedforward()</code>	72
5.36.3.7 <code>update()</code>	72
5.37 MovingAverage Class Reference	73
5.37.1 Detailed Description	73
5.37.2 Constructor & Destructor Documentation	73
5.37.2.1 <code>MovingAverage()</code> [1/2]	73
5.37.2.2 <code>MovingAverage()</code> [2/2]	74
5.37.3 Member Function Documentation	74
5.37.3.1 <code>add_entry()</code>	74
5.37.3.2 <code>get_average()</code>	74
5.37.3.3 <code>get_size()</code>	75
5.38 Odometry3Wheel Class Reference	75
5.38.1 Detailed Description	76
5.38.2 Constructor & Destructor Documentation	77
5.38.2.1 <code>Odometry3Wheel()</code>	77
5.38.3 Member Function Documentation	77
5.38.3.1 <code>tune()</code>	77
5.38.3.2 <code>update()</code>	77
5.39 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference	78
5.39.1 Detailed Description	78
5.39.2 Member Data Documentation	78
5.39.2.1 <code>off_axis_center_dist</code>	78
5.39.2.2 <code>wheel_diam</code>	78
5.39.2.3 <code>wheelbase_dist</code>	78
5.40 OdometryBase Class Reference	79
5.40.1 Detailed Description	80
5.40.2 Constructor & Destructor Documentation	80
5.40.2.1 <code>OdometryBase()</code>	80
5.40.3 Member Function Documentation	80
5.40.3.1 <code>background_task()</code>	80
5.40.3.2 <code>end_async()</code>	81
5.40.3.3 <code>get_accel()</code>	81
5.40.3.4 <code>get_angular_accel_deg()</code>	81
5.40.3.5 <code>get_angular_speed_deg()</code>	81
5.40.3.6 <code>get_position()</code>	81
5.40.3.7 <code>get_speed()</code>	82
5.40.3.8 <code>pos_diff()</code>	82
5.40.3.9 <code>rot_diff()</code>	82
5.40.3.10 <code>set_position()</code>	83
5.40.3.11 <code>smallest_angle()</code>	83

5.40.3.12 update()	83
5.40.4 Member Data Documentation	84
5.40.4.1 accel	84
5.40.4.2 ang_accel_deg	84
5.40.4.3 ang_speed_deg	84
5.40.4.4 current_pos	84
5.40.4.5 handle	84
5.40.4.6 mut	84
5.40.4.7 speed	84
5.40.4.8 zero_pos	85
5.41 screen::OdometryPage Class Reference	85
5.41.1 Detailed Description	85
5.41.2 Member Function Documentation	85
5.41.2.1 draw()	85
5.41.2.2 update()	86
5.42 OdometryTank Class Reference	86
5.42.1 Detailed Description	87
5.42.2 Constructor & Destructor Documentation	87
5.42.2.1 OdometryTank() [1/3]	87
5.42.2.2 OdometryTank() [2/3]	88
5.42.2.3 OdometryTank() [3/3]	88
5.42.3 Member Function Documentation	89
5.42.3.1 set_position()	89
5.42.3.2 update()	89
5.43 OdomSetPosition Class Reference	89
5.43.1 Detailed Description	90
5.43.2 Constructor & Destructor Documentation	90
5.43.2.1 OdomSetPosition()	90
5.43.3 Member Function Documentation	91
5.43.3.1 run()	91
5.44 screen::Page Class Reference	91
5.44.1 Detailed Description	91
5.44.2 Member Function Documentation	91
5.44.2.1 draw()	91
5.44.2.2 update()	92
5.45 Parallel Class Reference	92
5.45.1 Detailed Description	93
5.45.2 Member Function Documentation	93
5.45.2.1 on_timeout()	93
5.45.2.2 run()	93
5.46 parallel_runner_info Struct Reference	94
5.47 PID Class Reference	94

5.47.1 Detailed Description	95
5.47.2 Member Enumeration Documentation	95
5.47.2.1 ERROR_TYPE	95
5.47.3 Constructor & Destructor Documentation	95
5.47.3.1 PID()	95
5.47.4 Member Function Documentation	96
5.47.4.1 get()	96
5.47.4.2 get_error()	96
5.47.4.3 get_target()	96
5.47.4.4 get_type()	96
5.47.4.5 init()	96
5.47.4.6 is_on_target()	97
5.47.4.7 reset()	97
5.47.4.8 set_limits()	97
5.47.4.9 set_target()	98
5.47.4.10 update()	98
5.48 PID::pid_config_t Struct Reference	98
5.48.1 Detailed Description	99
5.49 PIDFF Class Reference	99
5.49.1 Member Function Documentation	100
5.49.1.1 get()	100
5.49.1.2 init()	100
5.49.1.3 is_on_target()	100
5.49.1.4 set_limits()	101
5.49.1.5 set_target()	101
5.49.1.6 update() [1/2]	101
5.49.1.7 update() [2/2]	101
5.50 point_t Struct Reference	102
5.50.1 Detailed Description	102
5.50.2 Member Function Documentation	102
5.50.2.1 dist()	102
5.50.2.2 operator+()	103
5.50.2.3 operator-()	103
5.51 pose_t Struct Reference	103
5.51.1 Detailed Description	104
5.52 PurePursuitCommand Class Reference	104
5.52.1 Detailed Description	105
5.52.2 Constructor & Destructor Documentation	105
5.52.2.1 PurePursuitCommand()	105
5.52.3 Member Function Documentation	105
5.52.3.1 on_timeout()	105
5.52.3.2 run()	106

5.53 robot_specs_t Struct Reference	106
5.53.1 Detailed Description	106
5.54 screen::ScreenData Struct Reference	107
5.54.1 Detailed Description	107
5.55 Serializer Class Reference	107
5.55.1 Detailed Description	108
5.55.2 Constructor & Destructor Documentation	108
5.55.2.1 Serializer()	108
5.55.3 Member Function Documentation	108
5.55.3.1 bool_or()	108
5.55.3.2 double_or()	108
5.55.3.3 int_or()	109
5.55.3.4 save_to_disk()	109
5.55.3.5 set_bool()	109
5.55.3.6 set_double()	110
5.55.3.7 set_int()	110
5.55.3.8 set_string()	110
5.55.3.9 string_or()	110
5.56 SpinRPMCommand Class Reference	111
5.56.1 Detailed Description	112
5.56.2 Constructor & Destructor Documentation	112
5.56.2.1 SpinRPMCommand()	112
5.56.3 Member Function Documentation	112
5.56.3.1 run()	112
5.57 PurePursuit::spline Struct Reference	112
5.57.1 Detailed Description	113
5.58 screen::StatsPage Class Reference	113
5.58.1 Member Function Documentation	113
5.58.1.1 draw()	113
5.58.1.2 update()	114
5.59 TankDrive Class Reference	114
5.59.1 Detailed Description	115
5.59.2 Constructor & Destructor Documentation	115
5.59.2.1 TankDrive()	115
5.59.3 Member Function Documentation	116
5.59.3.1 drive_arcade()	116
5.59.3.2 drive_forward() [1/2]	116
5.59.3.3 drive_forward() [2/2]	117
5.59.3.4 drive_tank()	117
5.59.3.5 drive_to_point() [1/2]	118
5.59.3.6 drive_to_point() [2/2]	118
5.59.3.7 modify_inputs()	119

5.59.3.8 pure_pursuit() [1/2]	120
5.59.3.9 pure_pursuit() [2/2]	120
5.59.3.10 reset_auto()	121
5.59.3.11 stop()	121
5.59.3.12 turn_degrees() [1/2]	121
5.59.3.13 turn_degrees() [2/2]	121
5.59.3.14 turn_to_heading() [1/2]	122
5.59.3.15 turn_to_heading() [2/2]	123
5.60 TrapezoidProfile Class Reference	123
5.60.1 Detailed Description	124
5.60.2 Constructor & Destructor Documentation	124
5.60.2.1 TrapezoidProfile()	124
5.60.3 Member Function Documentation	124
5.60.3.1 calculate()	124
5.60.3.2 get_movement_time()	125
5.60.3.3 set_accel()	125
5.60.3.4 set_endpts()	125
5.60.3.5 set_max_v()	126
5.61 TurnDegreesCommand Class Reference	126
5.61.1 Detailed Description	127
5.61.2 Constructor & Destructor Documentation	127
5.61.2.1 TurnDegreesCommand()	127
5.61.3 Member Function Documentation	127
5.61.3.1 on_timeout()	127
5.61.3.2 run()	127
5.62 TurnToHeadingCommand Class Reference	128
5.62.1 Detailed Description	128
5.62.2 Constructor & Destructor Documentation	128
5.62.2.1 TurnToHeadingCommand()	128
5.62.3 Member Function Documentation	129
5.62.3.1 on_timeout()	129
5.62.3.2 run()	129
5.63 Vector2D Class Reference	129
5.63.1 Detailed Description	130
5.63.2 Constructor & Destructor Documentation	130
5.63.2.1 Vector2D() [1/2]	130
5.63.2.2 Vector2D() [2/2]	130
5.63.3 Member Function Documentation	131
5.63.3.1 get_dir()	131
5.63.3.2 get_mag()	131
5.63.3.3 get_x()	131
5.63.3.4 get_y()	131

5.63.3.5 normalize()	132
5.63.3.6 operator*()	132
5.63.3.7 operator+()	132
5.63.3.8 operator-()	132
5.63.3.9 point()	133
5.64 WaitUntilCondition Class Reference	133
5.64.1 Detailed Description	134
5.64.2 Member Function Documentation	134
5.64.2.1 run()	134
5.65 WaitUntilUpToSpeedCommand Class Reference	134
5.65.1 Detailed Description	135
5.65.2 Constructor & Destructor Documentation	135
5.65.2.1 WaitUntilUpToSpeedCommand()	135
5.65.3 Member Function Documentation	135
5.65.3.1 run()	135
6 File Documentation	137
6.1 robot_specs.h	137
6.2 custom_encoder.h	137
6.3 flywheel.h	138
6.4 lift.h	139
6.5 mecanum_drive.h	142
6.6 odometry_3wheel.h	142
6.7 odometry_base.h	143
6.8 odometry_tank.h	144
6.9 screen.h	144
6.10 tank_drive.h	145
6.11 auto_chooser.h	146
6.12 auto_command.h	147
6.13 command_controller.h	148
6.14 delay_command.h	149
6.15 drive_commands.h	149
6.16 flywheel_commands.h	151
6.17 feedback_base.h	152
6.18 feedforward.h	152
6.19 generic_auto.h	153
6.20 geometry.h	153
6.21 graph_drawer.h	154
6.22 logger.h	155
6.23 math_util.h	155
6.24 motion_controller.h	156
6.25 moving_average.h	157

6.26 pid.h	157
6.27 pidff.h	158
6.28 pure_pursuit.h	159
6.29 serializer.h	159
6.30 trapezoid_profile.h	160
6.31 vector2d.h	160
Index	163

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	10
AutoCommand	13
Async	9
Branch	15
DelayCommand	23
DriveForwardCommand	24
DriveStopCommand	26
DriveToPointCommand	28
FlywheelStopCommand	42
FlywheelStopMotorsCommand	44
FlywheelStopNonTasksCommand	45
FunctionCommand	46
InOrder	54
OdomSetPosition	89
Parallel	92
PurePursuitCommand	104
SpinRPMCommand	111
TurnDegreesCommand	126
TurnToHeadingCommand	128
WaitUntilCondition	133
WaitUntilUpToSpeedCommand	134
CommandController	16
Condition	20
FunctionCondition	47
IfTimePassed	53
vex::encoder	
CustomEncoder	20
AutoChooser::entry_t	30
Feedback	31
MotionController	69
PID	94
PIDFF	99
FeedForward	34
FeedForward::ff_config_t	35

Flywheel	36
GenericAuto	49
GraphDrawer	51
PurePursuit::hermite_point	53
Lift< T >	55
Lift< T >::lift_cfg_t	60
Logger	60
MotionController::m_profile_cfg_t	63
Mat2	64
MecanumDrive	64
MecanumDrive::mecanumdrive_config_t	68
motion_t	68
MovingAverage	73
Odometry3Wheel::odometry3wheel_cfg_t	78
OdometryBase	79
Odometry3Wheel	75
OdometryTank	86
screen::Page	91
screen::FunctionPage	48
screen::OdometryPage	85
screen::StatsPage	113
parallel_runner_info	94
PID::pid_config_t	98
point_t	102
pose_t	103
robot_specs_t	106
screen::ScreenData	107
Serializer	107
PurePursuit::spline	112
TankDrive	114
TrapezoidProfile	123
Vector2D	129

