RIT VEXU Core API

Generated by Doxygen 1.10.0

1 Core 1
1.1 Getting Started . . . 1
1.2 Features . . . 1

2 Hierarchical Index 3
2.1 Class Hierarchy . . . 3

3 Class Index 7
3.1 Class List . . . 7

4 File Index 11
4.1 File List . . . 11

5 Class Documentation 13
5.1 AndCondition Class Reference . . . 13
5.1.1 Member Function Documentation . . . 13
5.1.1.1 test() . . . 13
5.2 Async Class Reference . . . 14
5.2.1 Detailed Description . . . 14
5.2.2 Member Function Documentation . . . 15
5.2.2.1 run() . . . 15
5.3 AutoChooser Class Reference . . . 15
5.3.1 Detailed Description . . . 16
5.3.2 Constructor & Destructor Documentation . . . 16
5.3.2.1 AutoChooser() . . . 16
5.3.3 Member Function Documentation . . . 16
5.3.3.1 draw() . . . 16
5.3.3.2 get_choice() . . . 17
5.3.3.3 update() . . . 17
5.3.4 Member Data Documentation . . . 17
5.3.4.1 choice . . . 17
5.3.4.2 list . . . 17
5.4 AutoCommand Class Reference . . . 18
5.4.1 Member Function Documentation . . . 19
5.4.1.1 on_timeout() . . . 19
5.4.1.2 run() . . . 19
5.4.2 Member Data Documentation . . . 19
5.4.2.1 timeout_seconds . . . 19
5.5 BangBang Class Reference . . . 20
5.5.1 Member Function Documentation . . . 20
5.5.1.1 get() . . . 20
5.5.1.2 init() . . . 20
5.5.1.3 is_on_target() . . . 21
5.5.1.4 set_limits() . . . 21

5.5.1.5 update() . . . 21
5.6 BasicSolenoidSet Class Reference . . . 22
5.6.1 Detailed Description . . . 22
5.6.2 Constructor & Destructor Documentation . . . 22
5.6.2.1 BasicSolenoidSet() . . . 22
5.6.3 Member Function Documentation . . . 23
5.6.3.1 run() . . . 23
5.7 BasicSpinCommand Class Reference . . . 23
5.7.1 Detailed Description . . . 24
5.7.2 Constructor & Destructor Documentation . . . 24
5.7.2.1 BasicSpinCommand() . . . 24
5.7.3 Member Function Documentation . . . 25
5.7.3.1 run() . . . 25
5.8 BasicStopCommand Class Reference . . . 25
5.8.1 Detailed Description . . . 26
5.8.2 Constructor & Destructor Documentation . . . 26
5.8.2.1 BasicStopCommand() . . . 26
5.8.3 Member Function Documentation . . . 26
5.8.3.1 run() . . . 26
5.9 Branch Class Reference . . . 27
5.9.1 Detailed Description . . . 28
5.9.2 Member Function Documentation . . . 28
5.9.2.1 on_timeout() . . . 28
5.9.2.2 run() . . . 28
5.10 screen::ButtonConfig Struct Reference . . . 28
5.11 screen::ButtonWidget Class Reference . . . 28
5.11.1 Detailed Description . . . 29
5.11.2 Constructor & Destructor Documentation . . . 29
5.11.2.1 ButtonWidget() [1/2] . . . 29
5.11.2.2 ButtonWidget() [2/2] . . . 29
5.11.3 Member Function Documentation . . . 30
5.11.3.1 update() . . . 30
5.12 screen::CheckboxConfig Struct Reference . . . 30
5.13 CommandController Class Reference . . . 30
5.13.1 Detailed Description . . . 31
5.13.2 Constructor & Destructor Documentation . . . 31
5.13.2.1 CommandController() . . . 31
5.13.3 Member Function Documentation . . . 31
5.13.3.1 add() [1/3] . . . 31
5.13.3.2 add() [2/3] . . . 32
5.13.3.3 add() [3/3] . . . 32
5.13.3.4 add_cancel_func() . . . 32

5.13.3.5 add_delay() . . . 33
5.13.3.6 last_command_timed_out() . . . 33
5.13.3.7 run() . . . 33
5.14 Condition Class Reference . . . 33
5.14.1 Detailed Description . . . 34
5.15 CustomEncoder Class Reference . . . 34
5.15.1 Detailed Description . . . 34
5.15.2 Constructor & Destructor Documentation . . . 34
5.15.2.1 CustomEncoder() . . . 34
5.15.3 Member Function Documentation . . . 35
5.15.3.1 position() . . . 35
5.15.3.2 rotation() . . . 35
5.15.3.3 setPosition() . . . 35
5.15.3.4 setRotation() . . . 36
5.15.3.5 velocity() . . . 36
5.16 DelayCommand Class Reference . . . 36
5.16.1 Detailed Description . . . 37
5.16.2 Constructor & Destructor Documentation . . . 37
5.16.2.1 DelayCommand() . . . 37
5.16.3 Member Function Documentation . . . 37
5.16.3.1 run() . . . 37
5.17 DriveForwardCommand Class Reference . . . 38
5.17.1 Detailed Description . . . 39
5.17.2 Constructor & Destructor Documentation . . . 39
5.17.2.1 DriveForwardCommand() . . . 39
5.17.3 Member Function Documentation . . . 39
5.17.3.1 on_timeout() . . . 39
5.17.3.2 run() . . . 40
5.18 DriveStopCommand Class Reference . . . 40
5.18.1 Detailed Description . . . 41
5.18.2 Constructor & Destructor Documentation . . . 41
5.18.2.1 DriveStopCommand() . . . 41
5.18.3 Member Function Documentation . . . 41
5.18.3.1 on_timeout() . . . 41
5.18.3.2 run() . . . 41
5.19 DriveToPointCommand Class Reference . . . 42
5.19.1 Detailed Description . . . 43
5.19.2 Constructor & Destructor Documentation . . . 43
5.19.2.1 DriveToPointCommand() [1/2] . . . 43
5.19.2.2 DriveToPointCommand() [2/2] . . . 43
5.19.3 Member Function Documentation . . . 44
5.19.3.1 run() . . . 44

5.20 AutoChooser::entry_t Struct Reference . . . 44
5.20.1 Detailed Description . . . 44
5.20.2 Member Data Documentation . . . 44
5.20.2.1 name . . . 44
5.21 ExponentialMovingAverage Class Reference . . . 45
5.21.1 Detailed Description . . . 45
5.21.2 Constructor & Destructor Documentation . . . 45
5.21.2.1 ExponentialMovingAverage() [1/2] . . . 45
5.21.2.2 ExponentialMovingAverage() [2/2] . . . 46
5.21.3 Member Function Documentation . . . 46
5.21.3.1 add_entry() . . . 46
5.21.3.2 get_size() . . . 46
5.21.3.3 get_value() . . . 46
5.22 Feedback Class Reference . . . 47
5.22.1 Detailed Description . . . 47
5.22.2 Member Function Documentation . . . 48
5.22.2.1 get() . . . 48
5.22.2.2 init() . . . 48
5.22.2.3 is_on_target() . . . 48
5.22.2.4 set_limits() . . . 48
5.22.2.5 update() . . . 49
5.23 FeedForward Class Reference . . . 49
5.23.1 Detailed Description . . . 50
5.23.2 Constructor & Destructor Documentation . . . 50
5.23.2.1 FeedForward() . . . 50
5.23.3 Member Function Documentation . . . 50
5.23.3.1 calculate() . . . 50
5.24 FeedForward::ff_config_t Struct Reference . . . 51
5.24.1 Detailed Description . . . 51
5.24.2 Member Data Documentation . . . 51
5.24.2.1 kA . . . 51
5.24.2.2 kG . . . 52
5.24.2.3 kS . . . 52
5.24.2.4 kV . . . 52
5.25 Filter Class Reference . . . 52
5.25.1 Detailed Description . . . 52
5.25.2 Member Function Documentation . . . 53
5.25.2.1 add_entry() . . . 53
5.25.2.2 get_value() . . . 53
5.26 Flywheel Class Reference . . . 53
5.26.1 Detailed Description . . . 54
5.26.2 Constructor & Destructor Documentation . . . 54

5.26.2.1 Flywheel() . . . 54
5.26.3 Member Function Documentation . . . 54
5.26.3.1 get_motors() . . . 54
5.26.3.2 get_target() . . . 54
5.26.3.3 getRPM() . . . 55
5.26.3.4 is_on_target() . . . 55
5.26.3.5 Page() . . . 55
5.26.3.6 spin_manual() . . . 55
5.26.3.7 spin_rpm() . . . 56
5.26.3.8 SpinRpmCmd() . . . 56
5.26.3.9 stop() . . . 56
5.26.3.10 WaitUntilUpToSpeedCmd() . . . 56
5.26.4 Friends And Related Symbol Documentation . . . 57
5.26.4.1 spinRPMTask . . . 57
5.27 FlywheelPage Class Reference . . . 57
5.27.1 Member Function Documentation . . . 57
5.27.1.1 draw() . . . 57
5.27.1.2 update() . . . 58
5.28 FlywheelStopCommand Class Reference . . . 58
5.28.1 Detailed Description . . . 59
5.28.2 Constructor & Destructor Documentation . . . 59
5.28.2.1 FlywheelStopCommand() . . . 59
5.28.3 Member Function Documentation . . . 59
5.28.3.1 run() . . . 59
5.29 FlywheelStopMotorsCommand Class Reference . . . 59
5.29.1 Detailed Description . . . 60
5.29.2 Constructor & Destructor Documentation . . . 60
5.29.2.1 FlywheelStopMotorsCommand() . . . 60
5.29.3 Member Function Documentation . . . 60
5.29.3.1 run() . . . 60
5.30 FlywheelStopNonTasksCommand Class Reference . . . 61
5.30.1 Detailed Description . . . 61
5.31 FunctionCommand Class Reference . . . 62
5.31.1 Detailed Description . . . 62
5.31.2 Member Function Documentation . . . 63
5.31.2.1 run() . . . 63
5.32 FunctionCondition Class Reference . . . 63
5.32.1 Detailed Description . . . 63
5.32.2 Member Function Documentation . . . 64
5.32.2.1 test() . . . 64
5.33 screen::FunctionPage Class Reference . . . 64
5.33.1 Detailed Description . . . 64

5.33.2 Constructor & Destructor Documentation . . . 64
5.33.2.1 FunctionPage() . . . 64
5.33.3 Member Function Documentation . . . 65
5.33.3.1 draw() . . . 65
5.33.3.2 update() . . . 65
5.34 GenericAuto Class Reference . . . 65
5.34.1 Detailed Description . . . 66
5.34.2 Member Function Documentation . . . 66
5.34.2.1 add() . . . 66
5.34.2.2 add_async() . . . 66
5.34.2.3 add_delay() . . . 66
5.34.2.4 run() . . . 67
5.35 GraphDrawer Class Reference . . . 67
5.35.1 Constructor & Destructor Documentation . . . 67
5.35.1.1 GraphDrawer() . . . 67
5.35.2 Member Function Documentation . . . 68
5.35.2.1 add_samples() [1/2] . . . 68
5.35.2.2 add_samples() [2/2] . . . 68
5.35.2.3 draw() . . . 68
5.36 PurePursuit::hermite_point Struct Reference . . . 69
5.36.1 Detailed Description . . . 69
5.37 IfTimePassed Class Reference . . . 69
5.37.1 Detailed Description . . . 70
5.37.2 Member Function Documentation . . . 70
5.37.2.1 test() . . . 70
5.38 InOrder Class Reference . . . 70
5.38.1 Detailed Description . . . 71
5.38.2 Member Function Documentation . . . 71
5.38.2.1 on_timeout() . . . 71
5.38.2.2 run() . . . 71
5.39 screen::LabelConfig Struct Reference . . . 72
5.40 Lift< T > Class Template Reference . . . 72
5.40.1 Detailed Description . . . 72
5.40.2 Constructor & Destructor Documentation . . . 73
5.40.2.1 Lift() . . . 73
5.40.3 Member Function Documentation . . . 73
5.40.3.1 control_continuous() . . . 73
5.40.3.2 control_manual() . . . 73
5.40.3.3 control_setpoints() . . . 74
5.40.3.4 get_async() . . . 74
5.40.3.5 get_setpoint() . . . 74
5.40.3.6 hold() . . . 74

5.40.3.7 home() . . . 75
5.40.3.8 set_async() . . . 75
5.40.3.9 set_position() . . . 75
5.40.3.10 set_sensor_function() . . . 75
5.40.3.11 set_sensor_reset() . . . 76
5.40.3.12 set_setpoint() . . . 76
5.41 Lift< T >::lift_cfg_t Struct Reference . . . 76
5.41.1 Detailed Description . . . 76
5.42 Logger Class Reference . . . 77
5.42.1 Detailed Description . . . 77
5.42.2 Constructor & Destructor Documentation . . . 77
5.42.2.1 Logger() . . . 77
5.42.3 Member Function Documentation . . . 78
5.42.3.1 Log() [1/2] . . . 78
5.42.3.2 Log() [2/2] . . . 78
5.42.3.3 Logf() [1/2] . . . 78
5.42.3.4 Logf() [2/2] . . . 79
5.42.3.5 Logln() [1/2] . . . 79
5.42.3.6 Logln() [2/2] . . . 79
5.43 MotionController::m_profile_cfg_t Struct Reference . . . 79
5.43.1 Detailed Description . . . 80
5.44 Mat2 Struct Reference . . . 80
5.45 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage Class Reference . . . 80
5.45.1 Detailed Description . . . 81
5.45.2 Constructor & Destructor Documentation . . . 81
5.45.2.1 MaybeMessage() . . . 81
5.45.3 Member Function Documentation . . . 81
5.45.3.1 has_message() . . . 81
5.45.3.2 message() . . . 82
5.46 MecanumDrive Class Reference . . . 82
5.46.1 Detailed Description . . . 82
5.46.2 Constructor & Destructor Documentation . . . 82
5.46.2.1 MecanumDrive() . . . 82
5.46.3 Member Function Documentation . . . 83
5.46.3.1 auto_drive() . . . 83
5.46.3.2 auto_turn() . . . 83
5.46.3.3 drive() . . . 84
5.46.3.4 drive_raw() . . . 85
5.47 MecanumDrive::mecanumdrive_config_t Struct Reference . . . 85
5.47.1 Detailed Description . . . 85
5.48 motion_t Struct Reference . . . 85
5.48.1 Detailed Description . . . 86

5.49 MotionController Class Reference . . . 86
5.49.1 Detailed Description . . . 87
5.49.2 Constructor & Destructor Documentation . . . 87
5.49.2.1 MotionController() . . . 87
5.49.3 Member Function Documentation . . . 88
5.49.3.1 get() . . . 88
5.49.3.2 get_motion() . . . 88
5.49.3.3 init() . . . 88
5.49.3.4 is_on_target() . . . 88
5.49.3.5 set_limits() . . . 89
5.49.3.6 tune_feedforward() . . . 89
5.49.3.7 update() . . . 90
5.50 MovingAverage Class Reference . . . 90
5.50.1 Detailed Description . . . 90
5.50.2 Constructor & Destructor Documentation . . . 91
5.50.2.1 MovingAverage() [1/2] . . . 91
5.50.2.2 MovingAverage() [2/2] . . . 91
5.50.3 Member Function Documentation . . . 91
5.50.3.1 add_entry() . . . 91
5.50.3.2 get_size() . . . 91
5.50.3.3 get_value() . . . 92
5.51 Odometry3Wheel Class Reference . . . 92
5.51.1 Detailed Description . . . 94
5.51.2 Constructor & Destructor Documentation . . . 94
5.51.2.1 Odometry3Wheel() . . . 94
5.51.3 Member Function Documentation . . . 94
5.51.3.1 tune() . . . 94
5.51.3.2 update() . . . 95
5.52 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference . . . 95
5.52.1 Detailed Description . . . 95
5.52.2 Member Data Documentation . . . 96
5.52.2.1 off_axis_center_dist . . . 96
5.52.2.2 wheel_diam . . . 96
5.52.2.3 wheelbase_dist . . . 96
5.53 OdometryBase Class Reference . . . 96
5.53.1 Detailed Description . . . 97
5.53.2 Constructor & Destructor Documentation . . . 97
5.53.2.1 OdometryBase() . . . 97
5.53.3 Member Function Documentation . . . 98
5.53.3.1 background_task() . . . 98
5.53.3.2 end_async() . . . 98
5.53.3.3 get_accel() . . . 98

5.53.3.4 get_angular_accel_deg() . . . 98
5.53.3.5 get_angular_speed_deg() . . . 99
5.53.3.6 get_position() . . . 99
5.53.3.7 get_speed() . . . 99
5.53.3.8 pos_diff() . . . 99
5.53.3.9 rot_diff() . . . 100
5.53.3.10 set_position() . . . 100
5.53.3.11 smallest_angle() . . . 100
5.53.3.12 update() . . . 101
5.53.4 Member Data Documentation . . . 101
5.53.4.1 accel . . . 101
5.53.4.2 ang_accel_deg . . . 101
5.53.4.3 ang_speed_deg . . . 101
5.53.4.4 current_pos . . . 102
5.53.4.5 handle . . . 102
5.53.4.6 mut . . . 102
5.53.4.7 speed . . . 102
5.53.4.8 zero_pos . . . 102
5.54 screen::OdometryPage Class Reference . . . 102
5.54.1 Detailed Description . . . 103
5.54.2 Constructor & Destructor Documentation . . . 103
5.54.2.1 OdometryPage() . . . 103
5.54.3 Member Function Documentation . . . 103
5.54.3.1 draw() . . . 103
5.54.3.2 update() . . . 104
5.55 OdometryTank Class Reference . . . 104
5.55.1 Detailed Description . . . 105
5.55.2 Constructor & Destructor Documentation . . . 105
5.55.2.1 OdometryTank() [1/3] . . . 105
5.55.2.2 OdometryTank() [2/3] . . . 106
5.55.2.3 OdometryTank() [3/3] . . . 106
5.55.3 Member Function Documentation . . . 107
5.55.3.1 set_position() . . . 107
5.55.3.2 update() . . . 107
5.56 OdomSetPosition Class Reference . . . 107
5.56.1 Detailed Description . . . 108
5.56.2 Constructor & Destructor Documentation . . . 108
5.56.2.1 OdomSetPosition() . . . 108
5.56.3 Member Function Documentation . . . 109
5.56.3.1 run() . . . 109
5.57 OrCondition Class Reference . . . 109
5.57.1 Member Function Documentation . . . 109

5.57.1.1 test() . . . 109
5.58 screen::Page Class Reference . . . 110
5.58.1 Detailed Description . . . 110
5.58.2 Member Function Documentation . . . 110
5.58.2.1 draw() . . . 110
5.58.2.2 update() . . . 111
5.59 Parallel Class Reference . . . 111
5.59.1 Detailed Description . . . 112
5.59.2 Member Function Documentation . . . 112
5.59.2.1 on_timeout() . . . 112
5.59.2.2 run() . . . 112
5.60 parallel_runner_info Struct Reference . . . 113
5.61 PurePursuit::Path Class Reference . . . 113
5.61.1 Detailed Description . . . 113
5.61.2 Constructor & Destructor Documentation . . . 113
5.61.2.1 Path() . . . 113
5.61.3 Member Function Documentation . . . 114
5.61.3.1 get_points() . . . 114
5.61.3.2 get_radius() . . . 114
5.61.3.3 is_valid() . . . 114
5.62 PID Class Reference . . . 114
5.62.1 Detailed Description . . . 115
5.62.2 Member Enumeration Documentation . . . 115
5.62.2.1 ERROR_TYPE . . . 115
5.62.3 Constructor & Destructor Documentation . . . 115
5.62.3.1 PID() . . . 115
5.62.4 Member Function Documentation . . . 116
5.62.4.1 get() . . . 116
5.62.4.2 get_error() . . . 116
5.62.4.3 get_sensor_val() . . . 116
5.62.4.4 get_target() . . . 117
5.62.4.5 init() . . . 117
5.62.4.6 is_on_target() . . . 118
5.62.4.7 reset() . . . 118
5.62.4.8 set_limits() . . . 118
5.62.4.9 set_target() . . . 119
5.62.4.10 update() [1/2] . . . 119
5.62.4.11 update() [2/2] . . . 119
5.62.5 Member Data Documentation . . . 120
5.62.5.1 config . . . 120
5.63 PID::pid_config_t Struct Reference . . . 120
5.63.1 Detailed Description . . . 120

5.63.2 Member Data Documentation . . . 120
5.63.2.1 error_method . . . 120
5.63.2.2 on_target_time . . . 121
5.64 PIDFF Class Reference . . . 121
5.64.1 Member Function Documentation . . . 121
5.64.1.1 get() . . . 121
5.64.1.2 init() . . . 121
5.64.1.3 is_on_target() . . . 122
5.64.1.4 set_limits() . . . 122
5.64.1.5 set_target() . . . 122
5.64.1.6 update() [1/2] . . . 123
5.64.1.7 update() [2/2] . . . 123
5.65 screen::PIDPage Class Reference . . . 124
5.65.1 Detailed Description . . . 124
5.65.2 Constructor & Destructor Documentation . . . 124
5.65.2.1 PIDPage() . . . 124
5.65.3 Member Function Documentation . . . 125
5.65.3.1 draw() . . . 125
5.65.3.2 update() . . . 125
5.66 plm_frame_t Struct Reference . . . 125
5.67 plm_packet_t Struct Reference . . . 126
5.68 plm_plane_t Struct Reference . . . 126
5.69 plm_samples_t Struct Reference . . . 126
5.70 point_t Struct Reference . . . 126
5.70.1 Detailed Description . . . 127
5.70.2 Member Function Documentation . . . 127
5.70.2.1 dist() . . . 127
5.70.2.2 operator+() . . . 127
5.70.2.3 operator-() . . . 128
5.71 pose_t Struct Reference . . . 128
5.71.1 Detailed Description . . . 129
5.72 PurePursuitCommand Class Reference . . . 129
5.72.1 Detailed Description . . . 129
5.72.2 Constructor & Destructor Documentation . . . 130
5.72.2.1 PurePursuitCommand() . . . 130
5.72.3 Member Function Documentation . . . 130
5.72.3.1 on_timeout() . . . 130
5.72.3.2 run() . . . 130
5.73 Rect Struct Reference . . . 130
5.74 RepeatUntil Class Reference . . . 131
5.74.1 Constructor & Destructor Documentation . . . 132
5.74.1.1 RepeatUntil() [1/2] . . . 132

5.74.1.2 RepeatUntil() [2/2] . . . 133
5.74.2 Member Function Documentation . . . 133
5.74.2.1 on_timeout() . . . 133
5.74.2.2 run() . . . 133
5.75 robot_specs_t Struct Reference . . . 134
5.75.1 Detailed Description . . . 134
5.75.2 Member Data Documentation . . . 134
5.75.2.1 drive_correction_cutoff . . . 134
5.76 screen::ScreenData Struct Reference . . . 134
5.76.1 Detailed Description . . . 135
5.77 screen::ScreenRect Struct Reference . . . 135
5.78 Serializer Class Reference . . . 135
5.78.1 Detailed Description . . . 136
5.78.2 Constructor & Destructor Documentation . . . 136
5.78.2.1 Serializer() . . . 136
5.78.3 Member Function Documentation . . . 137
5.78.3.1 bool_or() . . . 137
5.78.3.2 double_or() . . . 137
5.78.3.3 int_or() . . . 137
5.78.3.4 save_to_disk() . . . 138
5.78.3.5 set_bool() . . . 138
5.78.3.6 set_double() . . . 138
5.78.3.7 set_int() . . . 138
5.78.3.8 set_string() . . . 139
5.78.3.9 string_or() . . . 139
5.79 screen::SizedWidget Struct Reference . . . 139
5.80 SliderCfg Struct Reference . . . 140
5.81 screen::SliderConfig Struct Reference . . . 140
5.82 screen::SliderWidget Class Reference . . . 140
5.82.1 Detailed Description . . . 140
5.82.2 Constructor & Destructor Documentation . . . 141
5.82.2.1 SliderWidget() . . . 141
5.82.3 Member Function Documentation . . . 142
5.82.3.1 update() . . . 142
5.83 SpinRPMCommand Class Reference . . . 142
5.83.1 Detailed Description . . . 143
5.83.2 Constructor & Destructor Documentation . . . 143
5.83.2.1 SpinRPMCommand() . . . 143
5.83.3 Member Function Documentation . . . 144
5.83.3.1 run() . . . 144
5.84 PurePursuit::spline Struct Reference . . . 144
5.84.1 Detailed Description . . . 144

5.85 StateMachine< System, IDType, Message, delay_ms, do_log >::State Struct Reference 145
5.85.1 Detailed Description 145
5.86 StateMachine< System, IDType, Message, delay_ms, do_log > Class Template Reference 145
5.86.1 Detailed Description 146
5.86.2 Constructor & Destructor Documentation 146
5.86.2.1 StateMachine() 146
5.86.3 Member Function Documentation 147
5.86.3.1 current_state() 147
5.86.3.2 send_message() 147
5.87 screen::StatsPage Class Reference 147
5.87.1 Detailed Description 148
5.87.2 Constructor & Destructor Documentation 148
5.87.2.1 StatsPage() 148
5.87.3 Member Function Documentation 148
5.87.3.1 draw() 148
5.87.3.2 update() 148
5.88 TakeBackHalf Class Reference 149
5.88.1 Detailed Description 149
5.88.2 Member Function Documentation 150
5.88.2.1 get() 150
5.88.2.2 init() 150
5.88.2.3 is_on_target() 150
5.88.2.4 set_limits() 150
5.88.2.5 update() 151
5.89 TankDrive Class Reference 151
5.89.1 Detailed Description 152
5.89.2 Member Enumeration Documentation 152
5.89.2.1 BrakeType 152
5.89.3 Constructor & Destructor Documentation 153
5.89.3.1 TankDrive() 153
5.89.4 Member Function Documentation 153
5.89.4.1 drive_arcade() 153
5.89.4.2 drive_forward() [1/2] 154
5.89.4.3 drive_forward() [2/2] 154
5.89.4.4 drive_tank() 155
5.89.4.5 drive_tank_raw() 155
5.89.4.6 drive_to_point() [1/2] 156
5.89.4.7 drive_to_point() [2/2] 157
5.89.4.8 modify_inputs() 157
5.89.4.9 pure_pursuit() [1/2] 158
5.89.4.10 pure_pursuit() [2/2] 159
5.89.4.11 reset_auto() 160

5.89.4.12 stop() . . . 160
5.89.4.13 turn_degrees() [1/2] . . . 160
5.89.4.14 turn_degrees() [2/2] . . . 160
5.89.4.15 turn_to_heading() [1/2] . . . 161
5.89.4.16 turn_to_heading() [2/2] . . . 162
5.90 screen::TextConfig Struct Reference . . . 163
5.91 TimesTestedCondition Class Reference . . . 163
5.91.1 Member Function Documentation . . . 163
5.91.1.1 test() . . . 163
5.92 TrapezoidProfile Class Reference . . . 163
5.92.1 Detailed Description . . . 164
5.92.2 Constructor & Destructor Documentation . . . 164
5.92.2.1 TrapezoidProfile() . . . 164
5.92.3 Member Function Documentation . . . 165
5.92.3.1 calculate() . . . 165
5.92.3.2 get_movement_time() . . . 165
5.92.3.3 set_accel() . . . 165
5.92.3.4 set_endpts() . . . 165
5.92.3.5 set_max_v() . . . 166
5.93 TurnDegreesCommand Class Reference . . . 166
5.93.1 Detailed Description . . . 167
5.93.2 Constructor & Destructor Documentation . . . 167
5.93.2.1 TurnDegreesCommand() . . . 167
5.93.3 Member Function Documentation . . . 167
5.93.3.1 on_timeout() . . . 167
5.93.3.2 run() . . . 168
5.94 TurnToHeadingCommand Class Reference . . . 168
5.94.1 Detailed Description . . . 169
5.94.2 Constructor & Destructor Documentation . . . 169
5.94.2.1 TurnToHeadingCommand() . . . 169
5.94.3 Member Function Documentation . . . 169
5.94.3.1 on_timeout() . . . 169
5.94.3.2 run() . . . 169
5.95 Vector2D Class Reference . . . 170
5.95.1 Detailed Description . . . 170
5.95.2 Constructor & Destructor Documentation . . . 170
5.95.2.1 Vector2D() [1/2] . . . 170
5.95.2.2 Vector2D() [2/2] . . . 171
5.95.3 Member Function Documentation . . . 171
5.95.3.1 get_dir() . . . 171
5.95.3.2 get_mag() . . . 171
5.95.3.3 get_x() . . . 171

5.95.3.4 get_y() . . . 172
5.95.3.5 normalize() . . . 172
5.95.3.6 operator∗() . . . 172
5.95.3.7 operator+() . . . 172
5.95.3.8 operator-() . . . 173
5.95.3.9 point() . . . 173
5.96 VideoPlayer Class Reference . . . 173
5.96.1 Member Function Documentation . . . 174
5.96.1.1 draw() . . . 174
5.96.1.2 update() . . . 174
5.97 WaitUntilCondition Class Reference . . . 175
5.97.1 Detailed Description . . . 175
5.97.2 Member Function Documentation . . . 176
5.97.2.1 run() . . . 176
5.98 WaitUntilUpToSpeedCommand Class Reference . . . 176
5.98.1 Detailed Description . . . 177
5.98.2 Constructor & Destructor Documentation . . . 177
5.98.2.1 WaitUntilUpToSpeedCommand() . . . 177
5.98.3 Member Function Documentation . . . 177
5.98.3.1 run() . . . 177
5.99 screen::WidgetConfig Struct Reference . . . 177
5.100 screen::WidgetPage Class Reference . . . 178
5.100.1 Member Function Documentation . . . 178
5.100.1.1 draw() . . . 178
5.100.1.2 update() . . . 179

6 File Documentation **181**

6.1 robot_specs.h . . . 181
6.2 custom_encoder.h . . . 181
6.3 flywheel.h . . . 181
6.4 pl_mpeg.h . . . 182
6.5 video.h . . . 230
6.6 layout.h . . . 230
6.7 lift.h . . . 230
6.8 mecanum_drive.h . . . 233
6.9 odometry_3wheel.h . . . 233
6.10 odometry_base.h . . . 234
6.11 odometry_tank.h . . . 235
6.12 screen.h . . . 235
6.13 tank_drive.h . . . 238
6.14 auto_chooser.h . . . 239
6.15 auto_command.h . . . 239

6.16 basic_command.h . . . 241
6.17 command_controller.h . . . 242
6.18 delay_command.h . . . 242
6.19 drive_commands.h . . . 243
6.20 flywheel_commands.h . . . 245
6.21 bang_bang.h . . . 245
6.22 feedback_base.h . . . 246
6.23 feedforward.h . . . 246
6.24 motion_controller.h . . . 246
6.25 pid.h . . . 247
6.26 pidff.h . . . 248
6.27 take_back_half.h . . . 248
6.28 trapezoid_profile.h . . . 249
6.29 generic_auto.h . . . 249
6.30 geometry.h . . . 250
6.31 graph_drawer.h . . . 251
6.32 logger.h . . . 251
6.33 math_util.h . . . 251
6.34 moving_average.h . . . 252
6.35 pure_pursuit.h . . . 253
6.36 serializer.h . . . 254
6.37 state_machine.h . . . 255
6.38 vector2d.h . . . 256

Index **257**

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:

AutoCommand . . . 18
- Async . . . 14
- BasicSolenoidSet . . . 22
- BasicSpinCommand . . . 23
- BasicStopCommand . . . 25
- Branch . . . 27
- DelayCommand . . . 36
- DriveForwardCommand . . . 38
- DriveStopCommand . . . 40
- DriveToPointCommand . . . 42
- FlywheelStopCommand . . . 58
- FlywheelStopMotorsCommand . . . 59
- FlywheelStopNonTasksCommand . . . 61
- FunctionCommand . . . 62
- InOrder . . . 70
- OdomSetPosition . . . 107
- Parallel . . . 111
- PurePursuitCommand . . . 129
- RepeatUntil . . . 131
- SpinRPMCommand . . . 142
- TurnDegreesCommand . . . 166
- TurnToHeadingCommand . . . 168
- WaitUntilCondition . . . 175
- WaitUntilUpToSpeedCommand . . . 176

screen::ButtonConfig . . . 28
screen::ButtonWidget . . . 28
screen::CheckboxConfig . . . 30
CommandController . . . 30
Condition . . . 33
- AndCondition . . . 13
- FunctionCondition . . . 63
- IfTimePassed . . . 69
- OrCondition . . . 109
- TimesTestedCondition . . . 163

vex::encoder

CustomEncoder . . . 34
AutoChooser::entry_t . . . 44
Feedback . . . 47
BangBang . . . 20
MotionController . . . 86
PID . . . 114
PIDFF . . . 121
TakeBackHalf . . . 149
FeedForward . . . 49
FeedForward::ff_config_t . . . 51
Filter . . . 52
ExponentialMovingAverage . . . 45
MovingAverage . . . 90
Flywheel . . . 53
GenericAuto . . . 65
GraphDrawer . . . 67
PurePursuit::hermite_point . . . 69
screen::LabelConfig . . . 72
Lift< T > . . . 72
Lift< T >::lift_cfg_t . . . 76
Logger . . . 77
MotionController::m_profile_cfg_t . . . 79
Mat2 . . . 80
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage . . . 80
MecanumDrive . . . 82
MecanumDrive::mecanumdrive_config_t . . . 85
motion_t . . . 85
Odometry3Wheel::odometry3wheel_cfg_t . . . 95
OdometryBase . . . 96
Odometry3Wheel . . . 92
OdometryTank . . . 104
screen::Page . . . 110
AutoChooser . . . 15
FlywheelPage . . . 57
VideoPlayer . . . 173
screen::FunctionPage . . . 64
screen::OdometryPage . . . 102
screen::PIDPage . . . 124
screen::StatsPage . . . 147
screen::WidgetPage . . . 178
parallel_runner_info . . . 113
PurePursuit::Path . . . 113
PID::pid_config_t . . . 120
plm_frame_t . . . 125
plm_packet_t . . . 126
plm_plane_t . . . 126
plm_samples_t . . . 126
point_t . . . 126
pose_t . . . 128
Rect . . . 130
robot_specs_t . . . 134
screen::ScreenData . . . 134
screen::ScreenRect . . . 135
Serializer . . . 135
screen::SizedWidget . . . 139
SliderCfg . . . 140
screen::SliderConfig . . . 140

screen::SliderWidget 140
PurePursuit::spline 144
StateMachine< System, IDType, Message, delay_ms, do_log >::State 145
StateMachine< System, IDType, Message, delay_ms, do_log > 145
TankDrive 151
screen::TextConfig 163
TrapezoidProfile 163
Vector2D 170
screen::WidgetConfig 177

Chapter 3

Class Index

3.1 Class List

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

AndCondition . . . 13
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 . . . 14
AutoChooser . . . 15
AutoCommand . . . 18
BangBang . . . 20
BasicSolenoidSet . . . 22
BasicSpinCommand . . . 23
BasicStopCommand . . . 25
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 . . . 27
screen::ButtonConfig . . . 28
screen::ButtonWidget
Widget that does something when you tap it. The function is only called once when you first tap it 28
screen::CheckboxConfig . . . 30
CommandController . . . 30
Condition . . . 33
CustomEncoder . . . 34
DelayCommand . . . 36
DriveForwardCommand . . . 38
DriveStopCommand . . . 40
DriveToPointCommand . . . 42
AutoChooser::entry_t . . . 44
ExponentialMovingAverage . . . 45
Feedback . . . 47
FeedForward . . . 49
FeedForward::ff_config_t . . . 51
Filter . . . 52
Flywheel . . . 53
FlywheelPage . . . 57
FlywheelStopCommand . . . 58
FlywheelStopMotorsCommand . . . 59

FlywheelStopNonTasksCommand . . . 61
FunctionCommand . . . 62
FunctionCondition
FunctionCondition is a quick and dirty Condition to wrap some expression that should be evaluated at runtime . . . 63
screen::FunctionPage
Simple page that stores no internal data. the draw and update functions use only global data rather than storing anything . . . 64
GenericAuto . . . 65
GraphDrawer . . . 67
PurePursuit::hermite_point . . . 69
IfTimePassed
IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time_s . . . 69
InOrder
InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts? . . . 70
screen::LabelConfig . . . 72
Lift< T > . . . 72
Lift< T >::lift_cfg_t . . . 76
Logger
Class to simplify writing to files . . . 77
MotionController::m_profile_cfg_t . . . 79
Mat2 . . . 80
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage
MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1 . . . 80
MecanumDrive . . . 82
MecanumDrive::mecanumdrive_config_t . . . 85
motion_t . . . 85
MotionController . . . 86
MovingAverage . . . 90
Odometry3Wheel . . . 92
Odometry3Wheel::odometry3wheel_cfg_t . . . 95
OdometryBase . . . 96
screen::OdometryPage
Page that shows odometry position and rotation and a map (if an sd card with the file is on) . . . 102
OdometryTank . . . 104
OdomSetPosition . . . 107
OrCondition . . . 109
screen::Page
Page describes one part of the screen slideshow . . . 110
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 . . . 111
parallel_runner_info . . . 113
PurePursuit::Path . . . 113
PID . . . 114
PID::pid_config_t . . . 120
PIDFF . . . 121
screen::PIDPage
PIDPage provides a way to tune a pid controller on the screen . . . 124
plm_frame_t . . . 125
plm_packet_t . . . 126
plm_plane_t . . . 126
plm_samples_t . . . 126
point_t . . . 126
pose_t . . . 128

PurePursuitCommand . . . 129
Rect . . . 130
RepeatUntil . . . 131
robot_specs_t . . . 134
screen::ScreenData
Holds the data that will be passed to the screen thread you probably shouldnt have to use it . . . 134
screen::ScreenRect . . . 135
Serializer
Serializes Arbitrary data to a file on the SD Card . . . 135
screen::SizedWidget . . . 139
SliderCfg . . . 140
screen::SliderConfig . . . 140
screen::SliderWidget
Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread . . . 140
SpinRPMCommand . . . 142
PurePursuit::spline . . . 144
StateMachine< System, IDType, Message, delay_ms, do_log >::State . . . 145
StateMachine< System, IDType, Message, delay_ms, do_log >
State Machine :)))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message . . . 145
screen::StatsPage
Draws motor stats and battery stats to the screen . . . 147
TakeBackHalf
A velocity controller . . . 149
TankDrive . . . 151
screen::TextConfig . . . 163
TimesTestedCondition . . . 163
TrapezoidProfile . . . 163
TurnDegreesCommand . . . 166
TurnToHeadingCommand . . . 168
Vector2D . . . 170
VideoPlayer . . . 173
WaitUntilCondition
Waits until the condition is true . . . 175
WaitUntilUpToSpeedCommand . . . 176
screen::WidgetConfig . . . 177
screen::WidgetPage . . . 178

Chapter 4

File Index

4.1 File List

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

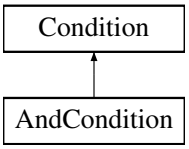
include/robot_specs.h . . . 181
include/subsystems/custom_encoder.h . . . 181
include/subsystems/flywheel.h . . . 181
include/subsystems/layout.h . . . 230
include/subsystems/lift.h . . . 230
include/subsystems/mecanum_drive.h . . . 233
include/subsystems/screen.h . . . 235
include/subsystems/tank_drive.h . . . 238
include/subsystems/fun/pl_mpeg.h . . . 182
include/subsystems/fun/video.h . . . 230
include/subsystems/odometry/odometry_3wheel.h . . . 233
include/subsystems/odometry/odometry_base.h . . . 234
include/subsystems/odometry/odometry_tank.h . . . 235
include/utils/auto_chooser.h . . . 239
include/utils/generic_auto.h . . . 249
include/utils/geometry.h . . . 250
include/utils/graph_drawer.h . . . 251
include/utils/logger.h . . . 251
include/utils/math_util.h . . . 251
include/utils/moving_average.h . . . 252
include/utils/pure_pursuit.h . . . 253
include/utils/serializer.h . . . 254
include/utils/state_machine.h . . . 255
include/utils/vector2d.h . . . 256
include/utils/command_structure/auto_command.h . . . 239
include/utils/command_structure/basic_command.h . . . 241
include/utils/command_structure/command_controller.h . . . 242
include/utils/command_structure/delay_command.h . . . 242
include/utils/command_structure/drive_commands.h . . . 243
include/utils/command_structure/flywheel_commands.h . . . 245
include/utils/controls/bang_bang.h . . . 245
include/utils/controls/feedback_base.h . . . 246
include/utils/controls/feedforward.h . . . 246
include/utils/controls/motion_controller.h . . . 246
include/utils/controls/pid.h . . . 247
include/utils/controls/pidff.h . . . 248
include/utils/controls/take_back_half.h . . . 248
include/utils/controls/trapezoid_profile.h . . . 249

Chapter 5

Class Documentation

5.1 AndCondition Class Reference

Inheritance diagram for AndCondition:



Public Member Functions

- **AndCondition** (Condition ∗A, Condition ∗B)
- bool test () override

Public Member Functions inherited from Condition

- Condition ∗ **Or** (Condition ∗b)
- Condition ∗ **And** (Condition ∗b)

5.1.1 Member Function Documentation

5.1.1.1 test()

```
bool AndCondition::test ( )  [inline], [override], [virtual]
```

Implements Condition.

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

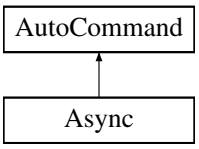
- src/utils/command_structure/auto_command.cpp

5.2 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)
- AutoCommand ∗ **withCancelCondition** (Condition ∗true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition ∗ **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.2.1 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.

5.2.2 Member Function Documentation

5.2.2.1 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.

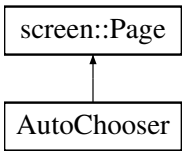
The documentation for this class was generated from the following files:

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

5.3 AutoChooser Class Reference

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

Inheritance diagram for AutoChooser:



Classes

- struct entry_t

Public Member Functions

- AutoChooser (std::vector< std::string > paths, size_t def=0)
- 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))
- void draw (vex::brain::lcd &, 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)
- size_t get_choice ()

Protected Attributes

- size_t choice
- std::vector< entry_t > list

Static Protected Attributes

- static const size_t **width** = 380
- static const size_t **height** = 220

5.3.1 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.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 AutoChooser()

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

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

Parameters

brain	the brain on which to draw the selection boxes

5.3.3 Member Function Documentation

5.3.3.1 draw()

```
void AutoChooser::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

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

Reimplemented from screen::Page.

5.3.3.2 get_choice()

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

Get the currently selected auto choice

Returns

the identifier to the auto path

Return the selected autonomous

5.3.3.3 update()

```
void AutoChooser::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

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

Reimplemented from screen::Page.

5.3.4 Member Data Documentation

5.3.4.1 choice

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

the current choice of auto

5.3.4.2 list

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

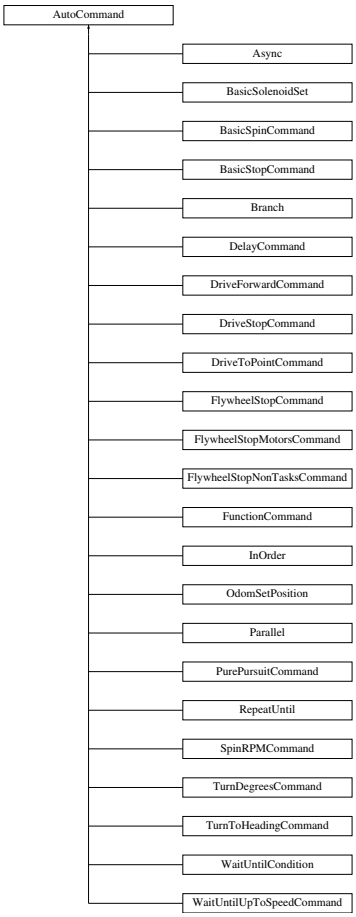
< a list of all possible auto choices

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

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

5.4 AutoCommand Class Reference

Inheritance diagram for AutoCommand:



Public Member Functions

- virtual bool run ()
- virtual void on_timeout ()
- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Public Attributes

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes

- static constexpr double **default_timeout** = 10.0

5.4.1 Member Function Documentation

5.4.1.1 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, RepeatUntil, DriveForwardCommand, TurnDegreesCommand, TurnToHeadingCommand, PurePursuitCommand, and DriveStopCommand.

5.4.1.2 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, RepeatUntil, BasicSpinCommand, BasicStopCommand, BasicSolenoidSet, DelayCommand, DriveForwardCommand, TurnDegreesCommand, DriveToPointCommand, TurnToHeadingCommand, PurePursuitCommand, DriveStopCommand, OdomSetPosition, SpinRPMCommand, WaitUntilUpToSpeedCommand, FlywheelStopCommand, and FlywheelStopMotorsCommand.

5.4.2 Member Data Documentation

5.4.2.1 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

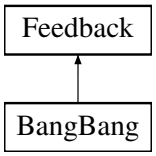
- a drive command hitting a wall and not being able to reach its target
- a command that waits until something is up to speed that never gets up to speed because of battery voltage
- something else...

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

- include/utils/command_structure/auto_command.h

5.5 BangBang Class Reference

Inheritance diagram for BangBang:



Public Member Functions

- **BangBang** (double thresshold, double low, double high)
- void init (double start_pt, double set_pt) override
- double update (double val) override
- double get () override
- void set_limits (double lower, double upper) override
- bool is_on_target () override

5.5.1 Member Function Documentation

5.5.1.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.5.1.2 init()

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

Initialize the feedback controller for a movement

Parameters

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	Movement starting velocity
end_vel	Movement ending velocity

Implements Feedback.

5.5.1.3 is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implements Feedback.