Chapter 3

Class Index

3.1 Class List

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

Async	
Async runs a command asynchronously will simply let it go and never look back THIS HAS A VERY NICHE USE CASE. THINK ABOUT IF YOU REALLY NEED IT	9
AutoChooser	10
AutoCommand	13
Branch	
Branch chooses from multiple options at runtime. the function decider returns an index into the choices vector If you wish to make no choice and skip this section, return NO_CHOICE; any choice that is out of bounds set to NO_CHOICE	15
CommandController	16
Condition	20
CustomEncoder	20
DelayCommand	23
DriveForwardCommand	24
DriveStopCommand	26
DriveToPointCommand	28
AutoChooser::entry_t	30
Feedback	31
FeedForward	34
FeedForward::ff_config_t	35
Flywheel	36
FlywheelStopCommand	42
FlywheelStopMotorsCommand	44
FlywheelStopNonTasksCommand	45
FunctionCommand	46
FunctionCondition	
FunctionCondition is a quick and dirty Condition to wrap some expression that should be evaluated at runtime	47
screen::FunctionPage	
screen::FunctionPage Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything	48
GenericAuto	49
GraphDrawer	51
PurePursuit::hermite_point	53

IfTimePassed	
IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time_s	53
InOrder	
InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timeouts?	54
Lift< T >	55
Lift< T >::lift_cfg_t	60
Logger	
Class to simplify writing to files	60
MotionController::m_profile_cfg_t	63
Mat2	64
MecanumDrive	64
MecanumDrive::mecanumdrive_config_t	68
motion_t	68
MotionController	69
MovingAverage	73
Odometry3Wheel	75
Odometry3Wheel::odometry3wheel_cfg_t	78
OdometryBase	79
screen::OdometryPage	
Page that shows odometry position and rotation and a map (if an sd card with the file is on) . .	85
OdometryTank	86
OdomSetPosition	89
screen::Page	
Page describes one part of the screen slideshow	91
Parallel	
Parallel runs multiple commands in parallel and waits for all to finish before continuing. if none finish before this command's timeout, it will call on_timeout on all children continue	92
parallel_runner_info	94
PID	94
PID::pid_config_t	98
PIDFF	99
point_t	102
pose_t	103
PurePursuitCommand	104
robot_specs_t	106
screen::ScreenData	
Holds the data that will be passed to the screen thread you probably shouldnt have to use it . .	107
Serializer	
Serializes Arbitrary data to a file on the SD Card	107
SpinRPMCommand	111
PurePursuit::spline	112
screen::StatsPage	113
TankDrive	114
TrapezoidProfile	123
TurnDegreesCommand	126
TurnToHeadingCommand	128
Vector2D	129
WaitUntilCondition	
Waits until the condition is true	133
WaitUntilUpToSpeedCommand	134

Chapter 4

File Index

4.1 File List

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

include/robot_specs.h	137
include/subsystems/custom_encoder.h	137
include/subsystems/flywheel.h	138
include/subsystems/lift.h	139
include/subsystems/mecanum_drive.h	142
include/subsystems/screen.h	144
include/subsystems/tank_drive.h	145
include/subsystems/odometry/odometry_3wheel.h	142
include/subsystems/odometry/odometry_base.h	143
include/subsystems/odometry/odometry_tank.h	144
include/utls/auto_chooser.h	146
include/utls/feedback_base.h	152
include/utls/feedforward.h	152
include/utls/generic_auto.h	153
include/utls/geometry.h	153
include/utls/graph_drawer.h	154
include/utls/logger.h	155
include/utls/math_util.h	155
include/utls/motion_controller.h	156
include/utls/moving_average.h	157
include/utls/pid.h	157
include/utls/pidff.h	158
include/utls/pure_pursuit.h	159
include/utls/serializer.h	159
include/utls/trapezoid_profile.h	160
include/utls/vector2d.h	160
include/utls/command_structure/auto_command.h	147
include/utls/command_structure/command_controller.h	148
include/utls/command_structure/delay_command.h	149
include/utls/command_structure/drive_commands.h	149
include/utls/command_structure/flywheel_commands.h	151

Chapter 5

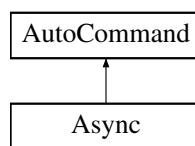
Class Documentation

5.1 Async Class Reference

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

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

Inheritance diagram for Async:



Public Member Functions

- **Async** ([AutoCommand](#) *cmd)
- 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

5.1.1 Detailed Description

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

5.1.2 Member Function Documentation

5.1.2.1 run()

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

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.2 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](#)

5.2.1 Detailed Description

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

5.2.2 Constructor & Destructor Documentation

5.2.2.1 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
--------------	--

5.2.3 Member Function Documentation

5.2.3.1 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.

5.2.3.2 get_choice()

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

Get the currently selected auto choice

Returns

the identifier to the auto path

Return the selected autonomous

5.2.3.3 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
-----------------	---------------------------------------

5.2.4 Member Data Documentation

5.2.4.1 brain

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

the brain to show the choices on

5.2.4.2 choice

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

the current choice of auto

5.2.4.3 list

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

< a list of all possible auto choices

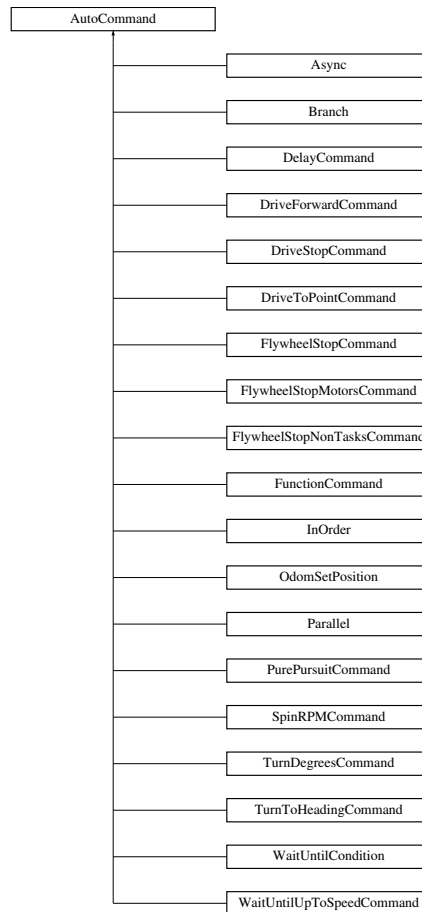
The documentation for this class was generated from the following files:

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

5.3 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

5.3.1 Detailed Description

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

5.3.2 Member Function Documentation

5.3.2.1 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 [InOrder](#), [Parallel](#), [Branch](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [TurnToHeadingCommand](#), [PurePursuitCommand](#), and [DriveStopCommand](#).

5.3.2.2 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 [FunctionCommand](#), [WaitUntilCondition](#), [InOrder](#), [Parallel](#), [Branch](#), [Async](#), [DelayCommand](#), [DriveForwardCommand](#), [TurnDegreesCommand](#), [DriveToPointCommand](#), [TurnToHeadingCommand](#), [PurePursuitCommand](#), [DriveStopCommand](#), [OdomSetPosition](#), [SpinRPMCommand](#), [WaitUntilUpToSpeedCommand](#), [FlywheelStopCommand](#), and [FlywheelStopMotorsCommand](#).

5.3.3 Member Data Documentation

5.3.3.1 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 ≤ 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:

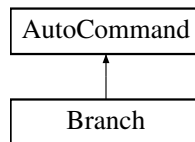
- include/utils/command_structure/auto_command.h

5.4 Branch Class Reference

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

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

Inheritance diagram for Branch:



Public Member Functions

- **Branch** ([Condition](#) *cond, [AutoCommand](#) *false_choice, [AutoCommand](#) *true_choice)
- 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

5.4.1 Detailed Description

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

5.4.2 Member Function Documentation

5.4.2.1 [on_timeout\(\)](#)

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

5.4.2.2 run()

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

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utls/command_structure/auto_command.h
- src/utls/command_structure/auto_command.cpp

5.5 CommandController Class Reference

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

Public Member Functions

- **CommandController ()**
Create an empty [CommandController](#). Add Command with [CommandController::add\(\)](#)
- **CommandController (std::initializer_list< [AutoCommand](#) * > cmds)**
Create a [CommandController](#) with commands pre added. More can be added with [CommandController::add\(\)](#)
- void **add** (std::vector< [AutoCommand](#) * > cmds)
- void **add** ([AutoCommand](#) *cmd, double timeout_seconds=10.0)
- void **add** (std::vector< [AutoCommand](#) * > cmds, double timeout_sec)
- void **add_delay** (int ms)
- void **add_cancel_func** (std::function< bool(void)> true_if_cancel)
add_cancel_func specifies that when this func evaluates to true, to cancel the command controller
- void **run** ()
- bool **last_command_timed_out** ()

5.5.1 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.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 CommandController()

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

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

Parameters

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

5.5.3 Member Function Documentation

5.5.3.1 add() [1/3]

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

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

Parameters

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

5.5.3.2 add() [2/3]

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

Adds a command to the queue

Parameters

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

Add multiple commands to the queue. No timeout here.

Parameters

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

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

Add multiple commands to the queue. No timeout here.

Parameters

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

5.5.3.4 add_cancel_func()

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

add_cancel_func specifies that when this func evaluates to true, to cancel the command controller

Parameters

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

5.5.3.5 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
-----------	--

5.5.3.6 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

5.5.3.7 run()

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

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

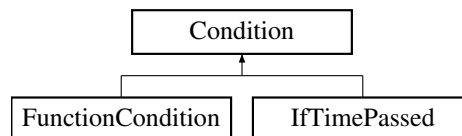
The documentation for this class was generated from the following files:

- include/utls/command_structure/command_controller.h
- src/utls/command_structure/command_controller.cpp

5.6 Condition Class Reference

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

Inheritance diagram for Condition:



Public Member Functions

- virtual bool **test** ()=0

5.6.1 Detailed Description

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

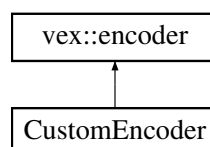
The documentation for this class was generated from the following file:

- include/utls/command_structure/auto_command.h

5.7 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)

5.7.1 Detailed Description

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

5.7.2 Constructor & Destructor Documentation

5.7.2.1 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

5.7.3 Member Function Documentation

5.7.3.1 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

5.7.3.2 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

5.7.3.3 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

5.7.3.4 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

5.7.3.5 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

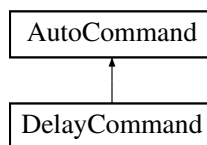
The documentation for this class was generated from the following files:

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

5.8 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

5.8.1 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

5.8.2 Constructor & Destructor Documentation

5.8.2.1 DelayCommand()

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

Construct a delay command

Parameters

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

5.8.3 Member Function Documentation

5.8.3.1 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](#).

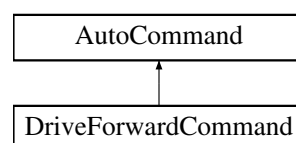
The documentation for this class was generated from the following file:

- include/utls/command_structure/delay_command.h

5.9 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

5.9.1 Detailed Description

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

5.9.2 Constructor & Destructor Documentation

5.9.2.1 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

5.9.3 Member Function Documentation**5.9.3.1 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](#).

5.9.3.2 run()

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

Run drive_forward Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

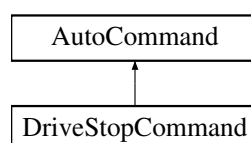
The documentation for this class was generated from the following files:

- include/utlis/command_structure/drive_commands.h
- src/utlis/command_structure/drive_commands.cpp

5.10 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

5.10.1 Detailed Description

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

5.10.2 Constructor & Destructor Documentation

5.10.2.1 DriveStopCommand()

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

Construct a DriveStop Command

Parameters

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

5.10.3 Member Function Documentation

5.10.3.1 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](#).

5.10.3.2 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](#).

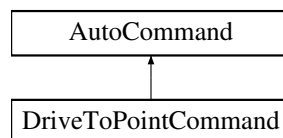
The documentation for this class was generated from the following files:

- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.11 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

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 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

5.11.2.2 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

5.11.3 Member Function Documentation**5.11.3.1 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/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.12 AutoChooser::entry_t Struct Reference

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

Public Attributes

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

5.12.1 Detailed Description

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

5.12.2 Member Data Documentation

5.12.2.1 height

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

height of the block

5.12.2.2 name

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

name of the auto represented by the block

5.12.2.3 width

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

width of the block

5.12.2.4 x

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

screen x position of the block

5.12.2.5 y

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

screen y position of the block

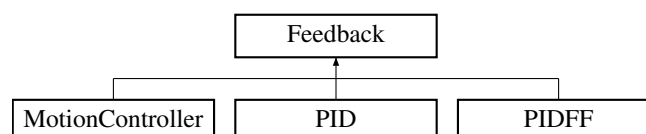
The documentation for this struct was generated from the following file:

- include/utils/auto_chooser.h

5.13 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** ()

5.13.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

Author

Ryan McGee

Date

9/25/2022

5.13.2 Member Function Documentation

5.13.2.1 **get()**

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

Returns

the last saved result from the feedback controller

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

5.13.2.2 **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](#).