5.5.1.4 set_limits()

```
void BangBang::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

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.5.1.5 update()

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

Iterate the feedback loop once with an updated sensor value

Parameters

val	value from the sensor

Returns

feedback loop result

Implements Feedback.

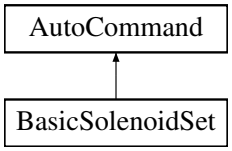
The documentation for this class was generated from the following files:

- include/utils/controls/bang_bang.h
- src/utils/controls/bang_bang.cpp

5.6 BasicSolenoidSet Class Reference

`#include <basic_command.h>`

Inheritance diagram for BasicSolenoidSet:



Public Member Functions

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

  Construct a new BasicSolenoidSet Command.
- bool run () override

  Runs the BasicSolenoidSet Overrides run command from AutoCommand.

Public Member Functions inherited from AutoCommand

- virtual void on_timeout ()
- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.6.1 Detailed Description

AutoCommand wrapper class for BasicSolenoidSet Using the Vex hardware functions

5.6.2 Constructor & Destructor Documentation

5.6.2.1 BasicSolenoidSet()

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

Construct a new BasicSolenoidSet Command.

Parameters

solenoid	Solenoid being set
setting	Setting of the solenoid in boolean (true,false)

5.6.3 Member Function Documentation

5.6.3.1 run()

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

Runs the BasicSolenoidSet Overrides run command from AutoCommand.

Returns

True Command runs once

Reimplemented from AutoCommand.

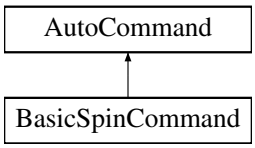
The documentation for this class was generated from the following files:

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

5.7 BasicSpinCommand Class Reference

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

Inheritance diagram for BasicSpinCommand:



Public Types

- enum **type** { **percent** , **voltage** , **veocity** }

Public Member Functions

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

  Construct a new BasicSpinCommand.
- bool run () override

  Runs the BasicSpinCommand Overrides run from Auto Command.

Public Member Functions inherited from AutoCommand

- virtual void on_timeout ()
- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.7.1 Detailed Description

AutoCommand wrapper class for BasicSpinCommand using the vex hardware functions

5.7.2 Constructor & Destructor Documentation

5.7.2.1 BasicSpinCommand()

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

Construct a new BasicSpinCommand.

a BasicMotorSpin Command

Parameters

motor	Motor to spin
direc	Direction of motor spin
setting	Power setting in volts,percentage,velocity
power	Value of desired power
motor	Motor port to spin
dir	Direction for spining
setting	Power setting in volts,percentage,velocity
power	Value of desired power

5.7.3 Member Function Documentation

5.7.3.1 run()

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

Runs the BasicSpinCommand Overrides run from Auto Command.

Run the BasicSpinCommand Overrides run from Auto Command.

Returns

True Async running command

True Command runs once

Reimplemented from AutoCommand.

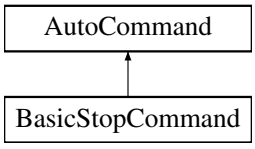
The documentation for this class was generated from the following files:

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

5.8 BasicStopCommand Class Reference

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

Inheritance diagram for BasicStopCommand:



Public Member Functions

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

  Construct a new BasicMotorStop Command.
- bool run () override

  Runs the BasicMotorStop Command Overrides run command from AutoCommand.

Public Member Functions inherited from AutoCommand

- virtual void on_timeout ()
- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.8.1 Detailed Description

AutoCommand wrapper class for BasicStopCommand Using the Vex hardware functions

5.8.2 Constructor & Destructor Documentation

5.8.2.1 BasicStopCommand()

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

Construct a new BasicMotorStop Command.

Construct a BasicMotorStop Command.

Parameters

motor	The motor to stop
setting	The brake setting for the motor
motor	Motor to stop
setting	Braketype setting brake,coast,hold

5.8.3 Member Function Documentation

5.8.3.1 run()

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

Runs the BasicMotorStop Command Overrides run command from AutoCommand.

Runs the BasicMotorStop command Ovverides run command from AutoCommand.

Returns

True Command runs once

Reimplemented from AutoCommand.

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

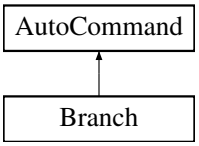
- include/utils/command_structure/basic_command.h
- src/utils/command_structure/basic_command.cpp

5.9 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.9.1 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.

5.9.2 Member Function Documentation

5.9.2.1 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.

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

5.10 screen::ButtonConfig Struct Reference

Public Attributes

- std::function< void()> **onclick**

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

- include/subsystems/screen.h

5.11 screen::ButtonWidget Class Reference

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

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

Public Member Functions

- ButtonWidget (std::function< void(void)> onpress, Rect rect, std::string name)

  Create a Button widget.
- ButtonWidget (void(*onpress)(), Rect rect, std::string name)

  Create a Button widget.
- bool update (bool was_pressed, int x, int y)

  responds to user input
- void **draw** (vex::brain::lcd &, bool first_draw, unsigned int frame_number)

  draws the button to the screen

5.11.1 Detailed Description

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

5.11.2 Constructor & Destructor Documentation

5.11.2.1 ButtonWidget() [1/2]

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

Create a Button widget.

Parameters

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

5.11.2.2 ButtonWidget() [2/2]

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

Create a Button widget.

Parameters

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

5.11.3 Member Function Documentation

5.11.3.1 update()

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

responds to user input

Parameters

was_pressed	if the screen is pressed
x	x position if the screen was pressed
y	y position if the screen was pressed

Returns

true if the button was pressed

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

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

5.12 screen::CheckboxConfig Struct Reference

Public Attributes

- std::function< void(bool)> **onupdate**

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

- include/subsystems/screen.h

5.13 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 ()

5.13.1 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.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 CommandController()

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

Create a CommandController with commands pre added. More can be added with CommandController::add()

Parameters

cmds	

5.13.3 Member Function Documentation

5.13.3.1 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

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

5.13.3.2 add() [2/3]

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

Adds a command to the queue

Parameters

cmd	the AutoCommand we want to add to our list
timeout_seconds	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

cmds	the AutoCommands we want to add to our list

5.13.3.3 add() [3/3]

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

Add multiple commands to the queue. No timeout here.

Parameters

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

Add multiple commands to the queue. No timeout here.

Parameters

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

5.13.3.4 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

true_if_cancel	a function that returns true when we want to cancel the command controller

5.13.3.5 add_delay()

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

Adds a command that will delay progression of the queue

Parameters

ms	- number of milliseconds to wait before continuing execution of autonomous

5.13.3.6 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

5.13.3.7 run()

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

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

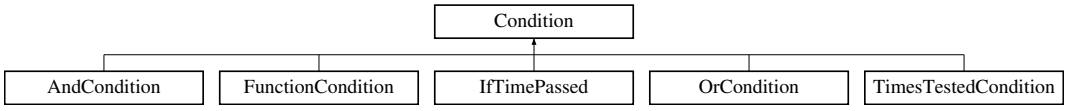
The documentation for this class was generated from the following files:

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

5.14 Condition Class Reference

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

Inheritance diagram for Condition:



Public Member Functions

- Condition * **Or** (Condition *b)
- Condition * **And** (Condition *b)
- virtual bool **test** ()=0

5.14.1 Detailed Description

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

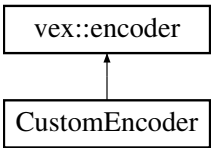
The documentation for this class was generated from the following files:

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

5.15 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)

5.15.1 Detailed Description

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

5.15.2 Constructor & Destructor Documentation

5.15.2.1 CustomEncoder()

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

Construct an encoder with a custom number of ticks

Parameters

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

5.15.3 Member Function Documentation

5.15.3.1 position()

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

get the position that the encoder is at

Parameters

units	the unit we want the return value to be in

Returns

the position of the encoder in the units specified

5.15.3.2 rotation()

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

get the rotation that the encoder is at

Parameters

units	the unit we want the return value to be in

Returns

the rotation of the encoder in the units specified

5.15.3.3 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

val	the numerical value of the position we are setting to
units	the unit of val

5.15.3.4 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

val	the numerical value of the angle we are setting to
units	the unit of val

5.15.3.5 velocity()

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

get the velocity that the encoder is moving at

Parameters

units	the unit we want the return value to be in

Returns

the velocity of the encoder in the units specified

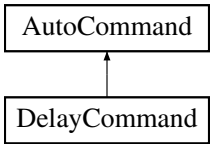
The documentation for this class was generated from the following files:

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

5.16 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.16.1 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

5.16.2 Constructor & Destructor Documentation

5.16.2.1 DelayCommand()

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

Construct a delay command

Parameters

ms	the number of milliseconds to delay for

5.16.3 Member Function Documentation

5.16.3.1 run()

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

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

Returns

true when complete

Reimplemented from AutoCommand.

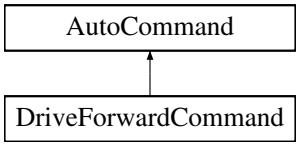
The documentation for this class was generated from the following file:

- include/utils/command_structure/delay_command.h

5.17 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, double end_speed=0)
- bool run () override
- void on_timeout () override

Public Member Functions inherited from AutoCommand

- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.17.1 Detailed Description

AutoCommand wrapper class for the drive_forward function in the TankDrive class

5.17.2 Constructor & Destructor Documentation

5.17.2.1 DriveForwardCommand()

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

File: drive_commands.h Desc: Holds all the AutoCommand subclasses that wrap (currently) TankDrive functions

Currently includes:

- drive_forward
- turn_degrees
- drive_to_point
- turn_to_heading
- stop

Also holds AutoCommand subclasses that wrap OdometryBase functions

Currently includes:

- set_position Construct a DriveForward Command

Parameters

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

5.17.3 Member Function Documentation

5.17.3.1 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.

5.17.3.2 run()

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

Run drive_forward Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

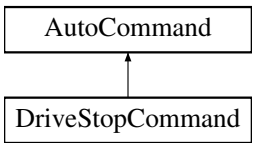
The documentation for this class was generated from the following files:

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

5.18 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.18.1 Detailed Description

AutoCommand wrapper class for the stop() function in the TankDrive class

5.18.2 Constructor & Destructor Documentation

5.18.2.1 DriveStopCommand()

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

Construct a DriveStop Command

Parameters

drive_sys	the drive system we are commanding

5.18.3 Member Function Documentation

5.18.3.1 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.

5.18.3.2 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.

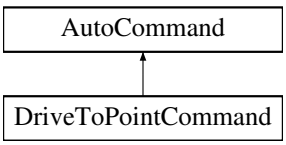
The documentation for this class was generated from the following files:

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

5.19 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, double end_speed=0)
- DriveToPointCommand (TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir, double max_speed=1, double end_speed=0)
- bool run () override

Public Member Functions inherited from AutoCommand

- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.19.1 Detailed Description

AutoCommand wrapper class for the drive_to_point function in the TankDrive class

5.19.2 Constructor & Destructor Documentation

5.19.2.1 DriveToPointCommand() [1/2]

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

Construct a DriveForward Command

Parameters

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

5.19.2.2 DriveToPointCommand() [2/2]

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

Construct a DriveForward Command

Parameters

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

5.19.3 Member Function Documentation

5.19.3.1 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

5.20 AutoChooser::entry_t Struct Reference

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

Public Attributes

- Rect **rect**
- std::string name

5.20.1 Detailed Description

entry_t is a datatype used to store information that the chooser knows about an auto selection button

5.20.2 Member Data Documentation

5.20.2.1 name

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

name of the auto repretsented by the block

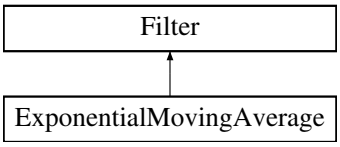
The documentation for this struct was generated from the following file:

- include/utils/auto_chooser.h

5.21 ExponentialMovingAverage Class Reference

`#include <moving_average.h>`

Inheritance diagram for ExponentialMovingAverage:



Public Member Functions

- ExponentialMovingAverage (int buffer_size)
- ExponentialMovingAverage (int buffer_size, double starting_value)
- void add_entry (double n) override
- double get_value () const override
- int get_size ()

5.21.1 Detailed Description

ExponentialMovingAverage

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

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

The ExponentialMovingAverage class provides an simple interface to do this smoothing from our noisy sensor values.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 ExponentialMovingAverage() [1/2]

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

Create a moving average calculator with 0 as the default value

Parameters

buffer_size	The size of the buffer. The number of samples that constitute a valid reading

5.21.2.2 ExponentialMovingAverage() [2/2]

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

Create a moving average calculator with a specified default value

Parameters

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

5.21.3 Member Function Documentation

5.21.3.1 add_entry()

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

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

Parameters

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

Implements Filter.

5.21.3.2 get_size()

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

How many samples the average is made from

Returns

the number of samples used to calculate this average

5.21.3.3 get_value()

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

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

Returns

the calculated average. sum(samples)/numsamples

How many samples the average is made from

Returns

the number of samples used to calculate this average

Implements Filter.

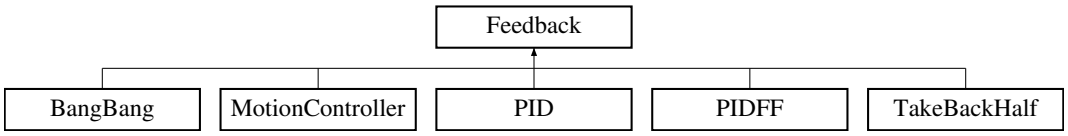
The documentation for this class was generated from the following files:

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

5.22 Feedback Class Reference

`#include <feedback_base.h>`

Inheritance diagram for Feedback:



Public Member Functions

- virtual void init (double start_pt, double set_pt)=0
- virtual double update (double val)=0
- virtual double get ()=0
- virtual void set_limits (double lower, double upper)=0
- virtual bool is_on_target ()=0

5.22.1 Detailed Description

Interface so that subsystems can easily switch between feedback loops

Author

Ryan McGee

Date

9/25/2022

5.22.2 Member Function Documentation

5.22.2.1 get()

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

Returns

the last saved result from the feedback controller

Implemented in BangBang, MotionController, PID, PIDFF, and TakeBackHalf.

5.22.2.2 init()

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

Initialize the feedback controller for a movement

Parameters

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	Movement starting velocity
end_vel	Movement ending velocity

Implemented in MotionController, TakeBackHalf, BangBang, PID, and PIDFF.

5.22.2.3 is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implemented in BangBang, MotionController, PID, PIDFF, and TakeBackHalf.

5.22.2.4 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

lower	Upper limit
upper	Lower limit

Implemented in BangBang, MotionController, PID, PIDFF, and TakeBackHalf.

5.22.2.5 update()

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

Iterate the feedback loop once with an updated sensor value

Parameters

val	value from the sensor

Returns

feedback loop result

Implemented in MotionController, PID, BangBang, PIDFF, and TakeBackHalf.

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

- include/utils/controls/feedback_base.h

5.23 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.

5.23.1 Detailed Description

FeedForward

Stores the feedfoward constants, and allows for quick computation. Feedfoward 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 reccommend 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

5.23.2 Constructor & Destructor Documentation

5.23.2.1 FeedForward()

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

Creates a FeedForward object.

Parameters

cfg	Configuration Struct for tuning

5.23.3 Member Function Documentation

5.23.3.1 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∗sgn(v) + kV∗v + kA∗a

Parameters

v	Requested velocity of system
a	Requested acceleration of system

Returns

A feedforward that should closely represent the system if tuned correctly

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

- include/utils/controls/feedforward.h

5.24 FeedForward::ff_config_t Struct Reference

`#include <feedforward.h>`

Public Attributes

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

5.24.1 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 $*$ desired velocity
- kA $*$ desired acceleration
- kG

5.24.2 Member Data Documentation

5.24.2.1 kA

`double FeedForward::ff_config_t::kA`

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

5.24.2.2 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.

5.24.2.3 kS

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

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

5.24.2.4 kV

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

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

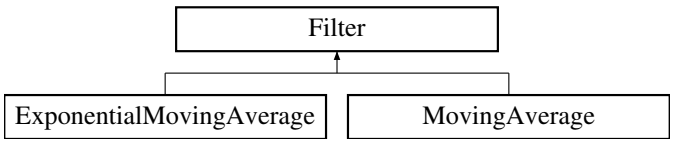
The documentation for this struct was generated from the following file:

- include/utils/controls/feedforward.h

5.25 Filter Class Reference

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

Inheritance diagram for Filter:



Public Member Functions

- virtual void add_entry (double n)=0
- virtual double get_value () const =0

5.25.1 Detailed Description

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

5.25.2 Member Function Documentation

5.25.2.1 add_entry()

```
virtual void Filter::add_entry (
            double n )  [pure virtual]
```

Implemented in MovingAverage, and ExponentialMovingAverage.

5.25.2.2 get_value()

```
virtual double Filter::get_value ( ) const  [pure virtual]
```

Implemented in MovingAverage, and ExponentialMovingAverage.

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

- include/utils/moving_average.h

5.26 Flywheel Class Reference

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

Public Member Functions

- Flywheel (vex::motor_group &motors, Feedback &feedback, FeedForward &helper, const double ratio, Filter &filt)
- double get_target () const
- double getRPM () const
- vex::motor_group & get_motors () const
- void spin_manual (double speed, directionType dir=fwd)
- void spin_rpm (double rpm)
- void stop ()
- bool is_on_target ()

  check if the feedback controller thinks the flywheel is on target
- screen::Page * Page () const

  Creates a page displaying info about the flywheel.
- AutoCommand * SpinRpmCmd (int rpm)

  Creates a new auto command to spin the flywheel at the desired velocity.
- AutoCommand * WaitUntilUpToSpeedCmd ()

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

Friends

- class **FlywheelPage**
- int spinRPMTask (void *wheelPointer)

5.26.1 Detailed Description

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

5.26.2 Constructor & Destructor Documentation

5.26.2.1 Flywheel()

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

Create the Flywheel object using PID + feedforward for control.

Parameters

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

5.26.3 Member Function Documentation

5.26.3.1 get_motors()

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

Returns the motors

Returns

the motors used to run the flywheel

5.26.3.2 get_target()

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

Return the target_rpm that the flywheel is currently trying to achieve

Returns

target_rpm the target rpm

Return the current value that the target_rpm should be set to

5.26.3.3 getRPM()

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

return the velocity of the flywheel

5.26.3.4 is_on_target()

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

check if the feedback controller thinks the flywheel is on target

Returns

true if on target

5.26.3.5 Page()

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

Creates a page displaying info about the flywheel.

Returns

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

5.26.3.6 spin_manual()

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

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

Parameters

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

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

Parameters

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

5.26.3.7 spin_rpm()

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

starts or sets the target_rpm thread at new value what control scheme is dependent on control_style

Parameters

rpm	- the target_rpm we want to spin at

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

Parameters

input_rpm	- set the current RPM

5.26.3.8 SpinRpmCmd()

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

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

Parameters

rpm	the rpm to spin at

Returns

an auto command to add to a command controller

5.26.3.9 stop()

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

Stops the motors. If manually spinning, this will do nothing just call spin_mainual(0.0) to send 0 volts

stop the RPM thread and the wheel

5.26.3.10 WaitUntilUpToSpeedCmd()

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

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

Returns

an auto command to add to a command controller

5.26.4 Friends And Related Symbol Documentation

5.26.4.1 spinRPMTask

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

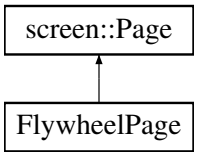
Runs a thread that keeps track of updating flywheel RPM and controlling it accordingly

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

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

5.27 FlywheelPage Class Reference

Inheritance diagram for FlywheelPage:



Public Member Functions

- **FlywheelPage** (const Flywheel &fw)
- void update (bool, int, int) override
- void draw (vex::brain::lcd &screen, bool, unsigned int) override

Static Public Attributes

- static const size_t **window_size** = 40

5.27.1 Member Function Documentation

5.27.1.1 draw()

```
void FlywheelPage::draw (
            vex::brain::lcd & screen,
            bool ,
            unsigned int  )  [inline], [override], [virtual]
```

See also

Page::draw

Reimplemented from screen::Page.

5.27.1.2 update()

```
void FlywheelPage::update (
            bool ,
            int ,
            int  )  [inline], [override], [virtual]
```

See also

Page::update

Reimplemented from screen::Page.

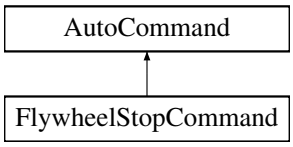
The documentation for this class was generated from the following file:

- src/subsystems/flywheel.cpp

5.28 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.28.1 Detailed Description

AutoCommand wrapper class for the stop function in the Flywheel class

5.28.2 Constructor & Destructor Documentation

5.28.2.1 FlywheelStopCommand()

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

Construct a FlywheelStopCommand

Parameters

flywheel	the flywheel system we are commanding

5.28.3 Member Function Documentation

5.28.3.1 run()

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

Run stop Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

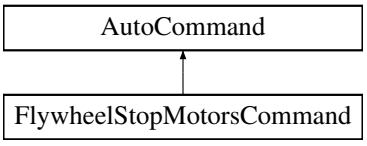
The documentation for this class was generated from the following files:

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

5.29 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.29.1 Detailed Description

AutoCommand wrapper class for the stopMotors function in the Flywheel class

5.29.2 Constructor & Destructor Documentation

5.29.2.1 FlywheelStopMotorsCommand()

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

Construct a FlywheeStopMotors Command

Parameters

flywheel	the flywheel system we are commanding

5.29.3 Member Function Documentation

5.29.3.1 run()

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

Run stop Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

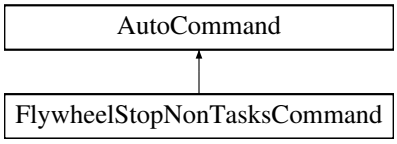
The documentation for this class was generated from the following files:

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

5.30 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.30.1 Detailed Description

AutoCommand wrapper class for the stopNonTasks function in the Flywheel class

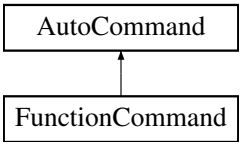
The documentation for this class was generated from the following files:

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

5.31 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.31.1 Detailed Description

FunctionCommand is fun and good way to do simple things Printing, launching nukes, and other quick and dirty one time things

5.31.2 Member Function Documentation

5.31.2.1 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:

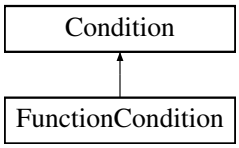
- include/utils/command_structure/auto_command.h

5.32 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

Public Member Functions inherited from Condition

- Condition * **Or** (Condition *b)
- Condition * **And** (Condition *b)

5.32.1 Detailed Description

FunctionCondition is a quick and dirty Condition to wrap some expression that should be evaluated at runtime.

5.32.2 Member Function Documentation

5.32.2.1 test()

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

Implements Condition.

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

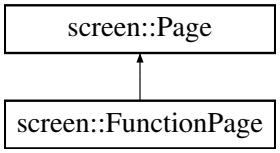
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.33 screen::FunctionPage Class Reference

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

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

Inheritance diagram for screen::FunctionPage:



Public Member Functions

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

5.33.1 Detailed Description

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

5.33.2 Constructor & Destructor Documentation

5.33.2.1 FunctionPage()

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

Creates a function page.

FunctionPage.

Parameters

update_f	the function called every tick to respond to user input or do data collection
draw_t	the function called to draw to the screen
update_f	drawing function
draw_f	drawing function

5.33.3 Member Function Documentation

5.33.3.1 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

See also

Page::draw

Reimplemented from screen::Page.

5.33.3.2 update()

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

update uses the supplied update function to update this page

See also

Page::update

Reimplemented from screen::Page.

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

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

5.34 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)

5.34.1 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

5.34.2 Member Function Documentation

5.34.2.1 add()

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

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

Parameters

new_state	the function to run

5.34.2.2 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

async_state	the function to run

5.34.2.3 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

ms	how long to wait in milliseconds

5.34.2.4 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

blocking	Whether or not to block the thread until all states have run

Returns

true after all states have finished.

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

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

5.35 GraphDrawer Class Reference

Public Member Functions

- GraphDrawer (int num_samples, double lower_bound, double upper_bound, std::vector< vex::color > colors, size_t num_series=1)

  Creates a graph drawer with the specified number of series (each series is a separate line)
- void add_samples (std::vector< point_t > sample)
- void add_samples (std::vector< double > sample)
- void draw (vex::brain::lcd &screen, int x, int y, int width, int height)

5.35.1 Constructor & Destructor Documentation

5.35.1.1 GraphDrawer()

```
GraphDrawer::GraphDrawer (
            int num_samples,
            double lower_bound,
            double upper_bound,
            std::vector< vex::color > colors,
            size_t num_series = 1 )
```

Creates a graph drawer with the specified number of series (each series is a separate line)

Parameters

num_samples	the number of samples to graph at a time (40 will graph the last 40 data points)
lower_bound	the bottom of the window when displaying (if upper_bound = lower_bound, auto calculate bounds)
upper_bound	the top of the window when displaying (if upper_bound = lower_bound, auto calculate bounds)
colors	the colors of the series. must be of size num_series
num_series	the number of series to graph

5.35.2 Member Function Documentation

5.35.2.1 add_samples() [1/2]

```
void GraphDrawer::add_samples (
            std::vector< double > sample )
```

add_samples adds a point to the graph, removing one from the back

Parameters

sample	a y coordinate of the next point to graph, the x coordinate is gotten from vex::timer::system(); (time in ms)

5.35.2.2 add_samples() [2/2]

```
void GraphDrawer::add_samples (
            std::vector< point_t > new_samples )
```

add_samples adds a point to the graph, removing one from the back

Parameters

sample	an x, y coordinate of the next point to graph

5.35.2.3 draw()

```
void GraphDrawer::draw (
            vex::brain::lcd & screen,
            int x,
            int y,
            int width,
            int height )
```

draws the graph to the screen in the constructor

Parameters

x	x position of the top left of the graphed region

Parameters

y	y position of the top left of the graphed region
width	the width of the graphed region
height	the height of the graphed region

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

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

5.36 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**

5.36.1 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:

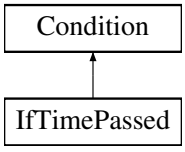
- include/utils/pure_pursuit.h

5.37 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

Public Member Functions inherited from Condition

- Condition * **Or** (Condition *b)
- Condition * **And** (Condition *b)

5.37.1 Detailed Description

IfTimePassed tests based on time since the command controller was constructed. Returns true if elapsed time > time_s.

5.37.2 Member Function Documentation

5.37.2.1 test()

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

Implements Condition.

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

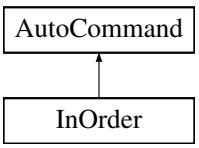
- include/utils/command_structure/auto_command.h
- src/utils/command_structure/auto_command.cpp

5.38 InOrder Class Reference

InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

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

Inheritance diagram for InOrder:



Public Member Functions

- **InOrder** (const InOrder &other)=default
- **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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.38.1 Detailed Description

InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

InOrder runs its commands sequentially then continues. How to handle timeout in this case. Automatically set it to sum of commands timouts?

5.38.2 Member Function Documentation

5.38.2.1 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.

5.38.2.2 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

5.39 screen::LabelConfig Struct Reference

Public Attributes

- std::string **label**

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

- include/subsystems/screen.h

5.40 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))

5.40.1 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

5.40.2 Constructor & Destructor Documentation

5.40.2.1 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

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

5.40.3 Member Function Documentation

5.40.3.1 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

up_ctrl	Button controlling the "UP" motion
down_ctrl	Button controlling the "DOWN" motion

5.40.3.2 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

up_btn	Raise the lift when true
down_btn	Lower the lift when true
volt_up	Motor voltage when raising the lift
volt_down	Motor voltage when lowering the lift

5.40.3.3 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

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

5.40.3.4 get_async()

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

Returns

whether or not the background thread is running the lift

5.40.3.5 get_setpoint()

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

Returns

The current setpoint for the lift

5.40.3.6 hold()

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

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

5.40.3.7 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.

5.40.3.8 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

val	Whether or not the background thread should run the lift

5.40.3.9 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

pos	A lift position enum type

Returns

True if the pid has reached the setpoint

5.40.3.10 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

fn_ptr	Pointer to custom sensor function

5.40.3.11 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

5.40.3.12 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

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

Returns

True if the pid has reached the setpoint

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

- include/subsystems/lift.h

5.41 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**

5.41.1 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

5.42 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 LogIn (const std::string &s)
  Write a string and newline to the log.
- void LogIn (LogLevel level, const std::string &s)
  Write a string and a newline to the log with a loglevel.
- void Logf (const char *fmt,...)
  Write a formatted string to the log.
- void Logf (LogLevel level, const char *fmt,...)
  Write a formatted string to the log with a loglevel.

Static Public Attributes

- static constexpr int **MAX_FORMAT_LEN** = 512
  maximum size for a string to be before it's written

5.42.1 Detailed Description

Class to simplify writing to files.

5.42.2 Constructor & Destructor Documentation

5.42.2.1 Logger()

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

Create a logger that will save to a file.

Parameters

filename	the file to save to

5.42.3 Member Function Documentation

5.42.3.1 Log() [1/2]

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

Write a string to the log.

Parameters

s	the string to write

5.42.3.2 Log() [2/2]

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

Write a string to the log with a loglevel.

Parameters

level	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
s	the string to write

5.42.3.3 Logf() [1/2]

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

Write a formatted string to the log.

Parameters

fmt	the format string (like printf)
...	the args

5.42.3.4 Logf() [2/2]

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

Write a formatted string to the log with a loglevel.

Parameters

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

5.42.3.5 Logln() [1/2]

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

Write a string and newline to the log.

Parameters

s	the string to write

5.42.3.6 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

level	the level to write. DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME
s	the string to write

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

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

5.43 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

5.43.1 Detailed Description

m_profile_config holds all data the motion controller uses to plan paths When motion pofile 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/controls/motion_controller.h

5.44 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

5.45 StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage Class Reference

MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.

`#include <state_machine.h>`

Public Member Functions

- **MaybeMessage** ()

  Empty message - when theres no message.
- MaybeMessage (Message msg)

  Create a maybemessage with a message.
- bool has_message ()

  check if the message is here
- Message message ()

  Get the message stored. The return value is invalid unless has_message returned true.

5.45.1 Detailed Description

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

MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.

5.45.2 Constructor & Destructor Documentation

5.45.2.1 MaybeMessage()

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

Create a maybemessage with a message.

Parameters

msg	the message to hold on to

5.45.3 Member Function Documentation

5.45.3.1 has_message()

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

check if the message is here

Returns

true if there is a message

5.45.3.2 message()

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

Get the message stored. The return value is invalid unless has_message returned true.

Returns

The message if it exists. Undefined otherwise

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

- include/utils/state_machine.h

5.46 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)

5.46.1 Detailed Description

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

5.46.2 Constructor & Destructor Documentation

5.46.2.1 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

5.46.3 Member Function Documentation

5.46.3.1 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

inches	How far the robot should drive, in inches
direction	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
speed	The maximum speed the robot should travel, in percent: -1.0->+1.0
gyro_correction	=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

inches	How far the robot should drive, in inches
direction	What direction the robot should travel in, in degrees. 0 is forward, +/-180 is reverse, clockwise is positive.
speed	The maximum speed the robot should travel, in percent: -1.0->+1.0
gyro_correction	= 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.

5.46.3.2 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

degrees	How many degrees to rotate the robot. Clockwise postive.
speed	What percentage to run the motors at: 0.0 -> 1.0
ignore_imu	=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

degrees	How many degrees to rotate the robot. Clockwise postive.
speed	What percentage to run the motors at: 0.0 -> 1.0
ignore_imu	= 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

5.46.3.3 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

left_y	left joystick, Y axis (forward / backwards)
left_x	left joystick, X axis (strafe left / right)
right_x	right joystick, X axis (rotation left / right)
power	=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

left_y	left joystick, Y axis (forward / backwards)
left_x	left joystick, X axis (strafe left / right)
right_x	right joystick, X axis (rotation left / right)
power	= 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)

5.46.3.4 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

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

5.47 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**

5.47.1 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

5.48 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

5.48.1 Detailed Description

motion_t is a description of 1 dimensional motion at a point in time.

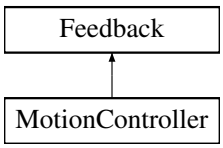
The documentation for this struct was generated from the following file:

- include/utils/controls/trapezoid_profile.h

5.49 MotionController Class Reference

`#include <motion_controller.h>`

Inheritance diagram for MotionController:



Classes

- struct m_profile_cfg_t

Public Member Functions

- MotionController (m_profile_cfg_t &config)

  Construct a new Motion Controller object.
- void init (double start_pt, double end_pt) override

  Initialize the motion profile for a new movement This will also reset the PID and profile timers.
- double update (double sensor_val) override

  Update the motion profile with a new sensor value.
- double get () override
- void set_limits (double lower, double upper) override
- bool is_on_target () override
- motion_t get_motion () const
- screen::Page * **Page** ()

Static Public Member Functions

- static FeedForward::ff_config_t tune_feedforward (TankDrive &drive, OdometryTank &odometry, double pct=0.6, double duration=2)

Friends

- class **MotionControllerPage**

5.49.1 Detailed Description

Motion Controller class

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

This takes the constants kS, kV, kA, kP, kI, kD, max_v and acceleration and wraps around a feedforward, PID and trapezoid profile. It does so with the following formula:

out = feedfoward.calculate(motion_profile.get(time_s)) + pid.get(motion_profile.get(time_s))

For PID and Feedforward specific formulae, see pid.h, feedforward.h, and trapezoid_profile.h

Author

Ryan McGee

Date

7/13/2022

5.49.2 Constructor & Destructor Documentation

5.49.2.1 MotionController()

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

Construct a new Motion Controller object.

Parameters

config	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

5.49.3 Member Function Documentation

5.49.3.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.49.3.2 get_motion()

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

Returns

The current postion, velocity and acceleration setpoints

5.49.3.3 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

start_pt	Movement starting position
end_pt	Movement ending posiiton

Implements Feedback.

5.49.3.4 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.

5.49.3.5 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

lower	upper limit
upper	lower limiet

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

Parameters

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.49.3.6 tune_feedforward()

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

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

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

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

Parameters

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

Returns

A tuned feedforward object

5.49.3.7 update()

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

Update the motion profile with a new sensor value.

Parameters

sensor_val	Value from the sensor

Returns

the motor input generated from the motion profile

Implements Feedback.

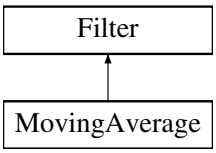
The documentation for this class was generated from the following files:

- include/utils/controls/motion_controller.h
- src/utils/controls/motion_controller.cpp

5.50 MovingAverage Class Reference

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

Inheritance diagram for MovingAverage:



Public Member Functions

- MovingAverage (int buffer_size)
- MovingAverage (int buffer_size, double starting_value)
- void add_entry (double n) override
- double get_value () const override
- int get_size () const

5.50.1 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).

5.50.2 Constructor & Destructor Documentation

5.50.2.1 MovingAverage() [1/2]

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

Create a moving average calculator with 0 as the default value

Parameters

buffer_size	The size of the buffer. The number of samples that constitute a valid reading

5.50.2.2 MovingAverage() [2/2]

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

Create a moving average calculator with a specified default value

Parameters

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

5.50.3 Member Function Documentation

5.50.3.1 add_entry()

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

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

Parameters

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

Implements Filter.

5.50.3.2 get_size()

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

How many samples the average is made from

Returns

the number of samples used to calculate this average

5.50.3.3 get_value()

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

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

Returns

the calculated average. sum(samples)/numsamples

How many samples the average is made from

Returns

the number of samples used to calculate this average

Implements Filter.

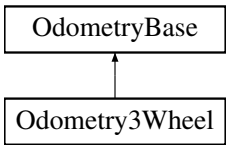
The documentation for this class was generated from the following files:

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

5.51 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)
- AutoCommand * **SetPositionCmd** (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

5.51.1 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

5.51.2 Constructor & Destructor Documentation

5.51.2.1 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

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

5.51.3 Member Function Documentation

5.51.3.1 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

con	Controller reference, for screen and button control
drive	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

5.51.3.2 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

5.52 Odometry3Wheel::odometry3wheel_cfg_t Struct Reference

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

Public Attributes

- double wheelbase_dist
- double off_axis_center_dist
- double wheel_diam

5.52.1 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

5.52.2 Member Data Documentation

5.52.2.1 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

5.52.2.2 wheel_diam

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

the diameter of the tracking wheel

5.52.2.3 wheelbase_dist

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

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

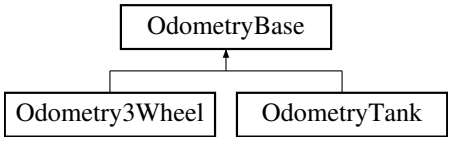
The documentation for this struct was generated from the following file:

- include/subsystems/odometry/odometry_3wheel.h

5.53 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)
- AutoCommand * **SetPositionCmd** (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

5.53.1 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

5.53.2 Constructor & Destructor Documentation

5.53.2.1 OdometryBase()

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

Construct a new Odometry Base object

Parameters

is_async	True to run constantly in the background, false to call update() manually

5.53.3 Member Function Documentation

5.53.3.1 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

ptr	Pointer to OdometryBase object

Returns

Required integer return code. Unused.

5.53.3.2 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.

5.53.3.3 get_accel()

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

Get the current acceleration

Returns

the acceleration rate of the robot (inch/s^2)

5.53.3.4 get_angular_accel_deg()

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

Get the current angular acceleration in degrees

Returns

the angular acceleration at which we are turning (deg/s^2)

5.53.3.5 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)

5.53.3.6 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

5.53.3.7 get_speed()

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

Get the current speed

Returns

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

5.53.3.8 pos_diff()

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

Get the distance between two points

Parameters

start_pos	distance from this point
end_pos	to this point

Returns

the euclidean distance between start_pos and end_pos

5.53.3.9 rot_diff()

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

Get the change in rotation between two points

Parameters

pos1	position with initial rotation
pos2	position with final rotation

Returns

change in rotation between pos1 and pos2

Get the change in rotation between two points

5.53.3.10 set_position()

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

Sets the current position of the robot

Parameters

newpos	the new position that the odometry will believe it is at

Sets the current position of the robot

Reimplemented in OdometryTank.

5.53.3.11 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

start_deg	intitial angle (degrees)
end_deg	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.

5.53.3.12 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.

5.53.4 Member Data Documentation

5.53.4.1 accel

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

the rate at which we are accelerating (inch/s^2)

5.53.4.2 ang_accel_deg

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

the rate at which we are accelerating our turn (deg/s^2)

5.53.4.3 ang_speed_deg

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

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

5.53.4.4 current_pos

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

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

5.53.4.5 handle

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

handle to the vex task that is running the odometry code

5.53.4.6 mut

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

Mutex to control multithreading

5.53.4.7 speed

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

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

5.53.4.8 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:

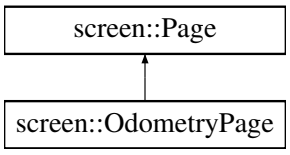
- include/subsystems/odometry/odometry_base.h
- src/subsystems/odometry/odometry_base.cpp

5.54 screen::OdometryPage Class Reference

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

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

Inheritance diagram for screen::OdometryPage:



Public Member Functions

- OdometryPage (OdometryBase &odom, double robot_width, double robot_height, bool do_trail)
  Create an odometry trail. Make sure odometry is initilized before now.
- void update (bool was_pressed, int x, int y) override
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

5.54.1 Detailed Description

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

5.54.2 Constructor & Destructor Documentation

5.54.2.1 OdometryPage()

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

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

Parameters

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

5.54.3 Member Function Documentation

5.54.3.1 draw()

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

See also

Page::draw

Reimplemented from screen::Page.

5.54.3.2 update()

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

See also

Page::update

Reimplemented from screen::Page.

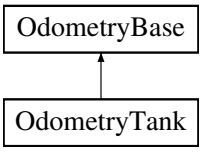
The documentation for this class was generated from the following files:

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

5.55 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)
- AutoCommand ∗ **SetPositionCmd** (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

5.55.1 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.

5.55.2 Constructor & Destructor Documentation

5.55.2.1 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

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

5.55.2.2 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

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

5.55.2.3 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

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

5.55.3 Member Function Documentation

5.55.3.1 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.

5.55.3.2 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.

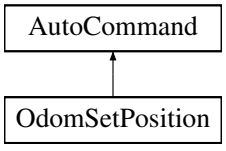
The documentation for this class was generated from the following files:

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

5.56 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.56.1 Detailed Description

AutoCommand wrapper class for the set_position function in the Odometry class

5.56.2 Constructor & Destructor Documentation

5.56.2.1 OdomSetPosition()

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

constructs a new OdomSetPosition command

Parameters

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

Construct an Odometry set pos

Parameters

odom	the odometry system we are setting
newpos	the now position to set the odometry to

5.56.3 Member Function Documentation

5.56.3.1 run()

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

Run set_position Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

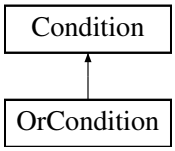
Reimplemented from AutoCommand.

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

5.57 OrCondition Class Reference

Inheritance diagram for OrCondition:



Public Member Functions

- **OrCondition** (Condition *A, Condition *B)
- bool test () override

Public Member Functions inherited from Condition

- Condition * **Or** (Condition *b)
- Condition * **And** (Condition *b)

5.57.1 Member Function Documentation

5.57.1.1 test()

```
bool OrCondition::test ( )  [inline], [override], [virtual]
```

Implements Condition.

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

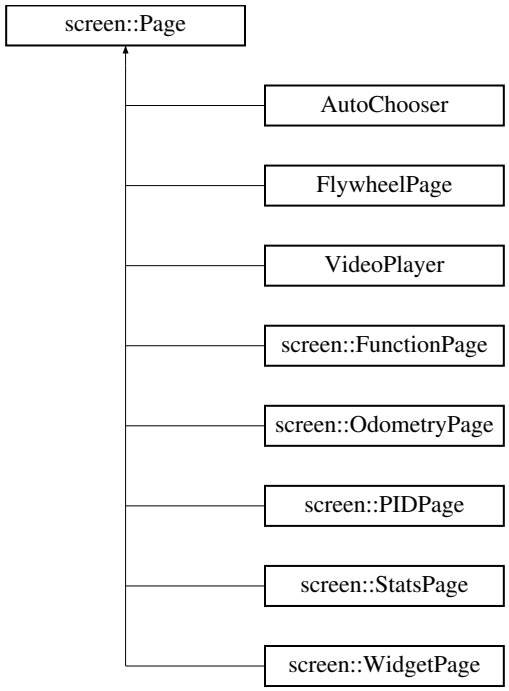
- src/utils/command_structure/auto_command.cpp

5.58 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)

5.58.1 Detailed Description

Page describes one part of the screen slideshow.

5.58.2 Member Function Documentation

5.58.2.1 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

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

Reimplemented in AutoChooser, screen::WidgetPage, screen::StatsPage, screen::OdometryPage, screen::FunctionPage, screen::PIDPage, VideoPlayer, and FlywheelPage.

5.58.2.2 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

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

Reimplemented in AutoChooser, VideoPlayer, screen::WidgetPage, screen::StatsPage, screen::OdometryPage, screen::FunctionPage, screen::PIDPage, and FlywheelPage.

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

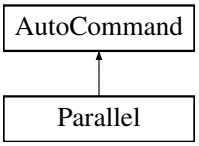
- include/subsystems/screen.h

5.59 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.59.1 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.

5.59.2 Member Function Documentation

5.59.2.1 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.

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

5.60 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/utils/command_structure/auto_command.cpp

5.61 PurePursuit::Path Class Reference

`#include <pure_pursuit.h>`

Public Member Functions

- Path (std::vector< point_t > points, double radius)
- std::vector< point_t > get_points ()
- double get_radius ()
- bool is_valid ()

5.61.1 Detailed Description

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

5.61.2 Constructor & Destructor Documentation

5.61.2.1 Path()

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

Create a Path

Parameters

points	the points that make up the path
radius	the lookahead radius for pure pursuit

5.61.3 Member Function Documentation

5.61.3.1 get_points()

```
std::vector< point_t > PurePursuit::Path::get_points ( )
```

Get the points associated with this Path

5.61.3.2 get_radius()

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

Get the radius associated with this Path

5.61.3.3 is_valid()

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

Get whether this path will behave as expected

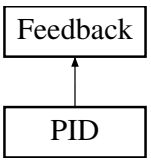
The documentation for this class was generated from the following files:

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

5.62 PID Class Reference

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

Inheritance diagram for PID:



Classes

- struct pid_config_t

Public Types

- enum ERROR_TYPE { **LINEAR** , **ANGULAR** }

Public Member Functions

- PID (pid_config_t &config)
- void init (double start_pt, double set_pt) override
- double update (double sensor_val) override
- double update (double sensor_val, double v_setpt)
- double get_sensor_val () const

  gets the sensor value that we were last updated with
- double get () override
- void set_limits (double lower, double upper) override
- bool is_on_target () override
- void reset ()
- double get_error ()
- double get_target () const
- void set_target (double target)

Public Attributes

- pid_config_t & config

5.62.1 Detailed Description

PID Class

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

out = kP∗error + kI∗integral(d Error) + kD∗(dError/dt)

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

Author

Ryan McGee

Date

4/3/2020

5.62.2 Member Enumeration Documentation

5.62.2.1 ERROR_TYPE

```
enum PID::ERROR_TYPE
```

An enum to distinguish between a linear and angular caluclation of PID error.

5.62.3 Constructor & Destructor Documentation

5.62.3.1 PID()

```
PID::PID (
            pid_config_t & config )
```

Create the PID object

Parameters

config	the configuration data for this controller

Create the PID object

5.62.4 Member Function Documentation

5.62.4.1 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.

5.62.4.2 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

5.62.4.3 get_sensor_val()

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

gets the sensor value that we were last updated with

Returns

sensor_val

5.62.4.4 get_target()

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

Get the PID's target

Returns

the target the PID controller is trying to achieve

5.62.4.5 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

start_pt	commpletely ignored for PID. necessary to satisfy Feedback base
set_pt	sets the target of the PID controller
start_vel	completely ignored for PID. necessary to satisfy Feedback base
end_vel	sets the target end velocity of the PID controller

Implements Feedback.

5.62.4.6 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.

5.62.4.7 reset()

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

Reset the PID loop by resetting time since 0 and accumulated error.

5.62.4.8 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

lower	the lower limit. the PID controller will never command the output go below `lower`
upper	the upper limit. the PID controller will never command the output go higher than `upper`

Set the limits on the PID out. The PID out will "clip" itself to be between the limits.

Implements Feedback.

5.62.4.9 set_target()

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

Set the target for the PID loop, where the robot is trying to end up

Parameters

target	the sensor reading we would like to achieve

Set the target for the PID loop, where the robot is trying to end up

5.62.4.10 update() [1/2]

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

Update the PID loop by taking the time difference from last update, and running the PID formula with the new sensor data

Parameters

sensor_val	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.

5.62.4.11 update() [2/2]

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

Update the PID loop by taking the time difference from last update, and running the PID formula with the new sensor data

Parameters

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

Returns

the new output. What would be returned by PID::get()

5.62.5 Member Data Documentation

5.62.5.1 config

```
pid_config_t& PID::config
```

configuration struct for this controller. see pid_config_t for information about what this contains

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

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

5.63 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**

  derivitave coeffecient d ∗ derivative(error)
- double **deadband**

  at what threshold are we close enough to be finished
- double on_target_time
- ERROR_TYPE error_method

5.63.1 Detailed Description

pid_config_t holds the configuration parameters for a pid controller In addtion 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

5.63.2 Member Data Documentation

5.63.2.1 error_method