5.13.2.3 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](#).

5.13.2.4 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](#).

5.13.2.5 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/utls/feedback_base.h`

5.14 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.

5.14.1 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

5.14.2 Constructor & Destructor Documentation

5.14.2.1 [FeedForward\(\)](#)

```
FeedForward::FeedForward (
    ff\_config\_t & cfg ) [inline]
```

Creates a [FeedForward](#) object.

Parameters

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

5.14.3 Member Function Documentation

5.14.3.1 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/utlis/feedforward.h

5.15 FeedForward::ff_config_t Struct Reference

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

Public Attributes

- double *kS*
- double *kV*
- double *kA*
- double *kG*

5.15.1 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

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

5.15.2 Member Data Documentation

5.15.2.1 kA

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

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

5.15.2.2 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.

5.15.2.3 kS

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

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

5.15.2.4 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/utls/feedforward.h

5.16 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](#) ()
- [AutoCommand](#) * [SpinRpmCmd](#) (int rpm)
- [AutoCommand](#) * [WaitUntilUpToSpeedCmd](#) ()

5.16.1 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.

5.16.2 Constructor & Destructor Documentation

5.16.2.1 [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.

5.16.2.2 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

5.16.2.3 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

5.16.2.4 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

5.16.3 Member Function Documentation

5.16.3.1 `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

5.16.3.2 `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

5.16.3.3 `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

5.16.3.4 `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](#)

5.16.3.5 `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

5.16.3.6 `getRPM()`

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

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

5.16.3.7 `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

5.16.3.8 `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

5.16.3.9 `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

5.16.3.10 `setPIDTarget()`

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

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

Parameters

<i>value</i>	- desired value of the PID
--------------	--

5.16.3.11 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

5.16.3.12 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

5.16.3.13 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
-----------------	-----------------------

5.16.3.14 stop()

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

stop the RPM thread and the wheel

5.16.3.15 stopMotors()

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

stop only the motors; exclusively for BANG BANG use

5.16.3.16 stopNonTasks()

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

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

5.16.3.17 updatePID()

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

updates the value of the [PID](#)

Parameters

<i>value</i>	- value to update the PID with
--------------	--

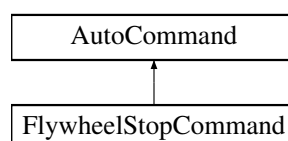
The documentation for this class was generated from the following files:

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

5.17 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

5.17.1 Detailed Description

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

5.17.2 Constructor & Destructor Documentation

5.17.2.1 FlywheelStopCommand()

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

Construct a [FlywheelStopCommand](#)

Parameters

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

5.17.3 Member Function Documentation

5.17.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

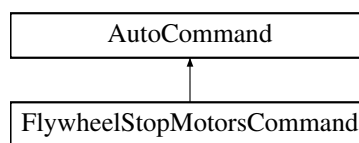
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.18 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

5.18.1 Detailed Description

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

5.18.2 Constructor & Destructor Documentation

5.18.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheelStopMotors Command

Parameters

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

5.18.3 Member Function Documentation

5.18.3.1 run()

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

Run stop Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

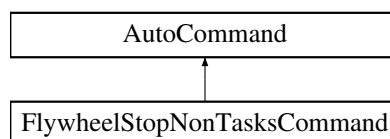
The documentation for this class was generated from the following files:

- include/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.19 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

5.19.1 Detailed Description

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

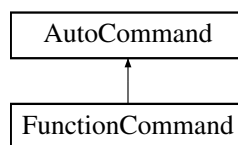
The documentation for this class was generated from the following files:

- include/utlis/command_structure/flywheel_commands.h
- src/utlis/command_structure/flywheel_commands.cpp

5.20 FunctionCommand Class Reference

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

Inheritance diagram for FunctionCommand:



Public Member Functions

- **FunctionCommand** (std::function< bool(void)> f)
- bool [run](#) ()

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

5.20.1 Detailed Description

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

5.20.2 Member Function Documentation

5.20.2.1 run()

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

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

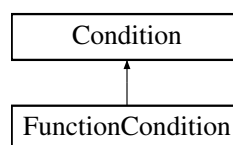
- include/utils/command_structure/auto_command.h

5.21 FunctionCondition Class Reference

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

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

Inheritance diagram for FunctionCondition:



Public Member Functions

- **FunctionCondition** (std::function< bool()> cond, std::function< void(void)> timeout=[]() {})
- bool [test](#) () override

5.21.1 Detailed Description

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

5.21.2 Member Function Documentation

5.21.2.1 test()

```
bool FunctionCondition::test ( ) [override], [virtual]
```

Implements [Condition](#).

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

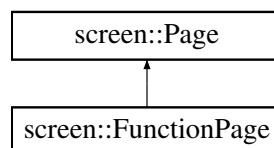
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.22 screen::FunctionPage Class Reference

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

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

Inheritance diagram for screen::FunctionPage:



Public Member Functions

- [FunctionPage](#) (update_func_t update_f, draw_func_t draw_t)
FunctionPage.
- void [update](#) (bool was_pressed, int x, int y) override
update uses the supplied update function to update this page
- void [draw](#) (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
draw uses the supplied draw function to draw to the screen

5.22.1 Detailed Description

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

5.22.2 Constructor & Destructor Documentation

5.22.2.1 FunctionPage()

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

[FunctionPage](#).

Parameters

<i>update↔ _f</i>	drawing function
<i>draw_f</i>	drawing function

5.22.3 Member Function Documentation

5.22.3.1 draw()

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

draw uses the supplied draw function to draw to the screen

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

5.22.3.2 update()

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

update uses the supplied update function to update this page

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

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

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

5.23 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)

5.23.1 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

5.23.2 Member Function Documentation

5.23.2.1 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
------------------	---------------------

5.23.2.2 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
--------------------	---------------------

5.23.2.3 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
-----------	----------------------------------

5.23.2.4 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/utlis/generic_auto.h
- src/utlis/generic_auto.cpp

5.24 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)

5.24.1 Constructor & Destructor Documentation

5.24.1.1 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
---------------	---

Parameters

<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

5.24.2 Member Function Documentation

5.24.2.1 `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
---------------	---

5.24.2.2 `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/utlis/graph_drawer.h`
- `src/utlis/graph_drawer.cpp`

5.25 PurePursuit::hermite_point Struct Reference

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

Public Member Functions

- [point_t](#) **getPoint** () const
- [Vector2D](#) **getTangent** () const

Public Attributes

- double **x**
- double **y**
- double **dir**
- double **mag**

5.25.1 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:

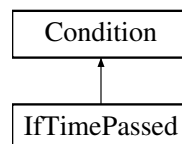
- include/utils/pure_pursuit.h

5.26 IfTimePassed Class Reference

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

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

Inheritance diagram for IfTimePassed:



Public Member Functions

- **IfTimePassed** (double time_s)
- bool [test](#) () override

5.26.1 Detailed Description

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

5.26.2 Member Function Documentation

5.26.2.1 test()

```
bool IfTimePassed::test ( ) [override], [virtual]
```

Implements [Condition](#).

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

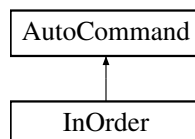
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.27 InOrder Class Reference

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

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

Inheritance diagram for InOrder:



Public Member Functions

- **InOrder** (std::queue< [AutoCommand](#) * > cmds)
- **InOrder** (std::initializer_list< [AutoCommand](#) * > cmds)
- 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

5.27.1 Detailed Description

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

5.27.2 Member Function Documentation

5.27.2.1 on_timeout()

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

5.27.2.2 run()

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

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.28 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))

5.28.1 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

5.28.2 Constructor & Destructor Documentation

5.28.2.1 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>	Lift characterization information; PID tunings and movement speeds
<i>setpoint_map</i>	A map of enum type T, in which each enum entry corresponds to a different lift height

5.28.3 Member Function Documentation

5.28.3.1 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

5.28.3.2 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

5.28.3.3 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).

5.28.3.4 get_async()

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

Returns

whether or not the background thread is running the lift

5.28.3.5 get_setpoint()

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

Returns

The current setpoint for the lift

5.28.3.6 hold()

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

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

5.28.3.7 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.

5.28.3.8 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
------------	--

5.28.3.9 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

5.28.3.10 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
---------------	-----------------------------------

5.28.3.11 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`

5.28.3.12 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>	Lift setpoint, in motor revolutions or sensor units defined by <code>get_sensor</code> . 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

5.29 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**

5.29.1 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

5.30 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

5.30.1 Detailed Description

Class to simplify writing to files.

5.30.2 Constructor & Destructor Documentation

5.30.2.1 `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
-----------------	---------------------

5.30.3 Member Function Documentation

5.30.3.1 Log() [1/2]

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

Write a string to the log.

Parameters

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

5.30.3.2 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

5.30.3.3 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)
<i>...</i>	the args

5.30.3.4 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

5.30.3.5 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
----------	---------------------

5.30.3.6 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

5.31 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

5.31.1 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

5.32 Mat2 Struct Reference

Public Member Functions

- [point_t](#) operator* (const [point_t](#) p) const

Static Public Member Functions

- static [Mat2](#) FromRotationDegrees (double degrees)

Public Attributes

- double **X11**
- double **X12**
- double **X21**
- double **X22**

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

- include/utils/geometry.h

5.33 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)

5.33.1 Detailed Description

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

5.33.2 Constructor & Destructor Documentation

5.33.2.1 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

5.33.3 Member Function Documentation

5.33.3.1 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.

Parameters

<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.

5.33.3.2 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

5.33.3.3 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)

5.33.3.4 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

5.34 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**

5.34.1 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

5.35 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

5.35.1 Detailed Description

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

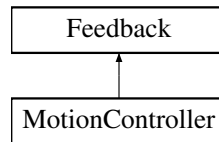
The documentation for this struct was generated from the following file:

- include/utils/trapezoid_profile.h

5.36 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](#) }

5.36.1 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:

$$\text{out} = \text{feedforward.calculate}(\text{motion_profile.get}(\text{time_s})) + \text{pid.get}(\text{motion_profile.get}(\text{time_s}))$$

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

Author

Ryan McGee

Date

7/13/2022

5.36.2 Constructor & Destructor Documentation

5.36.2.1 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
---------------	--

5.36.3 Member Function Documentation

5.36.3.1 get()

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

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

5.36.3.2 get_motion()

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

Returns

The current position, velocity and acceleration setpoints

5.36.3.3 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](#).

5.36.3.4 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](#).

5.36.3.5 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](#).

5.36.3.6 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 $[X=(pct \cdot kV \cdot V - kS), Y=(Acceleration)]$ along the movement, and since $kA \cdot Accel = pct \cdot kV \cdot V - kS$, 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

5.36.3.7 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

5.37 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](#) ()

5.37.1 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).

5.37.2 Constructor & Destructor Documentation

5.37.2.1 [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
--------------------	---

5.37.2.2 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

5.37.3 Member Function Documentation**5.37.3.1 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.
----------	--

5.37.3.2 get_average()

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

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

Returns

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

How many samples the average is made from

Returns

the number of samples used to calculate this average

5.37.3.3 get_size()

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

How many samples the average is made from

Returns

the number of samples used to calculate this average

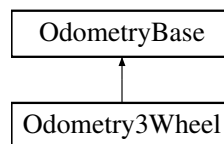
The documentation for this class was generated from the following files:

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

5.38 Odometry3Wheel Class Reference

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

Inheritance diagram for Odometry3Wheel:



Classes

- struct [odometry3wheel_cfg_t](#)

Public Member Functions

- [Odometry3Wheel](#) ([CustomEncoder](#) &lside_fwd, [CustomEncoder](#) &rside_fwd, [CustomEncoder](#) &off_axis, [odometry3wheel_cfg_t](#) &cfg, bool is_async=true)
- [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](#)

5.38.1 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

5.38.2 Constructor & Destructor Documentation

5.38.2.1 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

5.38.3 Member Function Documentation

5.38.3.1 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

5.38.3.2 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

5.39 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference

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

Public Attributes

- double [wheelbase_dist](#)
- double [off_axis_center_dist](#)
- double [wheel_diam](#)

5.39.1 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

5.39.2 Member Data Documentation

5.39.2.1 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

5.39.2.2 wheel_diam

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

the diameter of the tracking wheel

5.39.2.3 wheelbase_dist

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

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

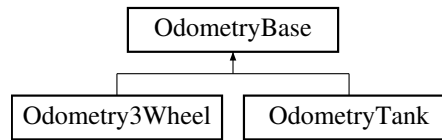
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry_3wheel.h

5.40 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](#)

5.40.1 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

5.40.2 Constructor & Destructor Documentation

5.40.2.1 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 update() manually
-----------------	---

5.40.3 Member Function Documentation

5.40.3.1 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 OdometryBase object
------------	--

Returns

Required integer return code. Unused.