```
ERROR_TYPE PID::pid_config_t::error_method
```

Linear or angular. wheter to do error as a simple subtraction or to wrap

5.63.2.2 on_target_time

```
double PID::pid_config_t::on_target_time
```

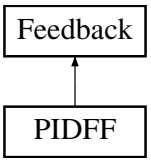
the time in seconds that we have to be on target for to say we are officially at the target

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

- include/utils/controls/pid.h

5.64 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 **get_target** () const
- double **get_sensor_val** () const
- 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
- void **reset** ()

Public Attributes

- PID **pid**

5.64.1 Member Function Documentation

5.64.1.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.64.1.2 init()

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

Initialize the feedback controller for a movement

Parameters

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	the current rate of change of the sensor value
end_vel	the desired ending rate of change of the sensor value

Initialize the feedback controller for a movement

Parameters

start_pt	the current sensor value
set_pt	where the sensor value should be

Implements Feedback.

5.64.1.3 is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implements Feedback.

5.64.1.4 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

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.64.1.5 set_target()

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

Set the target of the PID loop

Parameters

set↩ _pt	Setpoint / target value

5.64.1.6 update() [1/2]

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

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

Parameters

val	value from the sensor

Returns

feedback loop result

Implements Feedback.

5.64.1.7 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

val	value from the sensor
vel_setpt	Velocity for feedforward
a_setpt	Acceleration for feedfoward

Returns

feedback loop result

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

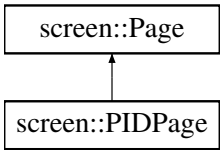
- include/utils/controls/pidff.h
- src/utils/controls/pidff.cpp

5.65 screen::PIDPage Class Reference

PIDPage provides a way to tune a pid controller on the screen.

`#include <screen.h>`

Inheritance diagram for screen::PIDPage:



Public Member Functions

- PIDPage (PID &pid, std::function< void(void)> onchange=[]() {})
  Create a PIDPage.
- **PIDPage** (PIDFF &pidff, std::function< void(void)> onchange=[]() {})
- void update (bool was_pressed, int x, int y) override
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

5.65.1 Detailed Description

PIDPage provides a way to tune a pid controller on the screen.

5.65.2 Constructor & Destructor Documentation

5.65.2.1 PIDPage()

```
screen::PIDPage::PIDPage (
            PID & pid,
            std::string name,
            std::function< void(void)> onchange = [](){} )
```

Create a PIDPage.

Parameters

pid	the pid controller we're changing
name	a name to recognize this pid controller if we've got multiple pid screens
onchange	a function that is called when a tuning parameter is changed. If you need to update stuff on that change register a handler here

5.65.3 Member Function Documentation

5.65.3.1 draw()

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

See also

Page::draw

Reimplemented from screen::Page.

5.65.3.2 update()

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

See also

Page::update

Reimplemented from screen::Page.

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

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

5.66 plm_frame_t Struct Reference

Public Attributes

- double **time**
- unsigned int **width**
- unsigned int **height**
- plm_plane_t **y**
- plm_plane_t **cr**
- plm_plane_t **cb**

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

- include/subsystems/fun/pl_mpeg.h

5.67 plm_packet_t Struct Reference

Public Attributes

- int **type**
- double **pts**
- size_t **length**
- uint8_t * **data**

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

- include/subsystems/fun/pl_mpeg.h

5.68 plm_plane_t Struct Reference

Public Attributes

- unsigned int **width**
- unsigned int **height**
- uint8_t * **data**

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

- include/subsystems/fun/pl_mpeg.h

5.69 plm_samples_t Struct Reference

Public Attributes

- double **time**
- unsigned int **count**
- float **interleaved** [PLM_AUDIO_SAMPLES_PER_FRAME *2]

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

- include/subsystems/fun/pl_mpeg.h

5.70 point_t Struct Reference

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

Public Member Functions

- double dist (const point_t other) const
- point_t operator+ (const point_t &other) const
- point_t operator- (const point_t &other) const
- 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

5.70.1 Detailed Description

Data structure representing an X,Y coordinate

5.70.2 Member Function Documentation

5.70.2.1 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

other	the point to measure the distance from

Returns

the euclidian distance between this and other

5.70.2.2 operator+()

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

Vector2D addition operation on points

Parameters

other	the point to add on to this

Returns

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

5.70.2.3 operator-()

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

Vector2D subtraction operation on points

Parameters

other	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

5.71 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

5.71.1 Detailed Description

Describes a single position and rotation

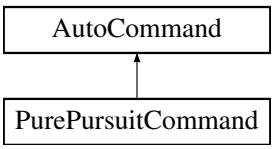
The documentation for this struct was generated from the following file:

- include/utils/geometry.h

5.72 PurePursuitCommand Class Reference

`#include <drive_commands.h>`

Inheritance diagram for PurePursuitCommand:



Public Member Functions

- PurePursuitCommand (TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0)
- bool run () override
- void on_timeout () override

Public Member Functions inherited from AutoCommand

- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.72.1 Detailed Description

Autocommand wrapper class for pure pursuit function in the TankDrive class

5.72.2 Constructor & Destructor Documentation

5.72.2.1 PurePursuitCommand()

```
PurePursuitCommand::PurePursuitCommand (
            TankDrive & drive_sys,
            Feedback & feedback,
            PurePursuit::Path path,
            directionType dir,
            double max_speed = 1,
            double end_speed = 0 )
```

Construct a Pure Pursuit AutoCommand

Parameters

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
feedback	The feedback controller determining speed
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)

5.72.3 Member Function Documentation

5.72.3.1 on_timeout()

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

Reset the drive system when it times out

Reimplemented from AutoCommand.

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

5.73 Rect Struct Reference

Public Member Functions

- point_t **dimensions** () const
- point_t **center** () const
- double **width** () const
- double **height** () const
- bool **contains** (point_t p) const

Static Public Member Functions

- static Rect **from_min_and_size** (point_t min, point_t size)

Public Attributes

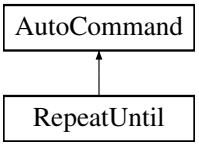
- point_t **min**
- point_t **max**

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

- include/utils/geometry.h

5.74 RepeatUntil Class Reference

Inheritance diagram for RepeatUntil:



Public Member Functions

- RepeatUntil (InOrder cmds, size_t repeats)

  RepeatUntil that runs a fixed number of times.
- RepeatUntil (InOrder cmds, Condition *true_to_end)

  RepeatUntil the condition.
- bool run () override
- void on_timeout () override

Public Member Functions inherited from AutoCommand

- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.74.1 Constructor & Destructor Documentation

5.74.1.1 RepeatUntil() [1/2]

```
RepeatUntil::RepeatUntil (
            InOrder cmds,
            size_t repeats )
```

RepeatUntil that runs a fixed number of times.

Parameters

cmds	the cmds to repeat
repeats	the number of repeats to do

5.74.1.2 RepeatUntil() [2/2]

```
RepeatUntil::RepeatUntil (
            InOrder cmds,
            Condition * true_to_end )
```

RepeatUntil the condition.

Parameters

cmds	the cmds to run
true_to_end	we will repeat until true_or_end.test() returns true

5.74.2 Member Function Documentation

5.74.2.1 on_timeout()

```
void RepeatUntil::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.

5.74.2.2 run()

```
bool RepeatUntil::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

5.75 robot_specs_t Struct Reference

`#include <robot_specs.h>`

Public Attributes

- double **robot_radius**

  if you were to draw a circle with this radius, the robot would be entirely contained within it
- double **odom_wheel_diam**

  the diameter of the wheels used for
- double **odom_gear_ratio**

  the ratio of the odometry wheel to the encoder reading odometry data
- double **dist_between_wheels**

  the distance between centers of the central drive wheels
- double drive_correction_cutoff
- Feedback * **drive_feedback**

  the default feedback for autonomous driving
- Feedback * **turn_feedback**

  the defualt 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

5.75.1 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.

5.75.2 Member Data Documentation

5.75.2.1 drive_correction_cutoff

`double robot_specs_t::drive_correction_cutoff`

the distance at which to stop trying to turn towards the target. If we are less than this value, we can continue driving forward to minimize our distance but will not try to spin around to point directly at the target

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

- include/robot_specs.h

5.76 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**

5.76.1 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

5.77 screen::ScreenRect Struct Reference

Public Attributes

- uint32_t **x1**
- uint32_t **y1**
- uint32_t **x2**
- uint32_t **y2**

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

- include/subsystems/screen.h

5.78 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

5.78.1 Detailed Description

Serializes Arbitrary data to a file on the SD Card.

5.78.2 Constructor & Destructor Documentation

5.78.2.1 Serializer()

```
Serializer::Serializer (
            const std::string & filename,
            bool flush_always = true )  [inline], [explicit]
```

create a Serializer

Parameters

filename	the file to read from. If filename does not exist we will create that file
flush_always	If true, after every write flush to a file. If false, you are responsible for calling save_to_disk

5.78.3 Member Function Documentation

5.78.3.1 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

name	name of value
otherwise	value if the name is not specified

Returns

the value if found or otherwise

5.78.3.2 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

name	name of value
otherwise	value if the name is not specified

Returns

the value if found or otherwise

5.78.3.3 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

name	name of value
otherwise	value if the name is not specified

Returns

the value if found or otherwise

5.78.3.4 save_to_disk()

```
void Serializer::save_to_disk ( ) const
```

saves current Serializer state to disk

forms data bytes then saves to filename this was openned with

5.78.3.5 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

name	name of bool
b	value of bool

5.78.3.6 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

name	name of double
d	value of double

5.78.3.7 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

name	name of integer
i	value of integer

5.78.3.8 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

name	name of string
i	value of string

5.78.3.9 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

name	name of value
otherwise	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

5.79 screen::SizedWidget Struct Reference

Public Attributes

- int **size**
- WidgetConfig & **widget**

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

- include/subsystems/screen.h

5.80 SliderCfg Struct Reference

Public Attributes

- double & **val**
- double **min**
- double **max**

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

- include/subsystems/layout.h

5.81 screen::SliderConfig Struct Reference

Public Attributes

- double & **val**
- double **low**
- double **high**

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

- include/subsystems/screen.h

5.82 screen::SliderWidget Class Reference

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

`#include <screen.h>`

Public Member Functions

- SliderWidget (double &val, double low, double high, Rect rect, std::string name)

  Creates a slider widget.
- bool update (bool was_pressed, int x, int y)

  responds to user input
- void **draw** (vex::brain::lcd &, bool first_draw, unsigned int frame_number)

  Page::draws the slide to the screen

5.82.1 Detailed Description

Widget that updates a double value. Updates by reference so watch out for race conditions cuz the screen stuff lives on another thread.

5.82.2 Constructor & Destructor Documentation

5.82.2.1 SliderWidget()

```
screen::SliderWidget::SliderWidget (
            double & val,
            double low,
            double high,
            Rect rect,
            std::string name )  [inline]
```

Creates a slider widget.

Parameters

val	reference to the value to modify
low	minimum value to go to
high	maximum value to go to
rect	rect to draw it
name	name of the value

5.82.3 Member Function Documentation

5.82.3.1 update()

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

responds to user input

Parameters

was_pressed	if the screen is pressed
x	x position if the screen was pressed
y	y position if the screen was pressed

Returns

true if the value updated

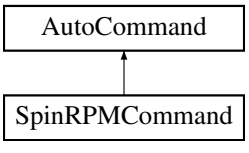
The documentation for this class was generated from the following files:

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

5.83 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.83.1 Detailed Description

File: flywheel_commands.h Desc: [insert meaningful desc] AutoCommand wrapper class for the spin_rpm function in the Flywheel class

5.83.2 Constructor & Destructor Documentation

5.83.2.1 SpinRPMCommand()

```
SpinRPMCommand::SpinRPMCommand (
            Flywheel & flywheel,
            int rpm )
```

Construct a SpinRPM Command

Parameters

flywheel	the flywheel sys to command
rpm	the rpm that we should spin at

File: flywheel_commands.cpp Desc: [insert meaningful desc]

5.83.3 Member Function Documentation

5.83.3.1 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

5.84 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**

5.84.1 Detailed Description

Represents a piece of a cubic spline with s(x) = a(x-xi)^3 + b(x-xi)^2 + c(x-xi) + 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

5.85 StateMachine< System, IDType, Message, delay_ms, do_log >::State Struct Reference

`#include <state_machine.h>`

Public Member Functions

- virtual void **entry** (System &)
- virtual MaybeMessage **work** (System &)
- virtual void **exit** (System &)
- virtual State * **respond** (System &s, Message m)=0
- virtual IDType **id** () const =0

5.85.1 Detailed Description

template<typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
struct StateMachine< System, IDType, Message, delay_ms, do_log >::State

Abstract class that all states for this machine must inherit from States MUST override respond() and id() in order to function correctly (the compiler won't have it any other way)

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

- include/utils/state_machine.h

5.86 StateMachine< System, IDType, Message, delay_ms, do_log > Class Template Reference

State Machine :))))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message.

`#include <state_machine.h>`

Classes

- class MaybeMessage

  MaybeMessage a message of Message type or nothing MaybeMessage m = {}; // empty MaybeMessage m = Message::EnumField1.
- struct State

Public Types

- using **thread_data** = std::pair<State *, StateMachine *>

Public Member Functions

- StateMachine (State *initial)

  Construct a state machine and immediatly start running it.
- IDType current_state () const

  retrieve the current state of the state machine. This is safe to call from external threads
- void send_message (Message msg)

  send a message to the state machine from outside

5.86.1 Detailed Description

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

State Machine :))))))) A fun fun way of controlling stateful subsystems - used in the 2023-2024 Over Under game for our overly complex intake-cata subsystem (see there for an example) The statemachine runs in a background thread and a user thread can interact with it through current_state and send_message.

Designwise: the System class should hold onto any motors, feedback controllers, etc that are persistent in the system States themselves should hold any data that *only* that state needs. For example if a state should be exitted after a certain amount of time, it should hold a timer rather than the System holding that timer. (see Junder from 2024 for an example of this design)

Template Parameters

System	The system that this is the base class of `class Thing : public StateMachine<Thing> @tparam IDType The ID enum that recognizes states. Hint hint, use an`enum class`
Message	the message enum that a state or an outside can send and that states respond to
delay_ms	the delay to wait between each state processing to allow other threads to work
do_log	true if you want print statements describing incoming messages and current states. If true, it is expected that IDType and Message have a function called to_string that takes them as its only parameter and returns a std::string

5.86.2 Constructor & Destructor Documentation

5.86.2.1 StateMachine()

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

Construct a state machine and immediatly start running it.

Parameters

initial	the state that the machine will begin in

5.86.3 Member Function Documentation

5.86.3.1 current_state()

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

retrieve the current state of the state machine. This is safe to call from external threads

Returns

the current state

5.86.3.2 send_message()

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

send a message to the state machine from outside

Parameters

msg	the message to send This is safe to call from external threads

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

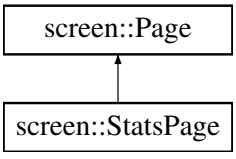
- include/utils/state_machine.h

5.87 screen::StatsPage Class Reference

Draws motor stats and battery stats to the screen.

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

Inheritance diagram for screen::StatsPage:



Public Member Functions

- StatsPage (std::map< std::string, vex::motor & > motors)

  Creates a stats page.
- void update (bool was_pressed, int x, int y) override
- void draw (vex::brain::lcd &, bool first_draw, unsigned int frame_number) override

5.87.1 Detailed Description

Draws motor stats and battery stats to the screen.

5.87.2 Constructor & Destructor Documentation

5.87.2.1 StatsPage()

```
screen::StatsPage::StatsPage (
            std::map< std::string, vex::motor & > motors )
```

Creates a stats page.

Parameters

motors	a map of string to motor that we want to draw on this page

5.87.3 Member Function Documentation

5.87.3.1 draw()

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

See also

Page::draw

Reimplemented from screen::Page.

5.87.3.2 update()

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

See also

Page::update

Reimplemented from screen::Page.

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

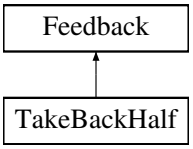
- include/subsystems/screen.h
- src/subsystems/screen.cpp

5.88 TakeBackHalf Class Reference

A velocity controller.

`#include <take_back_half.h>`

Inheritance diagram for TakeBackHalf:



Public Member Functions

- **TakeBackHalf** (double TBH_gain, double first_cross_split, double on_target_threshold)
- void init (double start_pt, double set_pt)
- double update (double val) override
- double get () override
- void set_limits (double lower, double upper) override
- bool is_on_target () override

Public Attributes

- double **TBH_gain**

  tuned parameter
- double **first_cross_split**

5.88.1 Detailed Description

A velocity controller.

Warning

If you try to use this as a position controller, it will fail.

5.88.2 Member Function Documentation

5.88.2.1 get()

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

Returns

the last saved result from the feedback controller

Implements Feedback.

5.88.2.2 init()

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

Initialize the feedback controller for a movement

Parameters

start_pt	the current sensor value
set_pt	where the sensor value should be
start_vel	Movement starting velocity (IGNORED)
end_vel	Movement ending velocity (IGNORED)

Implements Feedback.

5.88.2.3 is_on_target()

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

Returns

true if the feedback controller has reached it's setpoint

Implements Feedback.

5.88.2.4 set_limits()

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

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

Parameters

lower	Upper limit
upper	Lower limit

Implements Feedback.

5.88.2.5 update()

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

Iterate the feedback loop once with an updated sensor value

Parameters

val	value from the sensor

Returns

feedback loop result

Implements Feedback.

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

- include/utils/controls/take_back_half.h
- src/utils/controls/take_back_half.cpp

5.89 TankDrive Class Reference

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

Public Types

- enum class BrakeType { None , ZeroVelocity , Smart }

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, double end_speed=0.0)
- AutoCommand * **DriveToPointCmd** (Feedback &fb, point_t pt, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0)
- AutoCommand * **DriveForwardCmd** (double dist, vex::directionType dir=vex::forward, double max_↩ speed=1.0, double end_speed=0.0)

- AutoCommand * **DriveForwardCmd** (Feedback &fb, double dist, vex::directionType dir=vex::forward, double max_speed=1.0, double end_speed=0.0)
- AutoCommand * **TurnToHeadingCmd** (double heading, double max_speed=1.0, double end_speed=0.0)
- AutoCommand * **TurnToHeadingCmd** (Feedback &fb, double heading, double max_speed=1.0, double end_speed=0.0)
- AutoCommand * **TurnToPointCmd** (double x, double y, vex::directionType dir=vex::directionType::fwd, double max_speed=1.0, double end_speed=0.0)
- AutoCommand * **TurnDegreesCmd** (double degrees, double max_speed=1.0, double start_speed=0.0)
- AutoCommand * **TurnDegreesCmd** (Feedback &fb, double degrees, double max_speed=1.0, double end↩ _speed=0.0)
- AutoCommand * **PurePursuitCmd** (PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0)
- AutoCommand * **PurePursuitCmd** (Feedback &feedback, PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0)
- Condition * **DriveStalledCondition** (double stall_time)
- AutoCommand * **DriveTankCmd** (double left, double right)
- void stop ()
- void drive_tank (double left, double right, int power=1, BrakeType bt=BrakeType::None)
- void drive_tank_raw (double left, double right)
- void drive_arcade (double forward_back, double left_right, int power=1, BrakeType bt=BrakeType::None)
- bool drive_forward (double inches, directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0)
- bool drive_forward (double inches, directionType dir, double max_speed=1, double end_speed=0)
- bool turn_degrees (double degrees, Feedback &feedback, double max_speed=1, double end_speed=0)
- bool turn_degrees (double degrees, double max_speed=1, double end_speed=0)
- bool drive_to_point (double x, double y, vex::directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0)
- bool drive_to_point (double x, double y, vex::directionType dir, double max_speed=1, double end_speed=0)
- bool turn_to_heading (double heading_deg, Feedback &feedback, double max_speed=1, double end_↩ speed=0)
- bool turn_to_heading (double heading_deg, double max_speed=1, double end_speed=0)
- void reset_auto ()
- bool pure_pursuit (PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed=1, double end_speed=0)
- bool pure_pursuit (PurePursuit::Path path, directionType dir, double max_speed=1, double end_speed=0)

Static Public Member Functions

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

5.89.1 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

5.89.2 Member Enumeration Documentation

5.89.2.1 BrakeType

```
enum class TankDrive::BrakeType  [strong]
```

Enumerator

None	just send 0 volts to the motors
ZeroVelocity	try to bring the robot to rest. But don't try to hold position
Smart	bring the robot to rest and once it's stopped, try to hold that position

5.89.3 Constructor & Destructor Documentation

5.89.3.1 TankDrive()

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

Create the TankDrive object

Parameters

left_motors	left side drive motors
right_motors	right side drive motors
config	the configuration specification defining physical dimensions about the robot. See robot_specs_t for more info
odom	an odometry system to track position and rotation. this is necessary to execute autonomous paths

5.89.4 Member Function Documentation

5.89.4.1 drive_arcade()

```
void TankDrive::drive_arcade (
            double forward_back,
            double left_right,
            int power = 1,
            BrakeType bt = BrakeType::None )
```

Drive the robot using arcade style controls. forward_back controls the linear motion, left_right controls the turning.

forward_back and left_right are in "percent": -1.0 -> 1.0

Parameters

forward_back	the percent to move forward or backward
left_right	the percent to turn left or right
power	modifies the input velocities left^power, right^power
bt	breaktype. What to do if the driver lets go of the sticks

Drive the robot using arcade style controls. forward_back controls the linear motion, left_right controls the turning.

left_motors and right_motors are in "percent": -1.0 -> 1.0

5.89.4.2 drive_forward() [1/2]

```
bool TankDrive::drive_forward (
            double inches,
            directionType dir,
            double max_speed = 1,
            double end_speed = 0 )
```

Autonomously drive the robot forward a certain distance

Parameters

inches	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Autonomously drive the robot forward a certain distance

Parameters

inches	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have finished driving to our point

5.89.4.3 drive_forward() [2/2]

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

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

Parameters

inches	the distance to drive forward

Parameters

dir	the direction we want to travel forward and backward
feedback	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true when we have reached our target distance

Use odometry to drive forward a certain distance using a custom feedback controller

Returns whether or not the robot has reached it's destination.

Parameters

inches	the distance to drive forward
dir	the direction we want to travel forward and backward
feedback	the custom feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

5.89.4.4 drive_tank()

```
void TankDrive::drive_tank (
            double left,
            double right,
            int power = 1,
            BrakeType bt = BrakeType::None )
```

Drive the robot using differential style controls. left_motors controls the left motors, right_motors controls the right motors.

left_motors and right_motors are in "percent": -1.0 -> 1.0

Parameters

left	the percent to run the left motors
right	the percent to run the right motors
power	modifies the input velocities left^power, right^power
bt	breaktype. What to do if the driver lets go of the sticks

5.89.4.5 drive_tank_raw()

```
void TankDrive::drive_tank_raw (
```

```
            double left,
            double right )
```

Drive the robot raw-ly

Parameters

left	the percent to run the left motors (-1, 1)
right	the percent to run the right motors (-1, 1)

5.89.4.6 drive_to_point() [1/2]

```
bool TankDrive::drive_to_point (
            double x,
            double y,
            vex::directionType dir,
            double max_speed = 1,
            double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive_sys

Returns whether or not the robot has reached it's destination.

Parameters

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot. Here we use the default feedback controller from the drive_sys

Returns whether or not the robot has reached it's destination.