5.40.3.2 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.

5.40.3.3 get_accel()

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

Get the current acceleration

Returns

the acceleration rate of the robot (inch/s²)

5.40.3.4 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/s²)

5.40.3.5 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)

5.40.3.6 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

5.40.3.7 `get_speed()`

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

Get the current speed

Returns

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

5.40.3.8 `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

5.40.3.9 `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

5.40.3.10 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](#).

5.40.3.11 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.

5.40.3.12 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](#).

5.40.4 Member Data Documentation

5.40.4.1 accel

```
double OdometryBase::accel [protected]
```

the rate at which we are accelerating (inch/s²)

5.40.4.2 ang_accel_deg

```
double OdometryBase::ang_accel_deg [protected]
```

the rate at which we are accelerating our turn (deg/s²)

5.40.4.3 ang_speed_deg

```
double OdometryBase::ang_speed_deg [protected]
```

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

5.40.4.4 current_pos

```
pose_t OdometryBase::current_pos [protected]
```

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

5.40.4.5 handle

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

handle to the vex task that is running the odometry code

5.40.4.6 mut

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

Mutex to control multithreading

5.40.4.7 speed

```
double OdometryBase::speed [protected]
```

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

5.40.4.8 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:

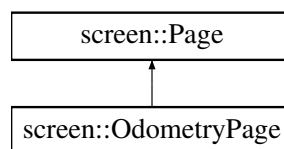
- include/subsystems/odometry/odometry_base.h
- src/subsystems/odometry/odometry_base.cpp

5.41 screen::OdometryPage Class Reference

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

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

Inheritance diagram for screen::OdometryPage:



Public Member Functions

- **OdometryPage** ([OdometryBase](#) &odom, double width, double height, bool do_trail)
- void [update](#) (bool was_pressed, int x, int y) override
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))
- void [draw](#) (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

5.41.1 Detailed Description

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

5.41.2 Member Function Documentation

5.41.2.1 draw()

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

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

Parameters

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

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

5.41.2.2 update()

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

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

Parameters

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

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

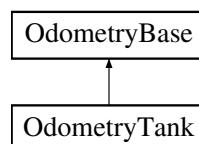
The documentation for this class was generated from the following files:

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

5.42 OdometryTank Class Reference

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

Inheritance diagram for OdometryTank:



Public Member Functions

- [OdometryTank](#) (vex::motor_group &left_side, vex::motor_group &right_side, [robot_specs_t](#) &config, vex::inertial *imu=NULL, bool is_async=true)
- [OdometryTank](#) ([CustomEncoder](#) &left_custom_enc, [CustomEncoder](#) &right_custom_enc, [robot_specs_t](#) &config, vex::inertial *imu=NULL, bool is_async=true)
- [OdometryTank](#) (vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, [robot_specs_t](#) &config, vex::inertial *imu=NULL, bool is_async=true)
- [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](#)

5.42.1 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.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 [OdometryTank\(\)](#) [1/3]

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

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

Parameters

<i>left_side</i>	The left motors
<i>right_side</i>	The right motors
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t 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 update() .

5.42.2.2 OdometryTank() [2/3]

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

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

Parameters

<i>left_custom_enc</i>	The left custom encoder
<i>right_custom_enc</i>	The right custom encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t 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 update() .

5.42.2.3 OdometryTank() [3/3]

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

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

Parameters

<i>left_vex_enc</i>	The left vex encoder
<i>right_vex_enc</i>	The right vex encoder
<i>config</i>	the specifications that supply the odometry with descriptions of the robot. See robot_specs_t 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 update() .

5.42.3 Member Function Documentation

5.42.3.1 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

newpos	the position the odometry will take
------------------------	-------------------------------------

Resets the position and rotational data to the input.

Reimplemented from [OdometryBase](#).

5.42.3.2 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](#).

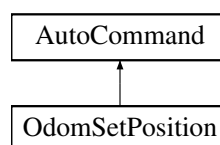
The documentation for this class was generated from the following files:

- include/subsystems/odometry/odometry_tank.h
- src/subsystems/odometry/odometry_tank.cpp

5.43 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

5.43.1 Detailed Description

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

5.43.2 Constructor & Destructor Documentation

5.43.2.1 [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

5.43.3 Member Function Documentation

5.43.3.1 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:

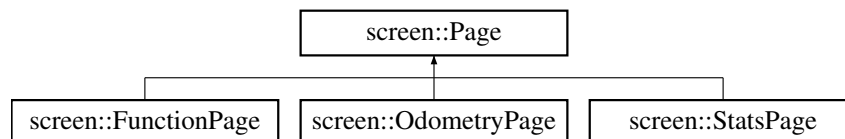
- include/utils/command_structure/drive_commands.h
- src/utils/command_structure/drive_commands.cpp

5.44 screen::Page Class Reference

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

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

Inheritance diagram for screen::Page:



Public Member Functions

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

5.44.1 Detailed Description

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

5.44.2 Member Function Documentation

5.44.2.1 draw()

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

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

Parameters

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

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

5.44.2.2 update()

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

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

Parameters

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

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

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

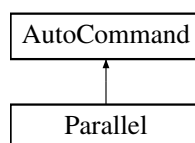
- include/subsystems/screen.h

5.45 Parallel Class Reference

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

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

Inheritance diagram for [Parallel](#):

**Public Member Functions**

- **Parallel** (std::initializer_list< [AutoCommand](#) * > cmds)
- 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

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

5.45.2 Member Function Documentation

5.45.2.1 `on_timeout()`

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

5.45.2.2 `run()`

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

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

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

- include/utls/command_structure/auto_command.h
- src/utls/command_structure/auto_command.cpp

5.46 parallel_runner_info Struct Reference

Public Attributes

- int **index**
- std::vector< vex::task * > * **runners**
- [AutoCommand](#) * **cmd**

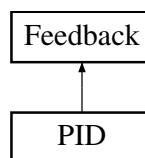
The documentation for this struct was generated from the following file:

- src/utis/command_structure/auto_command.cpp

5.47 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

5.47.1 Detailed Description

PID Class

Defines a standard feedback loop using the constants kP, kI, kD, deadband, and on_target_time. The formula is:

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

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

Author

Ryan McGee

Date

4/3/2020

5.47.2 Member Enumeration Documentation

5.47.2.1 ERROR_TYPE

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

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

5.47.3 Constructor & Destructor Documentation

5.47.3.1 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

5.47.4 Member Function Documentation

5.47.4.1 `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](#).

5.47.4.2 `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

5.47.4.3 `get_target()`

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

Get the [PID](#)'s target

Returns

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

5.47.4.4 `get_type()`

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

Reimplemented from [Feedback](#).

5.47.4.5 `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 PID . necessary to satisfy Feedback base
<i>set_pt</i>	sets the target of the PID controller

Implements [Feedback](#).

5.47.4.6 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](#).

5.47.4.7 reset()

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

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

5.47.4.8 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 PID controller will never command the output go below <i>lower</i>
<i>upper</i>	the upper limit. the PID 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](#).

5.47.4.9 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

5.47.4.10 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

5.48 PID::pid_config_t Struct Reference

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

Public Attributes

- double **p**
*proportional coeffecient $p * error()$*
- double **i**
*integral coeffecient $i * integral(error)$*
- 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.

5.48.1 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:

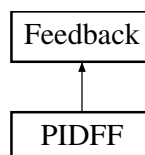
- deadband -
- on_target_time - for how long do we have to be at the target to stop As well, [pid_config_t](#) holds an error type which determines whether errors should be calculated as if the sensor position is a measure of distance or an angle

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

- include/utils/pid.h

5.49 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](#) }

5.49.1 Member Function Documentation

5.49.1.1 [get\(\)](#)

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

Returns

the last saved result from the feedback controller

Implements [Feedback](#).

5.49.1.2 [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](#).

5.49.1.3 [is_on_target\(\)](#)

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

Returns

true if the feedback controller has reached it's setpoint

Implements [Feedback](#).

5.49.1.4 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](#).

5.49.1.5 set_target()

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

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

Parameters

<i>set_← _pt</i>	Setpoint / target value
--------------------------------	-------------------------

5.49.1.6 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](#).

5.49.1.7 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 feedfoward

Returns

feedback loop result

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

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

5.50 point_t Struct Reference

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

Public Member Functions

- double [dist](#) (const [point_t](#) other) const
- [point_t operator+](#) (const [point_t](#) &other)
- [point_t operator-](#) (const [point_t](#) &other)
- [point_t operator*](#) (double s) const
- [point_t operator/](#) (double s) const
- [point_t operator-](#) () const
- [point_t operator+](#) () const
- bool [operator==](#) (const [point_t](#) &rhs)

Public Attributes

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

5.50.1 Detailed Description

Data structure representing an X,Y coordinate

5.50.2 Member Function Documentation

5.50.2.1 dist()

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

5.50.2.2 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)

5.50.2.3 operator-()

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

[Vector2D](#) subtraction operation on points

Parameters

<i>other</i>	the point_t 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

5.51 pose_t Struct Reference

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

Public Member Functions

- [point_t](#) `get_point ()`

Public Attributes

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

5.51.1 Detailed Description

Describes a single position and rotation

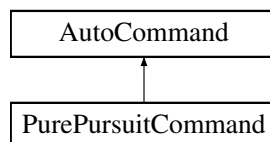
The documentation for this struct was generated from the following file:

- `include/utls/geometry.h`

5.52 PurePursuitCommand Class Reference

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

Inheritance diagram for PurePursuitCommand:



Public Member Functions

- [PurePursuitCommand](#) ([TankDrive](#) &drive_sys, [Feedback](#) &feedback, std::vector< [point_t](#) > path, direction↔ Type dir, double radius, 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

5.52.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the [TankDrive](#) class

5.52.2 Constructor & Destructor Documentation

5.52.2.1 PurePursuitCommand()

```
PurePursuitCommand::PurePursuitCommand (
    TankDrive & drive_sys,
    Feedback & feedback,
    std::vector< point\_t > path,
    directionType dir,
    double radius,
    double max_speed = 1 )
```

Construct a Pure Pursuit [AutoCommand](#)

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>radius</i>	How big the corner cutting should be - small values follow the path more closely
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

5.52.3 Member Function Documentation

5.52.3.1 on_timeout()

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

Reset the drive system when it times out

Reimplemented from [AutoCommand](#).

5.52.3.2 run()

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

Direct call to [TankDrive::pure_pursuit](#)

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

5.53 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

5.53.1 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

5.54 screen::ScreenData Struct Reference

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

Public Member Functions

- **ScreenData** (const std::vector< [Page](#) * > &m_pages, int m_page, vex::brain::lcd &m_screen)

Public Attributes

- std::vector< [Page](#) * > **pages**
- int **page** = 0
- vex::brain::lcd **screen**

5.54.1 Detailed Description

The [ScreenData](#) class holds the data that will be passed to the screen thread you probably shouldnt have to use it.

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

- src/subsystems/screen.cpp

5.55 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

5.55.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 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 save_to_disk

5.55.3 Member Function Documentation

5.55.3.1 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

5.55.3.2 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

5.55.3.3 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

5.55.3.4 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

5.55.3.5 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

5.55.3.6 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

5.55.3.7 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

5.55.3.8 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

5.55.3.9 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

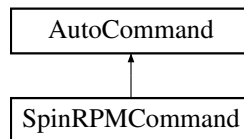
The documentation for this class was generated from the following files:

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

5.56 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

5.56.1 Detailed Description

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

5.56.2 Constructor & Destructor Documentation

5.56.2.1 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]

5.56.3 Member Function Documentation

5.56.3.1 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/utils/command_structure/flywheel_commands.h
- src/utils/command_structure/flywheel_commands.cpp

5.57 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**

5.57.1 Detailed Description

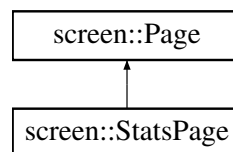
Represents a piece of a cubic spline with $s(x) = a(x-x_i)^3 + b(x-x_i)^2 + c(x-x_i) + d$ The `x_start` and `x_end` shows where the equation is valid.