Parameters

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target point

5.89.4.7 drive_to_point() [2/2]

```
bool TankDrive::drive_to_point (
            double x,
            double y,
            vex::directionType dir,
            Feedback & feedback,
            double max_speed = 1,
            double end_speed = 0 )
```

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

Parameters

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Use odometry to automatically drive the robot to a point on the field. X and Y is the final point we want the robot.

Returns whether or not the robot has reached it's destination.

Parameters

x	the x position of the target
y	the y position of the target
dir	the direction we want to travel forward and backward
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target point

5.89.4.8 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

input	the input before modification
power	the power to raise input to

Returns

input ^ 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

input	the input signal -1 -> 1
power	the power to raise the signal to

Returns

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

5.89.4.9 pure_pursuit() [1/2]

```
bool TankDrive::pure_pursuit (
            PurePursuit::Path path,
            directionType dir,
            double max_speed = 1,
            double end_speed = 0 )
```

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

Use the default drive feedback

Parameters

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

True when the path is complete

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

Use the default drive feedback

Parameters

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.89.4.10 pure_pursuit() [2/2]

```
bool TankDrive::pure_pursuit (
            PurePursuit::Path path,
            directionType dir,
            Feedback & feedback,
            double max_speed = 1,
            double end_speed = 0 )
```

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

Parameters

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
feedback	The feedback controller determining speed
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

True when the path is complete

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

Parameters

path	The list of coordinates to follow, in order
dir	Run the bot forwards or backwards
feedback	The feedback controller determining speed
max_speed	Limit the speed of the robot (for pid / pidff feedbacks)

Returns

True when the path is complete

5.89.4.11 reset_auto()

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

Reset the initialization for autonomous drive functions

5.89.4.12 stop()

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

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

5.89.4.13 turn_degrees() [1/2]

```
bool TankDrive::turn_degrees (
            double degrees,
            double max_speed = 1,
            double end_speed = 0 )
```

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses the defualt turning feedback of the drive system.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Autonomously turn the robot X degrees to counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses the defualt turning feedback of the drive system.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we turned te target number of degrees

5.89.4.14 turn_degrees() [2/2]

```
bool TankDrive::turn_degrees (
            double degrees,
```

```
            Feedback & feedback,
            double max_speed = 1,
            double end_speed = 0 )
```

Autonomously turn the robot X degrees counterclockwise (negative for clockwise), with a maximum motor speed of percent_speed (-1.0 -> 1.0)

Uses PID + Feedforward for it's control.

Parameters

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	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

degrees	degrees by which we will turn relative to the robot (+) turns ccw, (-) turns cw
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have turned our target number of degrees

5.89.4.15 turn_to_heading() [1/2]

```
bool TankDrive::turn_to_heading (
            double heading_deg,
            double max_speed = 1,
            double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

Parameters

heading_deg	the heading to which we will turn
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward. Uses the defualt turn feedback of the drive system

Parameters

heading_deg	the heading to which we will turn
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target heading

5.89.4.16 turn_to_heading() [2/2]

```
bool TankDrive::turn_to_heading (
            double heading_deg,
            Feedback & feedback,
            double max_speed = 1,
            double end_speed = 0 )
```

Turn the robot in place to an exact heading relative to the field. 0 is forward.

Parameters

heading_deg	the heading to which we will turn
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Turn the robot in place to an exact heading relative to the field. 0 is forward.

Parameters

heading_deg	the heading to which we will turn
feedback	the feedback controller we will use to travel. controls the rate at which we accelerate and drive.
max_speed	the maximum percentage of robot speed at which the robot will travel. 1 = full power
end_speed	the movement profile will attempt to reach this velocity by its completion

Returns

true if we have reached our target heading

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

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

5.90 screen::TextConfig Struct Reference

Public Attributes

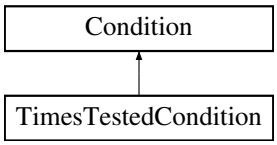
- std::function< std::string()> **text**

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

- include/subsystems/screen.h

5.91 TimesTestedCondition Class Reference

Inheritance diagram for TimesTestedCondition:



Public Member Functions

- **TimesTestedCondition** (size_t N)
- bool test () override

Public Member Functions inherited from Condition

- Condition * **Or** (Condition *b)
- Condition * **And** (Condition *b)

5.91.1 Member Function Documentation

5.91.1.1 test()

```
bool TimesTestedCondition::test ( )  [inline], [override], [virtual]
```

Implements Condition.

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

- include/utils/command_structure/auto_command.h

5.92 TrapezoidProfile Class Reference

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

Public Member Functions

- TrapezoidProfile (double max_v, double accel)

  Construct a new Trapezoid Profile object.
- motion_t calculate (double time_s)

  Run the trapezoidal profile based on the time that's ellapsed.
- void set_endpts (double start, double end)
- void set_accel (double accel)
- void set_max_v (double max_v)
- double get_movement_time ()

5.92.1 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 it's 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 feedfoward 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/rkm3ivu1yk

Author

Ryan McGee

Date

7/12/2022

5.92.2 Constructor & Destructor Documentation

5.92.2.1 TrapezoidProfile()

```
TrapezoidProfile::TrapezoidProfile (
            double max_v,
            double accel )
```

Construct a new Trapezoid Profile object.

Parameters

max_v	Maximum velocity the robot can run at
accel	Maximum acceleration of the robot

5.92.3 Member Function Documentation

5.92.3.1 calculate()

```
motion_t TrapezoidProfile::calculate (
            double time_s )
```

Run the trapezoidal profile based on the time that's ellapsed.

Parameters

time_s	Time since start of movement

Returns

motion_t Position, velocity and acceleration

5.92.3.2 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

5.92.3.3 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

accel	the acceleration amount to use

5.92.3.4 set_endpts()

```
void TrapezoidProfile::set_endpts (
            double start,
            double end )
```

set_endpts defines a start and end position

Parameters

start	the starting position of the path
end	the ending position of the path

5.92.3.5 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

max_v	the maximum velocity the robot can travel at

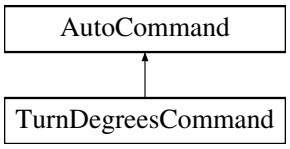
The documentation for this class was generated from the following files:

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

5.93 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, double end_speed=0)
- bool run () override
- void on_timeout () override

Public Member Functions inherited from AutoCommand

- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.93.1 Detailed Description

AutoCommand wrapper class for the turn_degrees function in the TankDrive class

5.93.2 Constructor & Destructor Documentation

5.93.2.1 TurnDegreesCommand()

```
TurnDegreesCommand::TurnDegreesCommand (
            TankDrive & drive_sys,
            Feedback & feedback,
            double degrees,
            double max_speed = 1,
            double end_speed = 0 )
```

Construct a TurnDegreesCommand Command

Parameters

drive_sys	the drive system we are commanding
feedback	the feedback controller we are using to execute the turn
degrees	how many degrees to rotate
max_speed	0 -> 1 percentage of the drive systems speed to drive at

5.93.3 Member Function Documentation

5.93.3.1 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.

5.93.3.2 run()

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

Run turn_degrees Overrides run from AutoCommand

Returns

true when execution is complete, false otherwise

Reimplemented from AutoCommand.

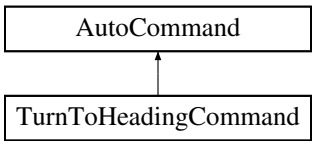
The documentation for this class was generated from the following files:

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

5.94 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, double end_speed=0)
- bool run () override
- void on_timeout () override

Public Member Functions inherited from AutoCommand

- AutoCommand * **withTimeout** (double t_seconds)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.94.1 Detailed Description

AutoCommand wrapper class for the turn_to_heading() function in the TankDrive class

5.94.2 Constructor & Destructor Documentation

5.94.2.1 TurnToHeadingCommand()

```
TurnToHeadingCommand::TurnToHeadingCommand (
            TankDrive & drive_sys,
            Feedback & feedback,
            double heading_deg,
            double max_speed = 1,
            double end_speed = 0 )
```

Construct a TurnToHeadingCommand Command

Parameters

drive_sys	the drive system we are commanding
feedback	the feedback controller we are using to execute the drive
heading_deg	the heading to turn to in degrees
max_speed	0 -> 1 percentage of the drive systems speed to drive at

5.94.3 Member Function Documentation

5.94.3.1 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.

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

5.95 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)

5.95.1 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

5.95.2 Constructor & Destructor Documentation

5.95.2.1 Vector2D() [1/2]

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

Construct a vector object.

Parameters

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

5.95.2.2 Vector2D() [2/2]

```
Vector2D::Vector2D (
            point_t p )
```

Construct a vector object from a cartesian point.

Parameters

p	point_t.x , point_t.y

5.95.3 Member Function Documentation

5.95.3.1 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 vetctor in radians

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

Use r2d() to convert.

5.95.3.2 get_mag()

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

Returns

the magnitude of the vector

Get the magnitude of the vector

5.95.3.3 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.

5.95.3.4 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.

5.95.3.5 normalize()

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

Changes the magnitude of the vector to 1

Returns

the normalized vector

Changes the magnetude of the vector to 1

5.95.3.6 operator∗()

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

Scales a Vector2D by a scalar with the ∗ operator

Parameters

x	the value to scale the vector by

Returns

the this Vector2D scaled by x

5.95.3.7 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

other	the vector to add to this

Returns

the sum of the vectors

5.95.3.8 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

other	the vector to subtract from this

Returns

the difference of the vectors

5.95.3.9 point()

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

Returns a point from the vector

Returns

the point represented by the vector

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

Returns

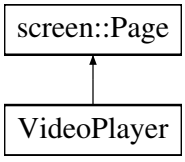
the x, y representation of the vector

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

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

5.96 VideoPlayer Class Reference

Inheritance diagram for VideoPlayer:



Public Member Functions

- 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 &screen, 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)

5.96.1 Member Function Documentation

5.96.1.1 draw()

```
void VideoPlayer::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

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

Reimplemented from screen::Page.

5.96.1.2 update()

```
void VideoPlayer::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

was_pressed	true if the screen has been pressed
x	x position of screen press (if the screen was pressed)
y	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:

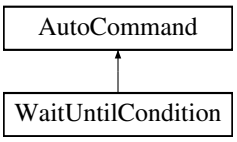
- include/subsystems/fun/video.h
- src/subsystems/fun/video.cpp

5.97 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.97.1 Detailed Description

Waits until the condition is true.

5.97.2 Member Function Documentation

5.97.2.1 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.

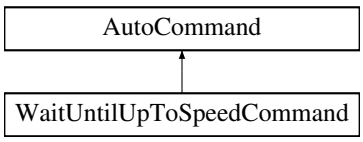
The documentation for this class was generated from the following file:

- include/utils/command_structure/auto_command.h

5.98 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)
- AutoCommand * **withCancelCondition** (Condition *true_to_end)

Additional Inherited Members

Public Attributes inherited from AutoCommand

- double timeout_seconds = default_timeout
- Condition * **true_to_end** = nullptr

Static Public Attributes inherited from AutoCommand

- static constexpr double **default_timeout** = 10.0

5.98.1 Detailed Description

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

5.98.2 Constructor & Destructor Documentation

5.98.2.1 WaitUntilUpToSpeedCommand()

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

Creat a WaitUntilUpToSpeedCommand

Parameters

flywheel	the flywheel system we are commanding
threshold_rpm	the threshold over and under the flywheel target RPM that we define to be acceptable

5.98.3 Member Function Documentation

5.98.3.1 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

5.99 screen::WidgetConfig Struct Reference

Public Types

- enum **Type** {
  Col , **Row** , **Slider** , **Button** ,
  Checkbox , **Label** , **Text** , **Graph** }

Public Attributes

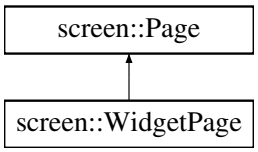
- Type **type**
- union {
  std::vector< SizedWidget > **widgets**
  SliderConfig **slider**
  ButtonConfig **button**
  CheckboxConfig **checkbox**
  LabelConfig **label**
  TextConfig **text**
  GraphDrawer * **graph**
  } **config**

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

- include/subsystems/screen.h

5.100 screen::WidgetPage Class Reference

Inheritance diagram for screen::WidgetPage:



Public Member Functions

- **WidgetPage** (WidgetConfig &cfg)
- 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)

5.100.1 Member Function Documentation

5.100.1.1 draw()

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

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

Parameters

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

Reimplemented from screen::Page.

5.100.1.2 update()

```
void screen::WidgetPage::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

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

Reimplemented from screen::Page.

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

- include/subsystems/screen.h

Chapter 6

File Documentation

6.1 robot_specs.h

```
00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003 #include "../core/include/utils/controls/pid.h"
00004
00011 typedef struct {
00012   double
00013       robot_radius;
00014
00015   double odom_wheel_diam;
00016   double odom_gear_ratio;
00017   double dist_between_wheels;
00018
00019   double drive_correction_cutoff;
00022
00023   Feedback *drive_feedback;
00024   Feedback *turn_feedback;
00025   PID::pid_config_t correction_pid;
00026
00027 } robot_specs_t;
```

6.2 custom_encoder.h

```
00001 #pragma once
00002 #include "vex.h"
00003
00008 class CustomEncoder : public vex::encoder {
00009   typedef vex::encoder super;
00010
00011 public:
00017   CustomEncoder(vex::triport::port &port, double ticks_per_rev);
00018
00024   void setRotation(double val, vex::rotationUnits units);
00025
00031   void setPosition(double val, vex::rotationUnits units);
00032
00038   double rotation(vex::rotationUnits units);
00039
00045   double position(vex::rotationUnits units);
00046
00052   double velocity(vex::velocityUnits units);
00053
00054 private:
00055   double tick_scalar;
00056 };
```

6.3 flywheel.h

```
00001 #pragma once
00002
```

```
00003 #include "../core/include/robot_specs.h"
00004 #include "../core/include/subsystems/screen.h"
00005 #include "../core/include/utils/command_structure/auto_command.h"
00006 #include "../core/include/utils/controls/feedforward.h"
00007 #include "../core/include/utils/controls/pid.h"
00008 #include "vex.h"
00009 #include <atomic>
00010
00018 class Flywheel {
00019
00020 public:
00021   // CONSTRUCTORS, GETTERS, AND SETTERS
00030   Flywheel(vex::motor_group &motors, Feedback &feedback, FeedForward &helper, const double ratio,
      Filter &filt);
00031
00036   double get_target() const;
00037
00041   double getRPM() const;
00042
00046   vex::motor_group &get_motors() const;
00047
00054   void spin_manual(double speed, directionType dir = fwd);
00055
00061   void spin_rpm(double rpm);
00062
00066   void stop();
00067
00072   bool is_on_target() { return fb.is_on_target(); }
00073
00078   screen::Page *Page() const;
00079
00085   AutoCommand *SpinRpmCmd(int rpm) {
00086
00087     return new FunctionCommand([this, rpm]() {
00088       spin_rpm(rpm);
00089       return true;
00090     });
00091   }
00092
00098   AutoCommand *WaitUntilUpToSpeedCmd() {
00099     return new WaitUntilCondition(new FunctionCondition([this]() { return is_on_target(); }));
00100   }
00101
00102 private:
00103   friend class FlywheelPage;
00104   friend int spinRPMTask(void *wheelPointer);
00105
00106   vex::motor_group &motors;
00107   bool task_running = false;
00108   Feedback &fb;
00109   FeedForward &ff;
00110   vex::mutex fb_mut;
00111   double ratio;
00112   std::atomic<double> target_rpm;
00113   task rpm_task;
00114   Filter &avger;
00115
00116   // Functions for internal use only
00121   void set_target(double value);
00125   double measure_RPM();
00126
00133   void spin_raw(double speed, directionType dir = fwd);
00134 };
```

6.4 pl_mpeg.h

```
00001 #include "vex.h"
00002 /*
00003 PL_MPEG - MPEG1 Video decoder, MP2 Audio decoder, MPEG-PS demuxer
00004
00005 Dominic Szablewski - https://phoboslab.org
00006
00007
00008 -- LICENSE: The MIT License(MIT)
00009
00010 Copyright(c) 2019 Dominic Szablewski
00011
00012 Permission is hereby granted, free of charge, to any person obtaining a copy of
00013 this software and associated documentation files(the "Software"), to deal in
00014 the Software without restriction, including without limitation the rights to
00015 use, copy, modify, merge, publish, distribute, sublicense, and / or sell copies
00016 of the Software, and to permit persons to whom the Software is furnished to do
00017 so, subject to the following conditions :
```

```
00018 The above copyright notice and this permission notice shall be included in all
00019 copies or substantial portions of the Software.
00020 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00021 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00022 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.IN NO EVENT SHALL THE
00023 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00024 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00025 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00026 SOFTWARE.
00027
00028
00029
00030
00031 -- Synopsis
00032
00033 // Define `PL_MPEG_IMPLEMENTATION` in *one* C/C++ file before including this
00034 // library to create the implementation.
00035
00036 #define PL_MPEG_IMPLEMENTATION
00037 #include "plmpeg.h"
00038
00039 // This function gets called for each decoded video frame
00040 void my_video_callback(plm_t *plm, plm_frame_t *frame, void *user) {
00041         // Do something with frame->y.data, frame->cr.data, frame->cb.data
00042 }
00043
00044 // This function gets called for each decoded audio frame
00045 void my_audio_callback(plm_t *plm, plm_samples_t *frame, void *user) {
00046         // Do something with samples->interleaved
00047 }
00048
00049 // Load a .mpg (MPEG Program Stream) file
00050 plm_t *plm = plm_create_with_filename("some-file.mpg");
00051
00052 // Install the video & audio decode callbacks
00053 plm_set_video_decode_callback(plm, my_video_callback, my_data);
00054 plm_set_audio_decode_callback(plm, my_audio_callback, my_data);
00055
00056
00057 // Decode
00058 do {
00059         plm_decode(plm, time_since_last_call);
00060 } while (!plm_has_ended(plm));
00061
00062 // All done
00063 plm_destroy(plm);
00064
00065
00066
00067 -- Documentation
00068
00069 This library provides several interfaces to load, demux and decode MPEG video
00070 and audio data. A high-level API combines the demuxer, video & audio decoders
00071 in an easy to use wrapper.
00072
00073 Lower-level APIs for accessing the demuxer, video decoder and audio decoder,
00074 as well as providing different data sources are also available.
00075
00076 Interfaces are written in an object oriented style, meaning you create object
00077 instances via various different constructor functions (plm_*create()),
00078 do some work on them and later dispose them via plm_*destroy().
00079
00080 plm_* ......... the high-level interface, combining demuxer and decoders
00081 plm_buffer_* .. the data source used by all interfaces
00082 plm_demux_* ... the MPEG-PS demuxer
00083 plm_video_* ... the MPEG1 Video ("mpeg1") decoder
00084 plm_audio_* ... the MPEG1 Audio Layer II ("mp2") decoder
00085
00086
00087 With the high-level interface you have two options to decode video & audio:
00088
00089  1. Use plm_decode() and just hand over the delta time since the last call.
00090     It will decode everything needed and call your callbacks (specified through
00091     plm_set_{video|audio}_decode_callback()) any number of times.
00092
00093  2. Use plm_decode_video() and plm_decode_audio() to decode exactly one
00094     frame of video or audio data at a time. How you handle the synchronization
00095     of both streams is up to you.
00096
00097 If you only want to decode video *or* audio through these functions, you should
00098 disable the other stream (plm_set_{video|audio}_enabled(FALSE))
00099
00100 Video data is decoded into a struct with all 3 planes (Y, Cr, Cb) stored in
00101 separate buffers. You can either convert this to RGB on the CPU (slow) via the
00102 plm_frame_to_rgb() function or do it on the GPU with the following matrix:
00103
00104 mat4 bt601 = mat4(
```

```
00105         1.16438,  0.00000,  1.59603, -0.87079,
00106         1.16438, -0.39176, -0.81297,  0.52959,
00107         1.16438,  2.01723,  0.00000, -1.08139,
00108         0, 0, 0, 1
00109 );
00110 gl_FragColor = vec4(y, cb, cr, 1.0) * bt601;
00111
00112 Audio data is decoded into a struct with either one single float array with the
00113 samples for the left and right channel interleaved, or if the
00114 PLM_AUDIO_SEPARATE_CHANNELS is defined *before* including this library, into
00115 two separate float arrays - one for each channel.
00116
00117
00118 Data can be supplied to the high level interface, the demuxer and the decoders
00119 in three different ways:
00120
00121  1. Using plm_create_from_filename() or with a file handle with
00122     plm_create_from_file().
00123
00124  2. Using plm_create_with_memory() and supplying a pointer to memory that
00125     contains the whole file.
00126
00127  3. Using plm_create_with_buffer(), supplying your own plm_buffer_t instance and
00128     periodically writing to this buffer.
00129
00130 When using your own plm_buffer_t instance, you can fill this buffer using
00131 plm_buffer_write(). You can either monitor plm_buffer_get_remaining() and push
00132 data when appropriate, or install a callback on the buffer with
00133 plm_buffer_set_load_callback() that gets called whenever the buffer needs more
00134 data.
00135
00136 A buffer created with plm_buffer_create_with_capacity() is treated as a ring
00137 buffer, meaning that data that has already been read, will be discarded. In
00138 contrast, a buffer created with plm_buffer_create_for_appending() will keep all
00139 data written to it in memory. This enables seeking in the already loaded data.
00140
00141
00142 There should be no need to use the lower level plm_demux_*, plm_video_* and
00143 plm_audio_* functions, if all you want to do is read/decode an MPEG-PS file.
00144 However, if you get raw mpeg1video data or raw mp2 audio data from a different
00145 source, these functions can be used to decode the raw data directly. Similarly,
00146 if you only want to analyze an MPEG-PS file or extract raw video or audio
00147 packets from it, you can use the plm_demux_* functions.
00148
00149
00150 This library uses malloc(), realloc() and free() to manage memory. Typically
00151 all allocation happens up-front when creating the interface. However, the
00152 default buffer size may be too small for certain inputs. In these cases plmpeg
00153 will realloc() the buffer with a larger size whenever needed. You can configure
00154 the default buffer size by defining PLM_BUFFER_DEFAULT_SIZE *before*
00155 including this library.
00156
00157 You can also define PLM_MALLOC, PLM_REALLOC and PLM_FREE to provide your own
00158 memory management functions.
00159
00160
00161 See below for detailed the API documentation.
00162
00163 */
00164
00165 #ifndef PL_MPEG_H
00166 #define PL_MPEG_H
00167
00168 #include <stdint.h>
00169 // #include <stdio.h>
00170
00171 #ifdef __cplusplus
00172 extern "C" {
00173 #endif
00174
00175 // -----------------------------------------------------------------------------
00176 // Public Data Types
00177
00178 // Object types for the various interfaces
00179
00180 typedef struct plm_t plm_t;
00181 typedef struct plm_buffer_t plm_buffer_t;
00182 typedef struct plm_demux_t plm_demux_t;
00183 typedef struct plm_video_t plm_video_t;
00184 typedef struct plm_audio_t plm_audio_t;
00185
00186 // Demuxed MPEG PS packet
00187 // The type maps directly to the various MPEG-PES start codes. PTS is the
00188 // presentation time stamp of the packet in seconds. Note that not all packets
00189 // have a PTS value, indicated by PLM_PACKET_INVALID_TS.
00190
00191 #define PLM_PACKET_INVALID_TS -1
```

```
00192
00193 typedef struct {
00194   int type;
00195   double pts;
00196   size_t length;
00197   uint8_t *data;
00198 } plm_packet_t;
00199
00200 // Decoded Video Plane
00201 // The byte length of the data is width * height. Note that different planes
00202 // have different sizes: the Luma plane (Y) is double the size of each of
00203 // the two Chroma planes (Cr, Cb) - i.e. 4 times the byte length.
00204 // Also note that the size of the plane does *not* denote the size of the
00205 // displayed frame. The sizes of planes are always rounded up to the nearest
00206 // macroblock (16px).
00207
00208 typedef struct {
00209   unsigned int width;
00210   unsigned int height;
00211   uint8_t *data;
00212 } plm_plane_t;
00213
00214 // Decoded Video Frame
00215 // width and height denote the desired display size of the frame. This may be
00216 // different from the internal size of the 3 planes.
00217
00218 typedef struct {
00219   double time;
00220   unsigned int width;
00221   unsigned int height;
00222   plm_plane_t y;
00223   plm_plane_t cr;
00224   plm_plane_t cb;
00225 } plm_frame_t;
00226
00227 // Callback function type for decoded video frames used by the high-level
00228 // plm_* interface
00229
00230 typedef void (*plm_video_decode_callback)(plm_t *self, plm_frame_t *frame, void *user);
00231
00232 // Decoded Audio Samples
00233 // Samples are stored as normalized (-1, 1) float either interleaved, or if
00234 // PLM_AUDIO_SEPARATE_CHANNELS is defined, in two separate arrays.
00235 // The `count` is always PLM_AUDIO_SAMPLES_PER_FRAME and just there for
00236 // convenience.
00237
00238 #define PLM_AUDIO_SAMPLES_PER_FRAME 1152
00239
00240 typedef struct {
00241   double time;
00242   unsigned int count;
00243 #ifdef PLM_AUDIO_SEPARATE_CHANNELS
00244   float left[PLM_AUDIO_SAMPLES_PER_FRAME];
00245   float right[PLM_AUDIO_SAMPLES_PER_FRAME];
00246 #else
00247   float interleaved[PLM_AUDIO_SAMPLES_PER_FRAME * 2];
00248 #endif
00249 } plm_samples_t;
00250
00251 // Callback function type for decoded audio samples used by the high-level
00252 // plm_* interface
00253
00254 typedef void (*plm_audio_decode_callback)(plm_t *self, plm_samples_t *samples, void *user);
00255
00256 // Callback function for plm_buffer when it needs more data
00257
00258 typedef void (*plm_buffer_load_callback)(plm_buffer_t *self, void *user);
00259
00260 // -----------------------------------------------------------------------------
00261 // plm_* public API
00262 // High-Level API for loading/demuxing/decoding MPEG-PS data
00263
00264 // Create a plmpeg instance with a filename. Returns NULL if the file could not
00265 // be opened.
00266
00267 plm_t *plm_create_with_filename(const char *filename);
00268
00269 // Create a plmpeg instance with a file handle. Pass TRUE to close_when_done to
00270 // let plmpeg call fclose() on the handle when plm_destroy() is called.
00271
00272 plm_t *plm_create_with_file(FIL *fh, int close_when_done);
00273
00274 // Create a plmpeg instance with a pointer to memory as source. This assumes the
00275 // whole file is in memory. The memory is not copied. Pass TRUE to
00276 // free_when_done to let plmpeg call free() on the pointer when plm_destroy()
00277 // is called.
00278
```

```
00279 plm_t *plm_create_with_memory(uint8_t *bytes, size_t length, int free_when_done);
00280
00281 // Create a plmpeg instance with a plm_buffer as source. Pass TRUE to
00282 // destroy_when_done to let plmpeg call plm_buffer_destroy() on the buffer when
00283 // plm_destroy() is called.
00284
00285 plm_t *plm_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done);
00286
00287 // Destroy a plmpeg instance and free all data.
00288
00289 void plm_destroy(plm_t *self);
00290
00291 // Get whether we have headers on all available streams and we can accurately
00292 // report the number of video/audio streams, video dimensions, framerate and
00293 // audio samplerate.
00294 // This returns FALSE if the file is not an MPEG-PS file or - when not using a
00295 // file as source - when not enough data is available yet.
00296
00297 int plm_has_headers(plm_t *self);
00298
00299 // Get or set whether video decoding is enabled. Default TRUE.
00300
00301 int plm_get_video_enabled(plm_t *self);
00302 void plm_set_video_enabled(plm_t *self, int enabled);
00303
00304 // Get the number of video streams (0--1) reported in the system header.
00305
00306 int plm_get_num_video_streams(plm_t *self);
00307
00308 // Get the display width/height of the video stream.
00309
00310 int plm_get_width(plm_t *self);
00311 int plm_get_height(plm_t *self);
00312
00313 // Get the framerate of the video stream in frames per second.
00314
00315 double plm_get_framerate(plm_t *self);
00316
00317 // Get or set whether audio decoding is enabled. Default TRUE.
00318
00319 int plm_get_audio_enabled(plm_t *self);
00320 void plm_set_audio_enabled(plm_t *self, int enabled);
00321
00322 // Get the number of audio streams (0--4) reported in the system header.
00323
00324 int plm_get_num_audio_streams(plm_t *self);
00325
00326 // Set the desired audio stream (0--3). Default 0.
00327
00328 void plm_set_audio_stream(plm_t *self, int stream_index);
00329
00330 // Get the samplerate of the audio stream in samples per second.
00331
00332 int plm_get_samplerate(plm_t *self);
00333
00334 // Get or set the audio lead time in seconds - the time in which audio samples
00335 // are decoded in advance (or behind) the video decode time. Typically this
00336 // should be set to the duration of the buffer of the audio API that you use
00337 // for output. E.g. for SDL2: (SDL_AudioSpec.samples / samplerate)
00338
00339 double plm_get_audio_lead_time(plm_t *self);
00340 void plm_set_audio_lead_time(plm_t *self, double lead_time);
00341
00342 // Get the current internal time in seconds.
00343
00344 double plm_get_time(plm_t *self);
00345
00346 // Get the video duration of the underlying source in seconds.
00347
00348 double plm_get_duration(plm_t *self);
00349
00350 // Rewind all buffers back to the beginning.
00351
00352 void plm_rewind(plm_t *self);
00353
00354 // Get or set looping. Default FALSE.
00355
00356 int plm_get_loop(plm_t *self);
00357 void plm_set_loop(plm_t *self, int loop);
00358
00359 // Get whether the file has ended. If looping is enabled, this will always
00360 // return FALSE.
00361
00362 int plm_has_ended(plm_t *self);
00363
00364 // Set the callback for decoded video frames used with plm_decode(). If no
00365 // callback is set, video data will be ignored and not be decoded. The *user
```

```
00366 // Parameter will be passed to your callback.
00367
00368 void plm_set_video_decode_callback(plm_t *self, plm_video_decode_callback fp, void *user);
00369
00370 // Set the callback for decoded audio samples used with plm_decode(). If no
00371 // callback is set, audio data will be ignored and not be decoded. The *user
00372 // Parameter will be passed to your callback.
00373
00374 void plm_set_audio_decode_callback(plm_t *self, plm_audio_decode_callback fp, void *user);
00375
00376 // Advance the internal timer by seconds and decode video/audio up to this time.
00377 // This will call the video_decode_callback and audio_decode_callback any number
00378 // of times. A frame-skip is not implemented, i.e. everything up to current time
00379 // will be decoded.
00380
00381 void plm_decode(plm_t *self, double seconds);
00382
00383 // Decode and return one video frame. Returns NULL if no frame could be decoded
00384 // (either because the source ended or data is corrupt). If you only want to
00385 // decode video, you should disable audio via plm_set_audio_enabled().
00386 // The returned plm_frame_t is valid until the next call to plm_decode_video()
00387 // or until plm_destroy() is called.
00388
00389 plm_frame_t *plm_decode_video(plm_t *self);
00390
00391 // Decode and return one audio frame. Returns NULL if no frame could be decoded
00392 // (either because the source ended or data is corrupt). If you only want to
00393 // decode audio, you should disable video via plm_set_video_enabled().
00394 // The returned plm_samples_t is valid until the next call to plm_decode_audio()
00395 // or until plm_destroy() is called.
00396
00397 plm_samples_t *plm_decode_audio(plm_t *self);
00398
00399 // Seek to the specified time, clamped between 0 -- duration. This can only be
00400 // used when the underlying plm_buffer is seekable, i.e. for files, fixed
00401 // memory buffers or _for_appending buffers.
00402 // If seek_exact is TRUE this will seek to the exact time, otherwise it will
00403 // seek to the last intra frame just before the desired time. Exact seeking can
00404 // be slow, because all frames up to the seeked one have to be decoded on top of
00405 // the previous intra frame.
00406 // If seeking succeeds, this function will call the video_decode_callback
00407 // exactly once with the target frame. If audio is enabled, it will also call
00408 // the audio_decode_callback any number of times, until the audio_lead_time is
00409 // satisfied.
00410 // Returns TRUE if seeking succeeded or FALSE if no frame could be found.
00411
00412 int plm_seek(plm_t *self, double time, int seek_exact);
00413
00414 // Similar to plm_seek(), but will not call the video_decode_callback,
00415 // audio_decode_callback or make any attempts to sync audio.
00416 // Returns the found frame or NULL if no frame could be found.
00417
00418 plm_frame_t *plm_seek_frame(plm_t *self, double time, int seek_exact);
00419
00420 // -----------------------------------------------------------------------------
00421 // plm_buffer public API
00422 // Provides the data source for all other plm_* interfaces
00423
00424 // The default size for buffers created from files or by the high-level API
00425
00426 #ifndef PLM_BUFFER_DEFAULT_SIZE
00427 #define PLM_BUFFER_DEFAULT_SIZE (128 * 1024)
00428 #endif
00429
00430 // Create a buffer instance with a filename. Returns NULL if the file could not
00431 // be opened.
00432
00433 plm_buffer_t *plm_buffer_create_with_filename(const char *filename);
00434
00435 // Create a buffer instance with a file handle. Pass TRUE to close_when_done
00436 // to let plmpeg call fclose() on the handle when plm_destroy() is called.
00437
00438 plm_buffer_t *plm_buffer_create_with_file(FIL *fh, int close_when_done);
00439
00440 // Create a buffer instance with a pointer to memory as source. This assumes
00441 // the whole file is in memory. The bytes are not copied. Pass 1 to
00442 // free_when_done to let plmpeg call free() on the pointer when plm_destroy()
00443 // is called.
00444
00445 plm_buffer_t *plm_buffer_create_with_memory(uint8_t *bytes, size_t length, int free_when_done);
00446
00447 // Create an empty buffer with an initial capacity. The buffer will grow
00448 // as needed. Data that has already been read, will be discarded.
00449
00450 plm_buffer_t *plm_buffer_create_with_capacity(size_t capacity);
00451
00452 // Create an empty buffer with an initial capacity. The buffer will grow
```

```
00453 // as needed. Decoded data will *not* be discarded. This can be used when
00454 // loading a file over the network, without needing to throttle the download.
00455 // It also allows for seeking in the already loaded data.
00456
00457 plm_buffer_t *plm_buffer_create_for_appending(size_t initial_capacity);
00458
00459 // Destroy a buffer instance and free all data
00460
00461 void plm_buffer_destroy(plm_buffer_t *self);
00462
00463 // Copy data into the buffer. If the data to be written is larger than the
00464 // available space, the buffer will realloc() with a larger capacity.
00465 // Returns the number of bytes written. This will always be the same as the
00466 // passed in length, except when the buffer was created _with_memory() for
00467 // which _write() is forbidden.
00468
00469 size_t plm_buffer_write(plm_buffer_t *self, uint8_t *bytes, size_t length);
00470
00471 // Mark the current byte length as the end of this buffer and signal that no
00472 // more data is expected to be written to it. This function should be called
00473 // just after the last plm_buffer_write().
00474 // For _with_capacity buffers, this is cleared on a plm_buffer_rewind().
00475
00476 void plm_buffer_signal_end(plm_buffer_t *self);
00477
00478 // Set a callback that is called whenever the buffer needs more data
00479
00480 void plm_buffer_set_load_callback(plm_buffer_t *self, plm_buffer_load_callback fp, void *user);
00481
00482 // Rewind the buffer back to the beginning. When loading from a file handle,
00483 // this also seeks to the beginning of the file.
00484
00485 void plm_buffer_rewind(plm_buffer_t *self);
00486
00487 // Get the total size. For files, this returns the file size. For all other
00488 // types it returns the number of bytes currently in the buffer.
00489
00490 size_t plm_buffer_get_size(plm_buffer_t *self);
00491
00492 // Get the number of remaining (yet unread) bytes in the buffer. This can be
00493 // useful to throttle writing.
00494
00495 size_t plm_buffer_get_remaining(plm_buffer_t *self);
00496
00497 // Get whether the read position of the buffer is at the end and no more data
00498 // is expected.
00499
00500 int plm_buffer_has_ended(plm_buffer_t *self);
00501
00502 // -----------------------------------------------------------------------------
00503 // plm_demux public API
00504 // Demux an MPEG Program Stream (PS) data into separate packages
00505
00506 // Various Packet Types
00507
00508 static const int PLM_DEMUX_PACKET_PRIVATE = 0xBD;
00509 static const int PLM_DEMUX_PACKET_AUDIO_1 = 0xC0;
00510 static const int PLM_DEMUX_PACKET_AUDIO_2 = 0xC1;
00511 static const int PLM_DEMUX_PACKET_AUDIO_3 = 0xC2;
00512 static const int PLM_DEMUX_PACKET_AUDIO_4 = 0xC2;
00513 static const int PLM_DEMUX_PACKET_VIDEO_1 = 0xE0;
00514
00515 // Create a demuxer with a plm_buffer as source. This will also attempt to read
00516 // the pack and system headers from the buffer.
00517
00518 plm_demux_t *plm_demux_create(plm_buffer_t *buffer, int destroy_when_done);
00519
00520 // Destroy a demuxer and free all data.
00521
00522 void plm_demux_destroy(plm_demux_t *self);
00523
00524 // Returns TRUE/FALSE whether pack and system headers have been found. This will
00525 // attempt to read the headers if non are present yet.
00526
00527 int plm_demux_has_headers(plm_demux_t *self);
00528
00529 // Returns the number of video streams found in the system header. This will
00530 // attempt to read the system header if non is present yet.
00531
00532 int plm_demux_get_num_video_streams(plm_demux_t *self);
00533
00534 // Returns the number of audio streams found in the system header. This will
00535 // attempt to read the system header if non is present yet.
00536
00537 int plm_demux_get_num_audio_streams(plm_demux_t *self);
00538
00539 // Rewind the internal buffer. See plm_buffer_rewind().
```

```
00540
00541 void plm_demux_rewind(plm_demux_t *self);
00542
00543 // Get whether the file has ended. This will be cleared on seeking or rewind.
00544
00545 int plm_demux_has_ended(plm_demux_t *self);
00546
00547 // Seek to a packet of the specified type with a PTS just before specified time.
00548 // If force_intra is TRUE, only packets containing an intra frame will be
00549 // considered - this only makes sense when the type is PLM_DEMUX_PACKET_VIDEO_1.
00550 // Note that the specified time is considered 0-based, regardless of the first
00551 // PTS in the data source.
00552
00553 plm_packet_t *plm_demux_seek(plm_demux_t *self, double time, int type, int force_intra);
00554
00555 // Get the PTS of the first packet of this type. Returns PLM_PACKET_INVALID_TS
00556 // if not packet of this packet type can be found.
00557
00558 double plm_demux_get_start_time(plm_demux_t *self, int type);
00559
00560 // Get the duration for the specified packet type - i.e. the span between the
00561 // the first PTS and the last PTS in the data source. This only makes sense when
00562 // the underlying data source is a file or fixed memory.
00563
00564 double plm_demux_get_duration(plm_demux_t *self, int type);
00565
00566 // Decode and return the next packet. The returned packet_t is valid until
00567 // the next call to plm_demux_decode() or until the demuxer is destroyed.
00568
00569 plm_packet_t *plm_demux_decode(plm_demux_t *self);
00570
00571 // -----------------------------------------------------------------------------
00572 // plm_video public API
00573 // Decode MPEG1 Video ("mpeg1") data into raw YCrCb frames
00574
00575 // Create a video decoder with a plm_buffer as source.
00576
00577 plm_video_t *plm_video_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done);
00578
00579 // Destroy a video decoder and free all data.
00580
00581 void plm_video_destroy(plm_video_t *self);
00582
00583 // Get whether a sequence header was found and we can accurately report on
00584 // dimensions and framerate.
00585
00586 int plm_video_has_header(plm_video_t *self);
00587
00588 // Get the framerate in frames per second.
00589
00590 double plm_video_get_framerate(plm_video_t *self);
00591
00592 // Get the display width/height.
00593
00594 int plm_video_get_width(plm_video_t *self);
00595 int plm_video_get_height(plm_video_t *self);
00596
00597 // Set "no delay" mode. When enabled, the decoder assumes that the video does
00598 // *not* contain any B-Frames. This is useful for reducing lag when streaming.
00599 // The default is FALSE.
00600
00601 void plm_video_set_no_delay(plm_video_t *self, int no_delay);
00602
00603 // Get the current internal time in seconds.
00604
00605 double plm_video_get_time(plm_video_t *self);
00606
00607 // Set the current internal time in seconds. This is only useful when you
00608 // manipulate the underlying video buffer and want to enforce a correct
00609 // timestamps.
00610
00611 void plm_video_set_time(plm_video_t *self, double time);
00612
00613 // Rewind the internal buffer. See plm_buffer_rewind().
00614
00615 void plm_video_rewind(plm_video_t *self);
00616
00617 // Get whether the file has ended. This will be cleared on rewind.
00618
00619 int plm_video_has_ended(plm_video_t *self);
00620
00621 // Decode and return one frame of video and advance the internal time by