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

- include/utils/pure_pursuit.h

5.58 screen::StatsPage Class Reference

Inheritance diagram for screen::StatsPage:



Public Member Functions

- **StatsPage** (std::map< std::string, vex::motor & > motors)
- void **update** (bool was_pressed, int x, int y) override
collect data, respond to screen input, do fast things (runs at 50hz even if you're not focused on this [Page](#) (only drawn page gets touch updates))
- void **draw** (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override
draw stored data to the screen (runs at 10 hz and only runs if this page is in front)

5.58.1 Member Function Documentation

5.58.1.1 draw()

```

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

```

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

Parameters

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

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

5.58.1.2 update()

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

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

Parameters

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

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

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

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

5.59 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)
- [AutoCommand](#) * [DriveToPointCmd](#) ([point_t](#) pt, vex::directionType dir=vex::forward, double max_speed=1.0)
- [AutoCommand](#) * [DriveToPointCmd](#) ([Feedback](#) &fb, [point_t](#) pt, vex::directionType dir=vex::forward, double max_speed=1.0)
- [AutoCommand](#) * [DriveForwardCmd](#) (double dist, vex::directionType dir=vex::forward, double max_speed=1.0)
- [AutoCommand](#) * [DriveForwardCmd](#) ([Feedback](#) &fb, double dist, vex::directionType dir=vex::forward, double max_speed=1.0)
- [AutoCommand](#) * [TurnToHeadingCmd](#) (double heading, double max_speed=1.0)
- [AutoCommand](#) * [TurnToHeadingCmd](#) ([Feedback](#) &fb, double heading, double max_speed=1.0)
- [AutoCommand](#) * [TurnDegreesCmd](#) (double degrees, double max_speed=1.0)

- [AutoCommand](#) * [TurnDegreesCmd](#) ([Feedback](#) &fb, double degrees, double max_speed=1.0)
- [AutoCommand](#) * [PurePursuitCmd](#) (std::vector< [point_t](#) > path, directionType dir, double radius, double max_speed=1)
- [AutoCommand](#) * [PurePursuitCmd](#) ([Feedback](#) &feedback, std::vector< [point_t](#) > path, directionType dir, double radius, double max_speed=1)
- void [stop](#) ()
- void [drive_tank](#) (double left, double right, int power=1)
- 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< [point_t](#) > path, directionType dir, double radius, [Feedback](#) &feedback, double max_speed=1)
- bool [pure_pursuit](#) (std::vector< [point_t](#) > path, directionType dir, double radius, double max_speed=1)

Static Public Member Functions

- static double [modify_inputs](#) (double input, int power=2)

5.59.1 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

5.59.2 Constructor & Destructor Documentation

5.59.2.1 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 robot_specs_t for more info
<i>odom</i>	an odometry system to track position and rotation. this is necessary to execute autonomous paths

5.59.3 Member Function Documentation

5.59.3.1 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

5.59.3.2 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

5.59.3.3 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

5.59.3.4 drive_tank()

```
void TankDrive::drive_tank (
    double left,
    double right,
    int power = 1 )
```

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 $left^{power}$, $right^{power}$
Generated by Isariver	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

5.59.3.5 `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

5.59.3.6 `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

5.59.3.7 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

input[^]power accounting for any sign issues that would arise with this naive solution

5.59.3.8 pure_pursuit() [1/2]

```
bool TankDrive::pure_pursuit (
    std::vector< point_t > path,
    directionType dir,
    double radius,
    double max_speed = 1 )
```

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Use the default drive feedback

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>radius</i>	How big the corner cutting should be - small values follow the path more closely
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.59.3.9 pure_pursuit() [2/2]

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

Drive the robot autonomously using a pure-pursuit algorithm - Input path with a set of waypoints - the robot will attempt to follow the points while cutting corners (radius) to save time (compared to stop / turn / start)

Parameters

<i>path</i>	The list of coordinates to follow, in order
<i>dir</i>	Run the bot forwards or backwards
<i>radius</i>	How big the corner cutting should be - small values follow the path more closely
<i>feedback</i>	The feedback controller determining speed
<i>max_speed</i>	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.59.3.10 reset_auto()

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

Reset the initialization for autonomous drive functions

5.59.3.11 stop()

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

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

5.59.3.12 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 to target number of degrees

5.59.3.13 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

5.59.3.14 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

5.59.3.15 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

5.60 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 elapsed.
- void [set_endpts](#) (double start, double end)
- void [set_accel](#) (double accel)
- void [set_max_v](#) (double max_v)
- double [get_movement_time](#) ()

5.60.1 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

5.60.2 Constructor & Destructor Documentation

5.60.2.1 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

5.60.3 Member Function Documentation

5.60.3.1 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

5.60.3.2 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

5.60.3.3 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
--------------	--------------------------------

5.60.3.4 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

5.60.3.5 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
--------------	--

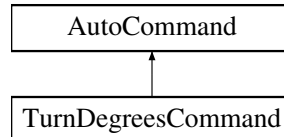
The documentation for this class was generated from the following files:

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

5.61 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

5.61.1 Detailed Description

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

5.61.2 Constructor & Destructor Documentation

5.61.2.1 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

5.61.3 Member Function Documentation

5.61.3.1 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](#).

5.61.3.2 run()

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

Run turn_degrees Overrides run from [AutoCommand](#)

Returns

true when execution is complete, false otherwise

Reimplemented from [AutoCommand](#).

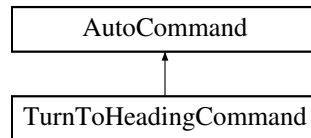
The documentation for this class was generated from the following files:

- include/utls/command_structure/drive_commands.h
- src/utls/command_structure/drive_commands.cpp

5.62 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

5.62.1 Detailed Description

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

5.62.2 Constructor & Destructor Documentation

5.62.2.1 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

5.62.3 Member Function Documentation

5.62.3.1 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](#).

5.62.3.2 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

5.63 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)

5.63.1 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

5.63.2 Constructor & Destructor Documentation

5.63.2.1 [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.

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

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

Construct a vector object from a cartesian point.

Parameters

<i>p</i>	point_t.x , point_t.y
----------	---

5.63.3 Member Function Documentation

5.63.3.1 `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.

5.63.3.2 `get_mag()`

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

Returns

the magnitude of the vector

Get the magnitude of the vector

5.63.3.3 `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.

5.63.3.4 `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.

5.63.3.5 normalize()

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

Changes the magnitude of the vector to 1

Returns

the normalized vector

Changes the magnetude of the vector to 1

5.63.3.6 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

5.63.3.7 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

5.63.3.8 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

5.63.3.9 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:

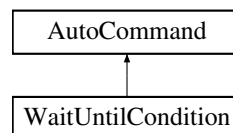
- include/utls/vector2d.h
- src/utls/vector2d.cpp

5.64 WaitUntilCondition Class Reference

Waits until the condition is true.

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

Inheritance diagram for WaitUntilCondition:



Public Member Functions

- **WaitUntilCondition** ([Condition](#) *cond)
- 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

5.64.1 Detailed Description

Waits until the condition is true.

5.64.2 Member Function Documentation

5.64.2.1 `run()`

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

Executes the command Overridden by child classes

Returns

true when the command is finished, false otherwise

Reimplemented from [AutoCommand](#).

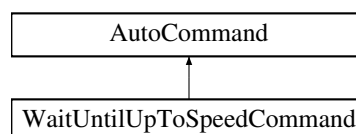
The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

5.65 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

5.65.1 Detailed Description

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

5.65.2 Constructor & Destructor Documentation

5.65.2.1 WaitUntilUpToSpeedCommand()

```
WaitUntilUpToSpeedCommand::WaitUntilUpToSpeedCommand (
    Flywheel & flywheel,
    int threshold_rpm )
```

Creat 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

5.65.3 Member Function Documentation

5.65.3.1 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 "../core/include/utils/command_structure/auto_command.h"
00019 #include <atomic>
00020
00021 using namespace vex;
00022
00023 class Flywheel
00024 {
00025     enum FlywheelControlStyle
00026     {
00027         PID_Feedforward,
00028         Feedforward,
00029         Take_Back_Half,
00030         Bang_Bang,
00031     };
00032
00033 public:
00034     // CONSTRUCTORS, GETTERS, AND SETTERS
00035     Flywheel(motor_group &motors, PID::pid_config_t &pid_config, FeedForward::ff_config_t &ff_config,
00036             const double ratio);
00037
00038     Flywheel(motor_group &motors, FeedForward::ff_config_t &ff_config, const double ratio);
00039
00040     Flywheel(motor_group &motors, double tbh_gain, const double ratio);
00041
00042     Flywheel(motor_group &motors, const double ratio);
00043
00044     double getDesiredRPM();
00045
00046     bool isTaskRunning();
00047
00048     motor_group *getMotors();
00049
00050     double measureRPM();
00051
00052     double getRPM();
00053     PID *getPID();
00054
00055     double getPIDValue();
00056
00057     double getFeedforwardValue();
00058
00059     double getTBHGain();
00060
00061     void setPIDTarget(double value);
00062
00063     void updatePID(double value);
00064
00065     // SPINNERS AND STOPPERS
00066
00067     void spin_raw(double speed, directionType dir = fwd);
00068
00069     void spin_manual(double speed, directionType dir = fwd);
00070
00071     void spinRPM(int rpm);
00072
00073     void stop();
00074
00075     void stopMotors();
00076
00077     void stopNonTasks();
00078
00079     AutoCommand *SpinRpmCmd(int rpm)
00080     {
00081         return new FunctionCommand([this]()
00082                                     {spinRPM(1000); return true; });
00083     }
00084
00085
00086
00087
00088
00089
00090
00091
00092
00093
00094
00095
00096
00097
00098
00099
00100
00101
00102
00103
00104
00105
00106
00107
00108
00109
00110
00111
00112
00113
00114
00115
00116
00117
00118
00119
00120
00121
00122
00123
00124
00125
00126
00127
00128
00129
00130
00131
00132
00133
00134
00135
00136
00137
00138
00139
00140
00141
00142
00143
00144
00145
00146
00147
00148
00149
00150
00151
00152
00153
00154
00155
00156
00157
00158
00159
00160
00161
00162
00163
00164
00165
00166
00167
00168
00169
00170
00171
00172
00173
00174
00175
00176
00177
00178

```

```

00179     AutoCommand *WaitUntilUpToSpeedCmd()
00180     {
00181         return new WaitUntilCondition(
00182             new FunctionCondition([this]() {
00183                 { return RPM == smoothedRPM; }));
00184     }
00185
00186 private:
00187     motor_group &motors;           // motors that make up the flywheel
00188     bool taskRunning = false;      // is the task (thread but not) currently running?
00189     PID pid;                       // PID on the flywheel
00190     FeedForward ff;               // FF constants for the flywheel
00191     double TBH_gain;              // TBH gain parameter for the flywheel
00192     double ratio;                 // multiplies the velocity by this value
00193     std::atomic<double> RPM;        // Desired RPM of the flywheel.
00194     task rpmTask;                 // task (thread but not) that handles spinning the wheel at a
    given RPM
00195     FlywheelControlStyle control_style; // how the flywheel should be controlled
00196     double smoothedRPM;
00197     MovingAverage RPM_avger;
00198 };

```

6.4 lift.h

```

00001 #pragma once
00002
00003 #include "vex.h"
00004 #include "../core/include/utlils/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 <<
00111         "\n";
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 <<
00120         "\n";
00121         is_async = true;
00122     } else
00123     {
00124         // Hold the lift at the last setpoint
00125         is_async = true;
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

```

```

00204     }
00205
00214     bool set_position(T pos)
00215     {
00216         this->setpoint = setpoint_map[pos];
00217         is_async = true;
00218
00219         return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00220     }
00221
00228     bool set_setpoint(double val)
00229     {
00230         this->setpoint = val;
00231         return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
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
00043     MecanumDrive(vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor
&right_rear,
00044                 vex::rotation *lateral_wheel=NULL, vex::inertial *imu=NULL, mecanumdrive_config_t
*config=NULL);
00045
00054     void drive_raw(double direction_deg, double magnitude, double rotation);
00055
00066     void drive(double left_y, double left_x, double right_x, int power=2);
00067
00080     bool auto_drive(double inches, double direction, double speed, bool gyro_correction=true);
00081
00092     bool auto_turn(double degrees, double speed, bool ignore_imu=false);
00093
00094     private:
00095
00096     vex::motor &left_front, &right_front, &left_rear, &right_rear;
00097
00098     mecanumdrive_config_t *config;
00099     vex::rotation *lateral_wheel;
00100     vex::inertial *imu;
00101
00102     PID *drive_pid = NULL;
00103     PID *drive_gyro_pid = NULL;
00104     PID *turn_pid = NULL;
00105
00106     bool init = true;
00107
00108 };

```

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

```

```

00032 class Odometry3Wheel : public OdometryBase
00033 {
00034     public:
00035
00040     typedef struct
00041     {
00042         double wheelbase_dist;
00043         double off_axis_center_dist;
00044         double wheel_diam;
00046     } odometry3wheel_cfg_t;
00047
00057     Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
odometry3wheel_cfg_t &cfg, bool is_async=true);
00058
00065     pose_t update() override;
00066
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
00012 class OdometryTank : public OdometryBase
00013 {
00014 public:
00015     OdometryTank(vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config,
00016                 vex::inertial *imu=NULL, bool is_async=true);
00017
00018     OdometryTank(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t
00019                 &config, vex::inertial *imu=NULL, bool is_async=true);
00020
00021     OdometryTank(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config,
00022                 vex::inertial *imu=NULL, bool is_async=true);
00023
00024     pose_t update() override;
00025
00026     void set_position(const pose_t &newpos=zero_pos) override;
00027
00028 private:
00029     static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff,
00030                                    double rside_diff, double angle_deg);
00031
00032     vex::motor_group *left_side, *right_side;
00033     CustomEncoder *left_custom_enc, *right_custom_enc;
00034     vex::encoder *left_vex_enc, *right_vex_enc;
00035     vex::inertial *imu;
00036     robot_specs_t &config;
00037
00038     double rotation_offset = 0;
00039 };

```

6.9 screen.h

```

00001 #pragma once
00002 #include "vex.h"
00003 #include <vector>
00004 #include <functional>
00005 #include <map>
00006 #include "../core/include/subsystems/odometry/odometry_base.h"
00007
00008 namespace screen
00009 {
00010     class Page
00011     {
00012     public:
00013         virtual void update(bool was_pressed, int x, int y);
00014         virtual void draw(vex::brain::lcd &screen, bool first_draw,
00015                          unsigned int frame_number);
00016     };
00017
00018     void start_screen(vex::brain::lcd &screen, std::vector<Page*> pages, int first_page = 0);
00019
00020     void stop_screen();
00021
00022     using update_func_t = std::function<void(bool, int, int)>;
00023
00024     using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;
00025 }

```



```

00050     class StatsPage : public Page
00051     {
00052     public:
00053         StatsPage(std::map<std::string, vex::motor &> motors);
00054         void update(bool was_pressed, int x, int y) override;
00055         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00056     private:
00057         void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int
00058 y, vex::brain::lcd &scr);
00059
00060         std::map<std::string, vex::motor &> motors;
00061         static const int y_start = 0;
00062         static const int per_column = 4;
00063         static const int row_height = 20;
00064         static const int row_width = 200;
00065     };
00066
00070     class OdometryPage : public Page
00071     {
00072     public:
00073         OdometryPage(OdometryBase &odom, double width, double height, bool do_trail);
00074         void update(bool was_pressed, int x, int y) override;
00075         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00076     private:
00077         static const int path_len = 40;
00078         static constexpr char const *field_filename = "vex_field_240p.png";
00080
00081         OdometryBase &odom;
00082         double width;
00083         double height;
00084         uint8_t *buf = nullptr;
00085         int buf_size = 0;
00086         pose_t path[path_len];
00087         int path_index = 0;
00088         bool do_trail;
00089     };
00090
00092     class FunctionPage : public Page
00093     {
00094     public:
00095         FunctionPage(update_func_t update_f, draw_func_t draw_t);
00096
00097         void update(bool was_pressed, int x, int y) override;
00098         void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00099     private:
00100         update_func_t update_f;
00101         draw_func_t draw_f;
00102     };
00103 };
00104
00105 }

```

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/include/utils/pure_pursuit.h"
00013 #include "../core/include/utils/command_structure/auto_command.h"
00014 #include <vector>
00015
00016 using namespace vex;
00017
00022 class TankDrive
00023 {
00024 public:
00032     TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
00033 *odom = NULL);
00034
00034     AutoCommand *DriveToPointCmd(point_t pt, vex::directionType dir = vex::forward, double max_speed =
00035 1.0);
00035     AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt, vex::directionType dir = vex::forward, double
00036 max_speed = 1.0);

```

```

00036
00037     AutoCommand *DriveForwardCmd(double dist, vex::directionType dir = vex::forward, double max_speed =
00038     1.0);
00038     AutoCommand *DriveForwardCmd(Feedback &fb, double dist, vex::directionType dir = vex::forward,
00039     double max_speed = 1.0);
00039
00040     AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0);
00041     AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading, double max_speed = 1.0);
00042
00043     AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0);
00044     AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees, double max_speed = 1.0);
00045
00046     AutoCommand *PurePursuitCmd(std::vector<point_t> path, directionType dir, double radius, double
00047     max_speed=1);
00047     AutoCommand *PurePursuitCmd(Feedback &feedback, std::vector<point_t> path, directionType dir, double
00048     radius, double max_speed=1);
00048
00052     void stop();
00053
00064     void drive_tank(double left, double right, int power=1);
00065
00076     void drive_arcade(double forward_back, double left_right, int power = 1);
00077
00088     bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed = 1);
00089
00098     bool drive_forward(double inches, directionType dir, double max_speed = 1);
00099
00110     bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1);
00111
00121     bool turn_degrees(double degrees, double max_speed = 1);
00122
00134     bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double max_speed
00135     = 1);
00135
00147     bool drive_to_point(double x, double y, vex::directionType dir, double max_speed = 1);
00148
00157     bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed = 1);
00165     bool turn_to_heading(double heading_deg, double max_speed = 1);
00166
00170     void reset_auto();
00171
00180     static double modify_inputs(double input, int power = 2);
00181
00194     bool pure_pursuit(std::vector<point_t> path, directionType dir, double radius, Feedback &feedback,
00195     double max_speed=1);
00209     bool pure_pursuit(std::vector<point_t> path, directionType dir, double radius, double max_speed=1);
00210
00211 private:
00212     motor_group &left_motors;
00213     motor_group &right_motors;
00214
00215     PID correction_pid;
00216     Feedback *drive_default_feedback = NULL;
00217     Feedback *turn_default_feedback = NULL;
00218
00219     OdometryBase *odometry;
00220
00221     robot_specs_t &config;
00222
00223     bool func_initialized = false;
00224     bool is_pure_pursuit = false;
00225 };

```

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;
00056 };

```

6.12 auto_command.h

```

00001
00007 #pragma once
00008
00009 #include "vex.h"
00010 #include <functional>
00011 #include <vector>
00012 #include <queue>
00013 #include <atomic>
00014
00015 class AutoCommand
00016 {
00017 public:
00018     static constexpr double default_timeout = 10.0;
00024     virtual bool run() { return true; }
00028     virtual void on_timeout() {}
00029     AutoCommand *withTimeout(double t_seconds)
00030     {
00031         if (this->timeout_seconds < 0)
00032         {
00033             // should never be timed out
00034             return this;
00035         }
00036         this->timeout_seconds = t_seconds;
00037         return this;
00038     }
00048     double timeout_seconds = default_timeout;
00049 };
00050
00055 class FunctionCommand : public AutoCommand
00056 {
00057 public:
00058     FunctionCommand(std::function<bool(void)> f) : f(f) {}
00059     bool run()
00060     {
00061         return f();
00062     }
00063 private:
00064     std::function<bool(void)> f;
00065 };
00066
00067
00077 class Condition
00078 {
00079 public:
00080     virtual bool test() = 0;
00081 };
00082
00084 class FunctionCondition : public Condition
00085 {
00086 public:
00087     FunctionCondition(
00088         std::function<bool(void)> cond, std::function<void(void)> timeout = []() {} ) : cond(cond),
00089         timeout(timeout)
00090     {
00091     }
00091     bool test() override;
00092 private:
00093     std::function<bool(void)> cond;
00094     std::function<void(void)> timeout;
00095 };
00096
00097
00099 class IfTimePassed : public Condition
00100 {
00101 public:

```

```

00102     IfTimePassed(double time_s);
00103     bool test() override;
00104
00105 private:
00106     double time_s;
00107     vex::timer tmr;
00108 };
00109
00111 class WaitUntilCondition : public AutoCommand
00112 {
00113 public:
00114     WaitUntilCondition(Condition *cond) : cond(cond) {}
00115     bool run() override
00116     {
00117         return cond->test();
00118     }
00119 private:
00120     Condition *cond;
00121 };
00122
00123
00126 class InOrder : public AutoCommand
00127 {
00128 public:
00129     InOrder(std::queue<AutoCommand *> cmds);
00130     InOrder(std::initializer_list<AutoCommand *> cmds);
00131     bool run() override;
00132     void on_timeout() override;
00133 private:
00134     AutoCommand *current_command = nullptr;
00135     std::queue<AutoCommand *> cmds;
00136     vex::timer tmr;
00137 };
00138
00139
00142 class Parallel : public AutoCommand
00143 {
00144 public:
00145     Parallel(std::initializer_list<AutoCommand *> cmds);
00146     bool run() override;
00147     void on_timeout() override;
00148 private:
00149     std::vector<AutoCommand *> cmds;
00150     std::vector<vex::task *> runners;
00151 };
00152
00153
00157 class Branch : public AutoCommand
00158 {
00159 public:
00160     Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
00161     ~Branch();
00162     bool run() override;
00163     void on_timeout() override;
00164 private:
00165     AutoCommand *false_choice;
00166     AutoCommand *true_choice;
00167     Condition *cond;
00168     bool choice = false;
00169     bool chosen = false;
00170     vex::timer tmr;
00171 };
00172
00173
00177 class Async : public AutoCommand
00178 {
00179 public:
00180     Async(AutoCommand *cmd) : cmd(cmd) {}
00181     bool run() override;
00182 private:
00183     AutoCommand *cmd = nullptr;
00184 };
00185

```

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:
00019     [[deprecated("Use list constructor instead.")] CommandController() : command_queue({}) {}
00020
00023     CommandController(std::initializer_list<AutoCommand *> cmds) : command_queue(cmds) {}
00029     [[deprecated("Use list constructor instead. If you need to make a decision before adding new
commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller")]]
void add(std::vector<AutoCommand *> cmds);
00030     void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00031
00042     [[deprecated("Use list constructor instead. If you need to make a decision before adding new
commands, use Branch (https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller")]]
void
00043     add(std::vector<AutoCommand *> cmds, double timeout_sec);
00050     void add_delay(int ms);
00051
00054     void add_cancel_func(std::function<bool(void)> true_if_cancel);
00055
00060     void run();
00061
00067     bool last_command_timed_out();
00068
00069 private:
00070     std::queue<AutoCommand *> command_queue;
00071     bool command_timed_out = false;
00072     std::function<bool()> should_cancel = []()
00073     { return false; };
00074 };

```

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 {
00038 public:
00039     DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
double max_speed=1);
00040
00046     bool run() override;
00050     void on_timeout() override;
00051
00052 private:
00053     // drive system to run the function on
00054     TankDrive &drive_sys;
00055
00056     // feedback controller to use

```

```

00057     Feedback &feedback;
00058
00059     // parameters for drive_forward
00060     double inches;
00061     directionType dir;
00062     double max_speed;
00063 };
00064
00069 class TurnDegreesCommand: public AutoCommand
00070 {
00071     public:
00072     TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed =
1);
00073
00079     bool run() override;
00083     void on_timeout() override;
00084
00085
00086     private:
00087     // drive system to run the function on
00088     TankDrive &drive_sys;
00089
00090     // feedback controller to use
00091     Feedback &feedback;
00092
00093     // parameters for turn_degrees
00094     double degrees;
00095     double max_speed;
00096 };
00097
00102 class DriveToPointCommand: public AutoCommand
00103 {
00104     public:
00105     DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType
dir, double max_speed = 1);
00106     DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
double max_speed=1);
00107
00113     bool run() override;
00114
00115     private:
00116     // drive system to run the function on
00117     TankDrive &drive_sys;
00118
00122     void on_timeout() override;
00123
00124
00125     // feedback controller to use
00126     Feedback &feedback;
00127
00128     // parameters for drive_to_point
00129     double x;
00130     double y;
00131     directionType dir;
00132     double max_speed;
00133 };
00134 };
00135
00141 class TurnToHeadingCommand: public AutoCommand
00142 {
00143     public:
00144     TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed =
1);
00145
00151     bool run() override;
00155     void on_timeout() override;
00156
00157
00158     private:
00159     // drive system to run the function on
00160     TankDrive &drive_sys;
00161
00162     // feedback controller to use
00163     Feedback &feedback;
00164
00165     // parameters for turn_to_heading
00166     double heading_deg;
00167     double max_speed;
00168 };
00169
00173 class PurePursuitCommand: public AutoCommand
00174 {
00175     public:
00185     PurePursuitCommand(TankDrive &drive_sys, Feedback &feedback, std::vector<point_t> path,
directionType dir, double radius, double max_speed=1);
00186
00190     bool run() override;

```

```

00191
00195 void on_timeout() override;
00196
00197 private:
00198 TankDrive &drive_sys;
00199 std::vector<point_t> path;
00200 directionType dir;
00201 double radius;
00202 Feedback &feedback;
00203 double max_speed;
00204
00205 };
00206
00211 class DriveStopCommand: public AutoCommand
00212 {
00213 public:
00214 DriveStopCommand(TankDrive &drive_sys);
00215
00221 bool run() override;
00222 void on_timeout() override;
00223
00224 private:
00225 // drive system to run the function on
00226 TankDrive &drive_sys;
00227 };
00228
00229
00230 // ==== ODOMETRY ====
00231
00236 class OdomSetPosition: public AutoCommand
00237 {
00238 public:
00244 OdomSetPosition(OdometryBase &odom, const pose_t &newpos=OdometryBase::zero_pos);
00245
00251 bool run() override;
00252
00253 private:
00254 // drive system with an odometry config
00255 OdometryBase &odom;
00256 pose_t newpos;
00257 };

```

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
00087 bool run() override;

```



```

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     [[deprecated("Use CommandController instead.")]]
00032     bool run(bool blocking);
00033
00038     [[deprecated("Use CommandController instead.")]]
00039     void add(state_ptr new_state);
00040
00045     [[deprecated("Use CommandController instead.")]]
00046     void add_async(state_ptr async_state);
00047
00052     [[deprecated("Use CommandController instead.")]]
00053     void add_delay(int ms);
00054
00055     private:
00056
00057     std::queue<state_ptr> state_list;
00058
00059 };

```

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) const
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     point_t operator*(double s) const
00049     {
00050         return {x * s, y * s};
00051     }
00052     point_t operator/(double s) const
00053     {
00054         return {x / s, y / s};
00055     }
00056
00057     point_t operator-() const
00058     {
00059         return {-x, -y};
00060     }
00061     point_t operator+() const
00062     {
00063         return {x, y};
00064     }
00065
00066     bool operator==(const point_t &rhs)
00067     {
00068         return x == rhs.x && y == rhs.y;
00069     }
00070 };
00071
00072 typedef struct
00073 {
00074     double x;
00075     double y;
00076     double rot;
00077
00078     point_t get_point()
00079     {
00080         return point_t{.x = x, .y = y};
00081     }
00082 } pose_t;
00083
00084 struct Mat2
00085 {
00086     double X11, X12;
00087     double X21, X22;
00088     point_t operator*(const point_t p) const
00089     {
00090         double outx = p.x * X11 + p.y * X12;
00091         double outy = p.x * X21 + p.y * X22;
00092         return {outx, outy};
00093     }
00094
00095     static Mat2 FromRotationDegrees(double degrees)
00096     {
00097         double rad = degrees * (M_PI / 180.0);
00098         double c = cos(rad);
00099         double s = sin(rad);
00100         return {c, -s, s, c};
00101     }
00102 };

```

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:
00014     GraphDrawer(vex::brain::lcd &screen, int num_samples, std::string x_label, std::string y_label,
00015         vex::color col, bool draw_border, double lower_bound, double upper_bound);
00016     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 Logln(const std::string &s);
00052
00056     void Logln(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 <vector>
00003 #include "math.h"
00004 #include "vex.h"
00005 #include "../core/include/utils/geometry.h"
00006
00007
00015 double clamp(double value, double low, double high);
00016
00023 double sign(double x);
00024
00025 double wrap_angle_deg(double input);
00026 double wrap_angle_rad(double input);
00027
00028 /*

```

```

00029 Calculates the variance of a set of numbers (needed for linear regression)
00030 https://en.wikipedia.org/wiki/Variance
00031 @param values the values for which the variance is taken
00032 @param mean the average of values
00033 */
00034 double variance(std::vector<double> const &values, double mean);
00035
00036
00037 /*
00038 Calculates the average of a vector of doubles
00039 @param values the list of values for which the average is taken
00040 */
00041 double mean(std::vector<double> const &values);
00042
00043 /*
00044 Calculates the covariance of a set of points (needed for linear regression)
00045 https://en.wikipedia.org/wiki/Covariance
00046
00047 @param points the points for which the covariance is taken
00048 @param meanx the mean value of all x coordinates in points
00049 @param meany the mean value of all y coordinates in points
00050 */
00051 double covariance(std::vector<std::pair<double, double> const &points, double meanx, double meany);
00052
00053 /*
00054 Calculates the slope and y intercept of the line of best fit for the data
00055 @param points the points for the data
00056 */
00057 std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double> const
&points);
00058
00059 double estimate_path_length(const std::vector<point_t> &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 #pragma once
00002 #include <vector>
00003
00016 class MovingAverage {
00017 public:
00018     /*
00019      * Create a moving average calculator with 0 as the default value
00020      *
00021      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
00022      * reading
00023      */
00023     MovingAverage(int buffer_size);
00024     /*
00025      * Create a moving average calculator with a specified default value
00026      * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
00027      * reading
00027      * @param starting_value The value that the average will be before any data is added
00028      */
00029     MovingAverage(int buffer_size, double starting_value);
00030
00031     /*
00032      * Add a reading to the buffer
00033      * Before:
00034      * [ 1 1 2 2 3 3] => 2
00035      * ^
00036      * After:
00037      * [ 2 1 2 2 3 3] => 2.16
00038      * ^
00039      * @param n    the sample that will be added to the moving average.
00040      */
00041     void add_entry(double n);
00042
00047     double get_average();
00048
00053     int get_size();
00054
00055 private:
00057     int buffer_index;           //index of the next value to be overridden
00058     std::vector<double> buffer; //all current data readings we've taken
00059     double current_avg;        //the current value of the data
00060
00061 };

```

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
00066     private:
00067
00068     FeedForward::ff_config_t &ff_cfg;
00069
00070     FeedForward ff;
00071

```

```

00072     double out;
00073     double lower_lim, upper_lim;
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 {
00011     struct spline
00012     {
00013         double a, b, c, d, x_start, x_end;
00014
00015         double getY(double x) {
00016             return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x - x_start) + d;
00017         }
00018     };
00019
00020     struct hermite_point
00021     {
00022         double x;
00023         double y;
00024         double dir;
00025         double mag;
00026
00027         point_t getPoint() const {
00028             return {x, y};
00029         }
00030
00031         Vector2D getTangent() const {
00032             return Vector2D(dir, mag);
00033         }
00034     };
00035
00036     extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
00037 point_t point2);
00038
00039     extern point_t get_lookahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00040
00041     extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00042
00043     extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
00044 weight_smooth, double tolerance);
00045
00046     extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00047
00048     extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double
00049 step);
00050
00051     extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
00052 radius);
00053
00054 }

```

6.29 serializer.h

```

00001 #pragma once
00002 #include <algorithm>
00003 #include <map>
00004 #include <string>
00005 #include <vector>
00006 #include <stdio.h>
00007
00008 const char serialization_separator = '$';
00009 const std::size_t MAX_FILE_SIZE = 4096;
00010
00011 class Serializer
00012 {
00013 private:
00014     bool flush_always;
00015     std::string filename;
00016     std::map<std::string, int> ints;
00017     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
00015 class Vector2D
00016 {
00017 public:
00024     Vector2D(double dir, double mag);
00025
00031     Vector2D(point_t p);
00032
00040     double get_dir() const;
00041
00045     double get_mag() const;
00046
00050     double get_x() const;
00051
00055     double get_y() const;
00056
00061     Vector2D normalize();
00062
00067     point_t point();
00068
00074     Vector2D operator*(const double &x);
00081     Vector2D operator+(const Vector2D &other);
00088     Vector2D operator-(const Vector2D &other);
00089
00090 private:
00091     double dir, mag;
00092
00093 };
00094
00095
00101 double deg2rad(double deg);
00102
00109 double rad2deg(double r);
```


Index

- accel
 - OdometryBase, [84](#)
- add
 - AutoChooser, [11](#)
 - CommandController, [17](#)
 - GenericAuto, [50](#)
- add_async
 - GenericAuto, [50](#)
- add_cancel_func
 - CommandController, [19](#)
- add_delay
 - CommandController, [19](#)
 - GenericAuto, [50](#)
- add_entry
 - MovingAverage, [74](#)
- add_sample
 - GraphDrawer, [52](#)
- ang_accel_deg
 - OdometryBase, [84](#)
- ang_speed_deg
 - OdometryBase, [84](#)
- Async, [9](#)
 - run, [10](#)
- auto_drive
 - MecanumDrive, [65](#)
- auto_turn
 - MecanumDrive, [66](#)
- AutoChooser, [10](#)
 - add, [11](#)
 - AutoChooser, [11](#)
 - brain, [12](#)
 - choice, [12](#)
 - get_choice, [11](#)
 - list, [12](#)
 - render, [11](#)
- AutoChooser::entry_t, [30](#)
 - height, [31](#)
 - name, [31](#)
 - width, [31](#)
 - x, [31](#)
 - y, [31](#)
- AutoCommand, [13](#)
 - on_timeout, [14](#)
 - run, [14](#)
 - timeout_seconds, [14](#)
- background_task
 - OdometryBase, [80](#)
- bool_or
 - Serializer, [108](#)
- brain
 - AutoChooser, [12](#)
- Branch, [15](#)
 - on_timeout, [15](#)
 - run, [15](#)
- calculate
 - FeedForward, [35](#)
 - TrapezoidProfile, [124](#)
- choice
 - AutoChooser, [12](#)
- CommandController, [16](#)
 - add, [17](#)
 - add_cancel_func, [19](#)
 - add_delay, [19](#)
 - CommandController, [16](#)
 - last_command_timed_out, [19](#)
 - run, [19](#)
- Condition, [20](#)
- control_continuous
 - Lift< T >, [57](#)
- control_manual
 - Lift< T >, [57](#)
- control_setpoints
 - Lift< T >, [57](#)
- Core, [1](#)
- current_pos
 - OdometryBase, [84](#)
- CustomEncoder, [20](#)
 - CustomEncoder, [21](#)
 - position, [21](#)
 - rotation, [21](#)
 - setPosition, [22](#)
 - setRotation, [22](#)
 - velocity, [22](#)
- DelayCommand, [23](#)
 - DelayCommand, [24](#)
 - run, [24](#)
- dist
 - point_t, [102](#)
- double_or
 - Serializer, [108](#)
- draw
 - GraphDrawer, [52](#)
 - screen::FunctionPage, [49](#)
 - screen::OdometryPage, [85](#)
 - screen::Page, [91](#)
 - screen::StatsPage, [113](#)
- drive

- MecanumDrive, 66
- drive_arcade
 - TankDrive, 116
- drive_forward
 - TankDrive, 116
- drive_raw
 - MecanumDrive, 67
- drive_tank
 - TankDrive, 117
- drive_to_point
 - TankDrive, 118
- DriveForwardCommand, 24
 - DriveForwardCommand, 25
 - on_timeout, 26
 - run, 26
- DriveStopCommand, 26
 - DriveStopCommand, 27
 - on_timeout, 27
 - run, 27
- DriveToPointCommand, 28
 - DriveToPointCommand, 29
 - run, 30
- end_async
 - OdometryBase, 80
- ERROR_TYPE
 - PID, 95
- Feedback, 31
 - get, 32
 - init, 32
 - is_on_target, 33
 - set_limits, 33
 - update, 33
- FeedForward, 34
 - calculate, 35
 - FeedForward, 34
- FeedForward::ff_config_t, 35
 - kA, 36
 - kG, 36
 - kS, 36
 - kV, 36
- Flywheel, 36
 - Flywheel, 37, 38
 - getDesiredRPM, 39
 - getFeedforwardValue, 39
 - getMotors, 39
 - getPID, 39
 - getPIDValue, 39
 - getRPM, 40
 - getTBHGain, 40
 - isTaskRunning, 40
 - measureRPM, 40
 - setPIDTarget, 40
 - spin_manual, 41
 - spin_raw, 41
 - spinRPM, 41
 - stop, 42
 - stopMotors, 42
 - stopNonTasks, 42
 - updatePID, 42
- FlywheelStopCommand, 42
 - FlywheelStopCommand, 43
 - run, 43
- FlywheelStopMotorsCommand, 44
 - FlywheelStopMotorsCommand, 44
 - run, 45
- FlywheelStopNonTasksCommand, 45
- FunctionCommand, 46
 - run, 47
- FunctionCondition, 47
 - test, 48
- FunctionPage
 - screen::FunctionPage, 48
- GenericAuto, 49
 - add, 50
 - add_async, 50
 - add_delay, 50
 - run, 50
- get
 - Feedback, 32
 - MotionController, 70
 - PID, 96
 - PIDFF, 100
- get_accel
 - OdometryBase, 81
- get_angular_accel_deg
 - OdometryBase, 81
- get_angular_speed_deg
 - OdometryBase, 81
- get_async
 - Lift< T >, 58
- get_average
 - MovingAverage, 74
- get_choice
 - AutoChooser, 11
- get_dir
 - Vector2D, 131
- get_error
 - PID, 96
- get_mag
 - Vector2D, 131
- get_motion
 - MotionController, 70
- get_movement_time
 - TrapezoidProfile, 125
- get_position
 - OdometryBase, 81
- get_setpoint
 - Lift< T >, 58
- get_size
 - MovingAverage, 74
- get_speed
 - OdometryBase, 81
- get_target
 - PID, 96
- get_type

- PID, 96
- get_x
 - Vector2D, 131
- get_y
 - Vector2D, 131
- getDesiredRPM
 - Flywheel, 39
- getFeedforwardValue
 - Flywheel, 39
- getMotors
 - Flywheel, 39
- getPID
 - Flywheel, 39
- getPIDValue
 - Flywheel, 39
- getRPM
 - Flywheel, 40
- getTBHGain
 - Flywheel, 40
- GraphDrawer, 51
 - add_sample, 52
 - draw, 52
 - GraphDrawer, 51
- handle
 - OdometryBase, 84
- height
 - AutoChooser::entry_t, 31
- hold
 - Lift< T >, 58
- home
 - Lift< T >, 58
- IfTimePassed, 53
 - test, 54
- include/robot_specs.h, 137
- include/subsystems/custom_encoder.h, 137
- include/subsystems/flywheel.h, 138
- include/subsystems/lift.h, 139
- include/subsystems/mecanum_drive.h, 142
- include/subsystems/odometry/odometry_3wheel.h, 142
- include/subsystems/odometry/odometry_base.h, 143
- include/subsystems/odometry/odometry_tank.h, 144
- include/subsystems/screen.h, 144
- include/subsystems/tank_drive.h, 145
- include/utils/auto_chooser.h, 146
- include/utils/command_structure/auto_command.h, 147
- include/utils/command_structure/command_controller.h, 148
- include/utils/command_structure/delay_command.h, 149
- include/utils/command_structure/drive_commands.h, 149
- include/utils/command_structure/flywheel_commands.h, 151
- include/utils/feedback_base.h, 152
- include/utils/feedforward.h, 152
- include/utils/generic_auto.h, 153
- include/utils/geometry.h, 153
- include/utils/graph_drawer.h, 154
- include/utils/logger.h, 155
- include/utils/math_util.h, 155
- include/utils/motion_controller.h, 156
- include/utils/moving_average.h, 157
- include/utils/pid.h, 157
- include/utils/pidff.h, 158
- include/utils/pure_pursuit.h, 159
- include/utils/serializer.h, 159
- include/utils/trapezoid_profile.h, 160
- include/utils/vector2d.h, 160
- init
 - Feedback, 32
 - MotionController, 71
 - PID, 96
 - PIDFF, 100
- InOrder, 54
 - on_timeout, 55
 - run, 55
- int_or
 - Serializer, 109
- is_on_target
 - Feedback, 33
 - MotionController, 71
 - PID, 97
 - PIDFF, 100
- isTaskRunning
 - Flywheel, 40
- kA
 - FeedForward::ff_config_t, 36
- kG
 - FeedForward::ff_config_t, 36
- kS
 - FeedForward::ff_config_t, 36
- kV
 - FeedForward::ff_config_t, 36
- last_command_timed_out
 - CommandController, 19
- Lift
 - Lift< T >, 56
- Lift< T >, 55
 - control_continuous, 57
 - control_manual, 57
 - control_setpoints, 57
 - get_async, 58
 - get_setpoint, 58
 - hold, 58
 - home, 58
 - Lift, 56
 - set_async, 58
 - set_position, 58
 - set_sensor_function, 59
 - set_sensor_reset, 59
 - set_setpoint, 59
- Lift< T >::lift_cfg_t, 60
- list
 - AutoChooser, 12

- Log
 - Logger, [62](#)
- Logf
 - Logger, [62](#)
- Logger, [60](#)
 - Log, [62](#)
 - Logf, [62](#)
 - Logger, [61](#)
 - LogIn, [63](#)
- LogIn
 - Logger, [63](#)
- Mat2, [64](#)
- measureRPM
 - Flywheel, [40](#)
- MecanumDrive, [64](#)
 - auto_drive, [65](#)
 - auto_turn, [66](#)
 - drive, [66](#)
 - drive_raw, [67](#)
 - MecanumDrive, [65](#)
- MecanumDrive::mecanumdrive_config_t, [68](#)
- modify_inputs
 - TankDrive, [119](#)
- motion_t, [68](#)
- MotionController, [69](#)
 - get, [70](#)
 - get_motion, [70](#)
 - init, [71](#)
 - is_on_target, [71](#)
 - MotionController, [70](#)
 - set_limits, [71](#)
 - tune_feedforward, [72](#)
 - update, [72](#)
- MotionController::m_profile_cfg_t, [63](#)
- MovingAverage, [73](#)
 - add_entry, [74](#)
 - get_average, [74](#)
 - get_size, [74](#)
 - MovingAverage, [73](#), [74](#)
- mut
 - OdometryBase, [84](#)
- name
 - AutoChooser::entry_t, [31](#)
- normalize
 - Vector2D, [131](#)
- Odometry3Wheel, [75](#)
 - Odometry3Wheel, [77](#)
 - tune, [77](#)
 - update, [77](#)
- Odometry3Wheel::odometry3wheel_cfg_t, [78](#)
 - off_axis_center_dist, [78](#)
 - wheel_diam, [78](#)
 - wheelbase_dist, [78](#)
- OdometryBase, [79](#)
 - accel, [84](#)
 - ang_accel_deg, [84](#)
 - ang_speed_deg, [84](#)
 - background_task, [80](#)
 - current_pos, [84](#)
 - end_async, [80](#)
 - get_accel, [81](#)
 - get_angular_accel_deg, [81](#)
 - get_angular_speed_deg, [81](#)
 - get_position, [81](#)
 - get_speed, [81](#)
 - handle, [84](#)
 - mut, [84](#)
 - OdometryBase, [80](#)
 - pos_diff, [82](#)
 - rot_diff, [82](#)
 - set_position, [82](#)
 - smallest_angle, [83](#)
 - speed, [84](#)
 - update, [83](#)
 - zero_pos, [84](#)
- OdometryTank, [86](#)
 - OdometryTank, [87](#), [88](#)
 - set_position, [89](#)
 - update, [89](#)
- OdomSetPosition, [89](#)
 - OdomSetPosition, [90](#)
 - run, [91](#)
- off_axis_center_dist
 - Odometry3Wheel::odometry3wheel_cfg_t, [78](#)
- on_timeout
 - AutoCommand, [14](#)
 - Branch, [15](#)
 - DriveForwardCommand, [26](#)
 - DriveStopCommand, [27](#)
 - InOrder, [55](#)
 - Parallel, [93](#)
 - PurePursuitCommand, [105](#)
 - TurnDegreesCommand, [127](#)
 - TurnToHeadingCommand, [129](#)
- operator+
 - point_t, [103](#)
 - Vector2D, [132](#)
- operator-
 - point_t, [103](#)
 - Vector2D, [132](#)
- operator*
 - Vector2D, [132](#)
- Parallel, [92](#)
 - on_timeout, [93](#)
 - run, [93](#)
- parallel_runner_info, [94](#)
- PID, [94](#)
 - ERROR_TYPE, [95](#)
 - get, [96](#)
 - get_error, [96](#)
 - get_target, [96](#)
 - get_type, [96](#)
 - init, [96](#)
 - is_on_target, [97](#)

- PID, 95
- reset, 97
- set_limits, 97
- set_target, 97
- update, 98
- PID::pid_config_t, 98
- PIDFF, 99
 - get, 100
 - init, 100
 - is_on_target, 100
 - set_limits, 100
 - set_target, 101
 - update, 101
- point
 - Vector2D, 133
- point_t, 102
 - dist, 102
 - operator+, 103
 - operator-, 103
- pos_diff
 - OdometryBase, 82
- pose_t, 103
- position
 - CustomEncoder, 21
- pure_pursuit
 - TankDrive, 120
- PurePursuit::hermite_point, 53
- PurePursuit::spline, 112
- PurePursuitCommand, 104
 - on_timeout, 105
 - PurePursuitCommand, 105
 - run, 105
- render
 - AutoChooser, 11
- reset
 - PID, 97
- reset_auto
 - TankDrive, 121
- robot_specs_t, 106
- rot_diff
 - OdometryBase, 82
- rotation
 - CustomEncoder, 21
- run
 - Async, 10
 - AutoCommand, 14
 - Branch, 15
 - CommandController, 19
 - DelayCommand, 24
 - DriveForwardCommand, 26
 - DriveStopCommand, 27
 - DriveToPointCommand, 30
 - FlywheelStopCommand, 43
 - FlywheelStopMotorsCommand, 45
 - FunctionCommand, 47
 - GenericAuto, 50
 - InOrder, 55
 - OdomSetPosition, 91
 - Parallel, 93
 - PurePursuitCommand, 105
 - SpinRPMCommand, 112
 - TurnDegreesCommand, 127
 - TurnToHeadingCommand, 129
 - WaitUntilCondition, 134
 - WaitUntilUpToSpeedCommand, 135
- save_to_disk
 - Serializer, 109
- screen::FunctionPage, 48
 - draw, 49
 - FunctionPage, 48
 - update, 49
- screen::OdometryPage, 85
 - draw, 85
 - update, 86
- screen::Page, 91
 - draw, 91
 - update, 92
- screen::ScreenData, 107
- screen::StatsPage, 113
 - draw, 113
 - update, 114
- Serializer, 107
 - bool_or, 108
 - double_or, 108
 - int_or, 109
 - save_to_disk, 109
 - Serializer, 108
 - set_bool, 109
 - set_double, 110
 - set_int, 110
 - set_string, 110
 - string_or, 110
- set_accel
 - TrapezoidProfile, 125
- set_async
 - Lift< T >, 58
- set_bool
 - Serializer, 109
- set_double
 - Serializer, 110
- set_endpts
 - TrapezoidProfile, 125
- set_int
 - Serializer, 110
- set_limits
 - Feedback, 33
 - MotionController, 71
 - PID, 97
 - PIDFF, 100
- set_max_v
 - TrapezoidProfile, 125
- set_position
 - Lift< T >, 58
 - OdometryBase, 82
 - OdometryTank, 89
- set_sensor_function

- Lift< T >, 59
- set_sensor_reset
 - Lift< T >, 59
- set_setpoint
 - Lift< T >, 59
- set_string
 - Serializer, 110
- set_target
 - PID, 97
 - PIDFF, 101
- setPIDTarget
 - Flywheel, 40
- setPosition
 - CustomEncoder, 22
- setRotation
 - CustomEncoder, 22
- smallest_angle
 - OdometryBase, 83
- speed
 - OdometryBase, 84
- spin_manual
 - Flywheel, 41
- spin_raw
 - Flywheel, 41
- spinRPM
 - Flywheel, 41
- SpinRPMCommand, 111
 - run, 112
 - SpinRPMCommand, 112
- stop
 - Flywheel, 42
 - TankDrive, 121
- stopMotors
 - Flywheel, 42
- stopNonTasks
 - Flywheel, 42
- string_or
 - Serializer, 110
- TankDrive, 114
 - drive_arcade, 116
 - drive_forward, 116
 - drive_tank, 117
 - drive_to_point, 118
 - modify_inputs, 119
 - pure_pursuit, 120
 - reset_auto, 121
 - stop, 121
 - TankDrive, 115
 - turn_degrees, 121
 - turn_to_heading, 122, 123
- test
 - FunctionCondition, 48
 - IfTimePassed, 54
- timeout_seconds
 - AutoCommand, 14
- TrapezoidProfile, 123
 - calculate, 124
 - get_movement_time, 125
 - set_accel, 125
 - set_endpts, 125
 - set_max_v, 125
 - TrapezoidProfile, 124
- tune
 - Odometry3Wheel, 77
- tune_feedforward
 - MotionController, 72
- turn_degrees
 - TankDrive, 121
- turn_to_heading
 - TankDrive, 122, 123
- TurnDegreesCommand, 126
 - on_timeout, 127
 - run, 127
 - TurnDegreesCommand, 127
- TurnToHeadingCommand, 128
 - on_timeout, 129
 - run, 129
 - TurnToHeadingCommand, 128
- update
 - Feedback, 33
 - MotionController, 72
 - Odometry3Wheel, 77
 - OdometryBase, 83
 - OdometryTank, 89
 - PID, 98
 - PIDFF, 101
 - screen::FunctionPage, 49
 - screen::OdometryPage, 86
 - screen::Page, 92
 - screen::StatsPage, 114
- updatePID
 - Flywheel, 42
- Vector2D, 129
 - get_dir, 131
 - get_mag, 131
 - get_x, 131
 - get_y, 131
 - normalize, 131
 - operator+, 132
 - operator-, 132
 - operator*, 132
 - point, 133
 - Vector2D, 130
- velocity
 - CustomEncoder, 22
- WaitUntilCondition, 133
 - run, 134
- WaitUntilUpToSpeedCommand, 134
 - run, 135
 - WaitUntilUpToSpeedCommand, 135
- wheel_diam
 - Odometry3Wheel::odometry3wheel_cfg_t, 78
- wheelbase_dist
 - Odometry3Wheel::odometry3wheel_cfg_t, 78

width

AutoChooser::entry_t, [31](#)

x

AutoChooser::entry_t, [31](#)

y

AutoChooser::entry_t, [31](#)

zero_pos

OdometryBase, [84](#)