00622 // 1/framerate seconds. The returned frame_t is valid until the next call of
00623 // plm_video_decode() or until the video decoder is destroyed.
00624
00625 plm_frame_t *plm_video_decode(plm_video_t *self);
00626
```

```
00627 // Convert the YCrCb data of a frame into interleaved R G B data. The stride
00628 // specifies the width in bytes of the destination buffer. I.e. the number of
00629 // bytes from one line to the next. The stride must be at least
00630 // (frame->width * bytes_per_pixel). The buffer pointed to by *dest must have a
00631 // size of at least (stride * frame->height).
00632 // Note that the alpha component of the dest buffer is always left untouched.
00633
00634 void plm_frame_to_rgb(plm_frame_t *frame, uint8_t *dest, int stride);
00635 void plm_frame_to_bgr(plm_frame_t *frame, uint8_t *dest, int stride);
00636 void plm_frame_to_rgba(plm_frame_t *frame, uint8_t *dest, int stride);
00637 void plm_frame_to_bgra(plm_frame_t *frame, uint8_t *dest, int stride);
00638 void plm_frame_to_argb(plm_frame_t *frame, uint8_t *dest, int stride);
00639 void plm_frame_to_abgr(plm_frame_t *frame, uint8_t *dest, int stride);
00640
00641 // -----------------------------------------------------------------------------
00642 // plm_audio public API
00643 // Decode MPEG-1 Audio Layer II ("mp2") data into raw samples
00644
00645 // Create an audio decoder with a plm_buffer as source.
00646
00647 plm_audio_t *plm_audio_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done);
00648
00649 // Destroy an audio decoder and free all data.
00650
00651 void plm_audio_destroy(plm_audio_t *self);
00652
00653 // Get whether a frame header was found and we can accurately report on
00654 // samplerate.
00655
00656 int plm_audio_has_header(plm_audio_t *self);
00657
00658 // Get the samplerate in samples per second.
00659
00660 int plm_audio_get_samplerate(plm_audio_t *self);
00661
00662 // Get the current internal time in seconds.
00663
00664 double plm_audio_get_time(plm_audio_t *self);
00665
00666 // Set the current internal time in seconds. This is only useful when you
00667 // manipulate the underlying video buffer and want to enforce a correct
00668 // timestamps.
00669
00670 void plm_audio_set_time(plm_audio_t *self, double time);
00671
00672 // Rewind the internal buffer. See plm_buffer_rewind().
00673
00674 void plm_audio_rewind(plm_audio_t *self);
00675
00676 // Get whether the file has ended. This will be cleared on rewind.
00677
00678 int plm_audio_has_ended(plm_audio_t *self);
00679
00680 // Decode and return one "frame" of audio and advance the internal time by
00681 // (PLM_AUDIO_SAMPLES_PER_FRAME/samplerate) seconds. The returned samples_t
00682 // is valid until the next call of plm_audio_decode() or until the audio
00683 // decoder is destroyed.
00684
00685 plm_samples_t *plm_audio_decode(plm_audio_t *self);
00686
00687 #ifdef __cplusplus
00688 }
00689 #endif
00690
00691 #endif // PL_MPEG_H
00692
00693 // -----------------------------------------------------------------------------
00694 // -----------------------------------------------------------------------------
00695 // IMPLEMENTATION
00696
00697 #ifdef PL_MPEG_IMPLEMENTATION
00698
00699 #include <stdlib.h>
00700 #include <string.h>
00701
00702 #ifndef TRUE
00703 #define TRUE 1
00704 #define FALSE 0
00705 #endif
00706
00707 #ifndef PLM_MALLOC
00708 #define PLM_MALLOC(sz) malloc(sz)
00709 #define PLM_FREE(p) free(p)
00710 #define PLM_REALLOC(p, sz) realloc(p, sz)
00711 #endif
00712
00713 #define PLM_UNUSED(expr) (void)(expr)
```

```
00714
00715 // -----------------------------------------------------------------------------
00716 // plm (high-level interface) implementation
00717
00718 struct plm_t {
00719   plm_demux_t *demux;
00720   double time;
00721   int has_ended;
00722   int loop;
00723   int has_decoders;
00724
00725   int video_enabled;
00726   int video_packet_type;
00727   plm_buffer_t *video_buffer;
00728   plm_video_t *video_decoder;
00729
00730   int audio_enabled;
00731   int audio_stream_index;
00732   int audio_packet_type;
00733   double audio_lead_time;
00734   plm_buffer_t *audio_buffer;
00735   plm_audio_t *audio_decoder;
00736
00737   plm_video_decode_callback video_decode_callback;
00738   void *video_decode_callback_user_data;
00739
00740   plm_audio_decode_callback audio_decode_callback;
00741   void *audio_decode_callback_user_data;
00742 };
00743
00744 int plm_init_decoders(plm_t *self);
00745 void plm_handle_end(plm_t *self);
00746 void plm_read_video_packet(plm_buffer_t *buffer, void *user);
00747 void plm_read_audio_packet(plm_buffer_t *buffer, void *user);
00748 void plm_read_packets(plm_t *self, int requested_type);
00749
00750 plm_t *plm_create_with_filename(const char *filename) {
00751   plm_buffer_t *buffer = plm_buffer_create_with_filename(filename);
00752   if (!buffer) {
00753     return NULL;
00754   }
00755   return plm_create_with_buffer(buffer, TRUE);
00756 }
00757
00758 plm_t *plm_create_with_file(FIL *fh, int close_when_done) {
00759   plm_buffer_t *buffer = plm_buffer_create_with_file(fh, close_when_done);
00760   return plm_create_with_buffer(buffer, TRUE);
00761 }
00762
00763 plm_t *plm_create_with_memory(uint8_t *bytes, size_t length, int free_when_done) {
00764   plm_buffer_t *buffer = plm_buffer_create_with_memory(bytes, length, free_when_done);
00765   return plm_create_with_buffer(buffer, TRUE);
00766 }
00767
00768 plm_t *plm_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done) {
00769   plm_t *self = (plm_t *)PLM_MALLOC(sizeof(plm_t));
00770   memset(self, 0, sizeof(plm_t));
00771
00772   self->demux = plm_demux_create(buffer, destroy_when_done);
00773   self->video_enabled = TRUE;
00774   self->audio_enabled = TRUE;
00775   plm_init_decoders(self);
00776
00777   return self;
00778 }
00779
00780 int plm_init_decoders(plm_t *self) {
00781   if (self->has_decoders) {
00782     return TRUE;
00783   }
00784
00785   if (!plm_demux_has_headers(self->demux)) {
00786     return FALSE;
00787   }
00788
00789   if (plm_demux_get_num_video_streams(self->demux) > 0) {
00790     if (self->video_enabled) {
00791       self->video_packet_type = PLM_DEMUX_PACKET_VIDEO_1;
00792     }
00793     self->video_buffer = plm_buffer_create_with_capacity(PLM_BUFFER_DEFAULT_SIZE);
00794     plm_buffer_set_load_callback(self->video_buffer, plm_read_video_packet, self);
00795   }
00796
00797   if (plm_demux_get_num_audio_streams(self->demux) > 0) {
00798     if (self->audio_enabled) {
00799       self->audio_packet_type = PLM_DEMUX_PACKET_AUDIO_1 + self->audio_stream_index;
00800     }
```

```
00801     self->audio_buffer = plm_buffer_create_with_capacity(PLM_BUFFER_DEFAULT_SIZE);
00802     plm_buffer_set_load_callback(self->audio_buffer, plm_read_audio_packet, self);
00803   }
00804
00805   if (self->video_buffer) {
00806     self->video_decoder = plm_video_create_with_buffer(self->video_buffer, TRUE);
00807   }
00808
00809   if (self->audio_buffer) {
00810     self->audio_decoder = plm_audio_create_with_buffer(self->audio_buffer, TRUE);
00811   }
00812
00813   self->has_decoders = TRUE;
00814   return TRUE;
00815 }
00816
00817 void plm_destroy(plm_t *self) {
00818   if (self->video_decoder) {
00819     plm_video_destroy(self->video_decoder);
00820   }
00821   if (self->audio_decoder) {
00822     plm_audio_destroy(self->audio_decoder);
00823   }
00824
00825   plm_demux_destroy(self->demux);
00826   PLM_FREE(self);
00827 }
00828
00829 int plm_get_audio_enabled(plm_t *self) { return self->audio_enabled; }
00830
00831 int plm_has_headers(plm_t *self) {
00832   if (!plm_demux_has_headers(self->demux)) {
00833     return FALSE;
00834   }
00835
00836   if (!plm_init_decoders(self)) {
00837     return FALSE;
00838   }
00839
00840   if ((self->video_decoder && !plm_video_has_header(self->video_decoder)) ||
00841       (self->audio_decoder && !plm_audio_has_header(self->audio_decoder))) {
00842     return FALSE;
00843   }
00844
00845   return TRUE;
00846 }
00847
00848 void plm_set_audio_enabled(plm_t *self, int enabled) {
00849   self->audio_enabled = enabled;
00850
00851   if (!enabled) {
00852     self->audio_packet_type = 0;
00853     return;
00854   }
00855
00856   self->audio_packet_type =
00857       (plm_init_decoders(self) && self->audio_decoder) ? PLM_DEMUX_PACKET_AUDIO_1 +
      self->audio_stream_index : 0;
00858 }
00859
00860 void plm_set_audio_stream(plm_t *self, int stream_index) {
00861   if (stream_index < 0 || stream_index > 3) {
00862     return;
00863   }
00864   self->audio_stream_index = stream_index;
00865
00866   // Set the correct audio_packet_type
00867   plm_set_audio_enabled(self, self->audio_enabled);
00868 }
00869
00870 int plm_get_video_enabled(plm_t *self) { return self->video_enabled; }
00871
00872 void plm_set_video_enabled(plm_t *self, int enabled) {
00873   self->video_enabled = enabled;
00874
00875   if (!enabled) {
00876     self->video_packet_type = 0;
00877     return;
00878   }
00879
00880   self->video_packet_type = (plm_init_decoders(self) && self->video_decoder) ?
      PLM_DEMUX_PACKET_VIDEO_1 : 0;
00881 }
00882
00883 int plm_get_num_video_streams(plm_t *self) { return plm_demux_get_num_video_streams(self->demux); }
00884
00885 int plm_get_width(plm_t *self) {
```

```
00886   return (plm_init_decoders(self) && self->video_decoder) ? plm_video_get_width(self->video_decoder) :
        0;
00887 }
00888
00889 int plm_get_height(plm_t *self) {
00890   return (plm_init_decoders(self) && self->video_decoder) ? plm_video_get_height(self->video_decoder)
        : 0;
00891 }
00892
00893 double plm_get_framerate(plm_t *self) {
00894   return (plm_init_decoders(self) && self->video_decoder) ?
        plm_video_get_framerate(self->video_decoder) : 0;
00895 }
00896
00897 int plm_get_num_audio_streams(plm_t *self) { return plm_demux_get_num_audio_streams(self->demux); }
00898
00899 int plm_get_samplerate(plm_t *self) {
00900   return (plm_init_decoders(self) && self->audio_decoder) ?
        plm_audio_get_samplerate(self->audio_decoder) : 0;
00901 }
00902
00903 double plm_get_audio_lead_time(plm_t *self) { return self->audio_lead_time; }
00904
00905 void plm_set_audio_lead_time(plm_t *self, double lead_time) { self->audio_lead_time = lead_time; }
00906
00907 double plm_get_time(plm_t *self) { return self->time; }
00908
00909 double plm_get_duration(plm_t *self) { return plm_demux_get_duration(self->demux,
        PLM_DEMUX_PACKET_VIDEO_1); }
00910
00911 void plm_rewind(plm_t *self) {
00912   if (self->video_decoder) {
00913     plm_video_rewind(self->video_decoder);
00914   }
00915
00916   if (self->audio_decoder) {
00917     plm_audio_rewind(self->audio_decoder);
00918   }
00919
00920   plm_demux_rewind(self->demux);
00921   self->time = 0;
00922 }
00923
00924 int plm_get_loop(plm_t *self) { return self->loop; }
00925
00926 void plm_set_loop(plm_t *self, int loop) { self->loop = loop; }
00927
00928 int plm_has_ended(plm_t *self) { return self->has_ended; }
00929
00930 void plm_set_video_decode_callback(plm_t *self, plm_video_decode_callback fp, void *user) {
00931   self->video_decode_callback = fp;
00932   self->video_decode_callback_user_data = user;
00933 }
00934
00935 void plm_set_audio_decode_callback(plm_t *self, plm_audio_decode_callback fp, void *user) {
00936   self->audio_decode_callback = fp;
00937   self->audio_decode_callback_user_data = user;
00938 }
00939
00940 void plm_decode(plm_t *self, double tick) {
00941   if (!plm_init_decoders(self)) {
00942     return;
00943   }
00944
00945   int decode_video = (self->video_decode_callback && self->video_packet_type);
00946   int decode_audio = (self->audio_decode_callback && self->audio_packet_type);
00947
00948   if (!decode_video && !decode_audio) {
00949     // Nothing to do here
00950     return;
00951   }
00952
00953   int did_decode = FALSE;
00954   int decode_video_failed = FALSE;
00955   int decode_audio_failed = FALSE;
00956
00957   double video_target_time = self->time + tick;
00958   double audio_target_time = self->time + tick + self->audio_lead_time;
00959
00960   do {
00961     did_decode = FALSE;
00962
00963     if (decode_video && plm_video_get_time(self->video_decoder) < video_target_time) {
00964       plm_frame_t *frame = plm_video_decode(self->video_decoder);
00965       if (frame) {
00966         self->video_decode_callback(self, frame, self->video_decode_callback_user_data);
00967         did_decode = TRUE;
```

```
00968       } else {
00969         decode_video_failed = TRUE;
00970       }
00971     }
00972
00973     if (decode_audio && plm_audio_get_time(self->audio_decoder) < audio_target_time) {
00974       plm_samples_t *samples = plm_audio_decode(self->audio_decoder);
00975       if (samples) {
00976         self->audio_decode_callback(self, samples, self->audio_decode_callback_user_data);
00977         did_decode = TRUE;
00978       } else {
00979         decode_audio_failed = TRUE;
00980       }
00981     }
00982   } while (did_decode);
00983
00984   // Did all sources we wanted to decode fail and the demuxer is at the end?
00985   if ((!decode_video || decode_video_failed) && (!decode_audio || decode_audio_failed) &&
00986       plm_demux_has_ended(self->demux)) {
00987     plm_handle_end(self);
00988     return;
00989   }
00990
00991   self->time += tick;
00992 }
00993
00994 plm_frame_t *plm_decode_video(plm_t *self) {
00995   if (!plm_init_decoders(self)) {
00996     return NULL;
00997   }
00998
00999   if (!self->video_packet_type) {
01000     return NULL;
01001   }
01002
01003   plm_frame_t *frame = plm_video_decode(self->video_decoder);
01004   if (frame) {
01005     self->time = frame->time;
01006   } else if (plm_demux_has_ended(self->demux)) {
01007     plm_handle_end(self);
01008   }
01009   return frame;
01010 }
01011
01012 plm_samples_t *plm_decode_audio(plm_t *self) {
01013   if (!plm_init_decoders(self)) {
01014     return NULL;
01015   }
01016
01017   if (!self->audio_packet_type) {
01018     return NULL;
01019   }
01020
01021   plm_samples_t *samples = plm_audio_decode(self->audio_decoder);
01022   if (samples) {
01023     self->time = samples->time;
01024   } else if (plm_demux_has_ended(self->demux)) {
01025     plm_handle_end(self);
01026   }
01027   return samples;
01028 }
01029
01030 void plm_handle_end(plm_t *self) {
01031   if (self->loop) {
01032     plm_rewind(self);
01033   } else {
01034     self->has_ended = TRUE;
01035   }
01036 }
01037
01038 void plm_read_video_packet(plm_buffer_t *buffer, void *user) {
01039   PLM_UNUSED(buffer);
01040   plm_t *self = (plm_t *)user;
01041   plm_read_packets(self, self->video_packet_type);
01042 }
01043
01044 void plm_read_audio_packet(plm_buffer_t *buffer, void *user) {
01045   PLM_UNUSED(buffer);
01046   plm_t *self = (plm_t *)user;
01047   plm_read_packets(self, self->audio_packet_type);
01048 }
01049
01050 void plm_read_packets(plm_t *self, int requested_type) {
01051   plm_packet_t *packet;
01052   while ((packet = plm_demux_decode(self->demux))) {
01053     if (packet->type == self->video_packet_type) {
01054       plm_buffer_write(self->video_buffer, packet->data, packet->length);
```

```
01055     } else if (packet->type == self->audio_packet_type) {
01056       plm_buffer_write(self->audio_buffer, packet->data, packet->length);
01057     }
01058
01059     if (packet->type == requested_type) {
01060       return;
01061     }
01062   }
01063
01064   if (plm_demux_has_ended(self->demux)) {
01065     if (self->video_buffer) {
01066       plm_buffer_signal_end(self->video_buffer);
01067     }
01068     if (self->audio_buffer) {
01069       plm_buffer_signal_end(self->audio_buffer);
01070     }
01071   }
01072 }
01073
01074 plm_frame_t *plm_seek_frame(plm_t *self, double time, int seek_exact) {
01075   if (!plm_init_decoders(self)) {
01076     return NULL;
01077   }
01078
01079   if (!self->video_packet_type) {
01080     return NULL;
01081   }
01082
01083   int type = self->video_packet_type;
01084
01085   double start_time = plm_demux_get_start_time(self->demux, type);
01086   double duration = plm_demux_get_duration(self->demux, type);
01087
01088   if (time < 0) {
01089     time = 0;
01090   } else if (time > duration) {
01091     time = duration;
01092   }
01093
01094   plm_packet_t *packet = plm_demux_seek(self->demux, time, type, TRUE);
01095   if (!packet) {
01096     return NULL;
01097   }
01098
01099   // Disable writing to the audio buffer while decoding video
01100   int previous_audio_packet_type = self->audio_packet_type;
01101   self->audio_packet_type = 0;
01102
01103   // Clear video buffer and decode the found packet
01104   plm_video_rewind(self->video_decoder);
01105   plm_video_set_time(self->video_decoder, packet->pts - start_time);
01106   plm_buffer_write(self->video_buffer, packet->data, packet->length);
01107   plm_frame_t *frame = plm_video_decode(self->video_decoder);
01108
01109   // If we want to seek to an exact frame, we have to decode all frames
01110   // on top of the intra frame we just jumped to.
01111   if (seek_exact) {
01112     while (frame && frame->time < time) {
01113       frame = plm_video_decode(self->video_decoder);
01114     }
01115   }
01116
01117   // Enable writing to the audio buffer again?
01118   self->audio_packet_type = previous_audio_packet_type;
01119
01120   if (frame) {
01121     self->time = frame->time;
01122   }
01123
01124   self->has_ended = FALSE;
01125   return frame;
01126 }
01127
01128 int plm_seek(plm_t *self, double time, int seek_exact) {
01129   plm_frame_t *frame = plm_seek_frame(self, time, seek_exact);
01130
01131   if (!frame) {
01132     return FALSE;
01133   }
01134
01135   if (self->video_decode_callback) {
01136     self->video_decode_callback(self, frame, self->video_decode_callback_user_data);
01137   }
01138
01139   // If audio is not enabled we are done here.
01140   if (!self->audio_packet_type) {
01141     return TRUE;
```

```
01142   }
01143
01144   // Sync up Audio. This demuxes more packets until the first audio packet
01145   // with a PTS greater than the current time is found. plm_decode() is then
01146   // called to decode enough audio data to satisfy the audio_lead_time.
01147
01148   double start_time = plm_demux_get_start_time(self->demux, self->video_packet_type);
01149   plm_audio_rewind(self->audio_decoder);
01150
01151   plm_packet_t *packet = NULL;
01152   while ((packet = plm_demux_decode(self->demux))) {
01153     if (packet->type == self->video_packet_type) {
01154       plm_buffer_write(self->video_buffer, packet->data, packet->length);
01155     } else if (packet->type == self->audio_packet_type && packet->pts - start_time > self->time) {
01156       plm_audio_set_time(self->audio_decoder, packet->pts - start_time);
01157       plm_buffer_write(self->audio_buffer, packet->data, packet->length);
01158       plm_decode(self, 0);
01159       break;
01160     }
01161   }
01162
01163   return TRUE;
01164 }
01165
01166 // -----------------------------------------------------------------------------
01167 // plm_buffer implementation
01168
01169 enum plm_buffer_mode { PLM_BUFFER_MODE_FILE, PLM_BUFFER_MODE_FIXED_MEM, PLM_BUFFER_MODE_RING,
      PLM_BUFFER_MODE_APPEND };
01170
01171 struct plm_buffer_t {
01172   size_t bit_index;
01173   size_t capacity;
01174   size_t length;
01175   size_t total_size;
01176   int discard_read_bytes;
01177   int has_ended;
01178   int free_when_done;
01179   int close_when_done;
01180   FIL *fh;
01181   plm_buffer_load_callback load_callback;
01182   void *load_callback_user_data;
01183   uint8_t *bytes;
01184   enum plm_buffer_mode mode;
01185 };
01186
01187 typedef struct {
01188   int16_t index;
01189   int16_t value;
01190 } plm_vlc_t;
01191
01192 typedef struct {
01193   int16_t index;
01194   uint16_t value;
01195 } plm_vlc_uint_t;
01196
01197 void plm_buffer_seek(plm_buffer_t *self, size_t pos);
01198 size_t plm_buffer_tell(plm_buffer_t *self);
01199 void plm_buffer_discard_read_bytes(plm_buffer_t *self);
01200 void plm_buffer_load_file_callback(plm_buffer_t *self, void *user);
01201
01202 int plm_buffer_has(plm_buffer_t *self, size_t count);
01203 int plm_buffer_read(plm_buffer_t *self, int count);
01204 void plm_buffer_align(plm_buffer_t *self);
01205 void plm_buffer_skip(plm_buffer_t *self, size_t count);
01206 int plm_buffer_skip_bytes(plm_buffer_t *self, uint8_t v);
01207 int plm_buffer_next_start_code(plm_buffer_t *self);
01208 int plm_buffer_find_start_code(plm_buffer_t *self, int code);
01209 int plm_buffer_no_start_code(plm_buffer_t *self);
01210 int16_t plm_buffer_read_vlc(plm_buffer_t *self, const plm_vlc_t *table);
01211 uint16_t plm_buffer_read_vlc_uint(plm_buffer_t *self, const plm_vlc_uint_t *table);
01212
01213 plm_buffer_t *plm_buffer_create_with_filename(const char *filename) {
01214   FIL *fh = vexFileOpen(filename, "rb"); // fopen(filename, "rb");
01215   if (!fh) {
01216     return NULL;
01217   }
01218   return plm_buffer_create_with_file(fh, TRUE);
01219 }
01220
01221 plm_buffer_t *plm_buffer_create_with_file(FIL *fh, int close_when_done) {
01222   plm_buffer_t *self = plm_buffer_create_with_capacity(PLM_BUFFER_DEFAULT_SIZE);
01223   self->fh = fh;
01224   self->close_when_done = close_when_done;
01225   self->mode = PLM_BUFFER_MODE_FILE;
01226   self->discard_read_bytes = TRUE;
01227
```

```
01228   vexFileSeek(self->fh, 0, SEEK_END);
01229   self->total_size = vexFileTell(self->fh);
01230   vexFileSeek(self->fh, 0, SEEK_SET);
01231
01232   plm_buffer_set_load_callback(self, plm_buffer_load_file_callback, NULL);
01233   return self;
01234 }
01235
01236 plm_buffer_t *plm_buffer_create_with_memory(uint8_t *bytes, size_t length, int free_when_done) {
01237   plm_buffer_t *self = (plm_buffer_t *)PLM_MALLOC(sizeof(plm_buffer_t));
01238   memset(self, 0, sizeof(plm_buffer_t));
01239   self->capacity = length;
01240   self->length = length;
01241   self->total_size = length;
01242   self->free_when_done = free_when_done;
01243   self->bytes = bytes;
01244   self->mode = PLM_BUFFER_MODE_FIXED_MEM;
01245   self->discard_read_bytes = FALSE;
01246   return self;
01247 }
01248
01249 plm_buffer_t *plm_buffer_create_with_capacity(size_t capacity) {
01250   plm_buffer_t *self = (plm_buffer_t *)PLM_MALLOC(sizeof(plm_buffer_t));
01251   memset(self, 0, sizeof(plm_buffer_t));
01252   self->capacity = capacity;
01253   self->free_when_done = TRUE;
01254   self->bytes = (uint8_t *)PLM_MALLOC(capacity);
01255   self->mode = PLM_BUFFER_MODE_RING;
01256   self->discard_read_bytes = TRUE;
01257   return self;
01258 }
01259
01260 plm_buffer_t *plm_buffer_create_for_appending(size_t initial_capacity) {
01261   plm_buffer_t *self = plm_buffer_create_with_capacity(initial_capacity);
01262   self->mode = PLM_BUFFER_MODE_APPEND;
01263   self->discard_read_bytes = FALSE;
01264   return self;
01265 }
01266
01267 void plm_buffer_destroy(plm_buffer_t *self) {
01268   if (self->fh && self->close_when_done) {
01269     vexFileClose(self->fh);
01270   }
01271   if (self->free_when_done) {
01272     PLM_FREE(self->bytes);
01273   }
01274   PLM_FREE(self);
01275 }
01276
01277 size_t plm_buffer_get_size(plm_buffer_t *self) {
01278   return (self->mode == PLM_BUFFER_MODE_FILE) ? self->total_size : self->length;
01279 }
01280
01281 size_t plm_buffer_get_remaining(plm_buffer_t *self) { return self->length - (self->bit_index >> 3); }
01282
01283 size_t plm_buffer_write(plm_buffer_t *self, uint8_t *bytes, size_t length) {
01284   if (self->mode == PLM_BUFFER_MODE_FIXED_MEM) {
01285     return 0;
01286   }
01287
01288   if (self->discard_read_bytes) {
01289     // This should be a ring buffer, but instead it just shifts all unread
01290     // data to the beginning of the buffer and appends new data at the end.
01291     // Seems to be good enough.
01292
01293     plm_buffer_discard_read_bytes(self);
01294     if (self->mode == PLM_BUFFER_MODE_RING) {
01295       self->total_size = 0;
01296     }
01297   }
01298
01299   // Do we have to resize to fit the new data?
01300   size_t bytes_available = self->capacity - self->length;
01301   if (bytes_available < length) {
01302     size_t new_size = self->capacity;
01303     do {
01304       new_size *= 2;
01305     } while (new_size - self->length < length);
01306     self->bytes = (uint8_t *)PLM_REALLOC(self->bytes, new_size);
01307     self->capacity = new_size;
01308   }
01309
01310   memcpy(self->bytes + self->length, bytes, length);
01311   self->length += length;
01312   self->has_ended = FALSE;
01313   return length;
01314 }
```

```
01315
01316 void plm_buffer_signal_end(plm_buffer_t *self) { self->total_size = self->length; }
01317
01318 void plm_buffer_set_load_callback(plm_buffer_t *self, plm_buffer_load_callback fp, void *user) {
01319   self->load_callback = fp;
01320   self->load_callback_user_data = user;
01321 }
01322
01323 void plm_buffer_rewind(plm_buffer_t *self) { plm_buffer_seek(self, 0); }
01324
01325 void plm_buffer_seek(plm_buffer_t *self, size_t pos) {
01326   self->has_ended = FALSE;
01327
01328   if (self->mode == PLM_BUFFER_MODE_FILE) {
01329     vexFileSeek(self->fh, pos, SEEK_SET);
01330     self->bit_index = 0;
01331     self->length = 0;
01332   } else if (self->mode == PLM_BUFFER_MODE_RING) {
01333     if (pos != 0) {
01334       // Seeking to non-0 is forbidden for dynamic-mem buffers
01335       return;
01336     }
01337     self->bit_index = 0;
01338     self->length = 0;
01339     self->total_size = 0;
01340   } else if (pos < self->length) {
01341     self->bit_index = pos « 3;
01342   }
01343 }
01344
01345 size_t plm_buffer_tell(plm_buffer_t *self) {
01346   return self->mode == PLM_BUFFER_MODE_FILE ? vexFileTell(self->fh) + (self->bit_index » 3) -
      self->length
01347                                             : self->bit_index » 3;
01348 }
01349
01350 void plm_buffer_discard_read_bytes(plm_buffer_t *self) {
01351   size_t byte_pos = self->bit_index » 3;
01352   if (byte_pos == self->length) {
01353     self->bit_index = 0;
01354     self->length = 0;
01355   } else if (byte_pos > 0) {
01356     memmove(self->bytes, self->bytes + byte_pos, self->length - byte_pos);
01357     self->bit_index -= byte_pos « 3;
01358     self->length -= byte_pos;
01359   }
01360 }
01361
01362 void plm_buffer_load_file_callback(plm_buffer_t *self, void *user) {
01363   PLM_UNUSED(user);
01364
01365   if (self->discard_read_bytes) {
01366     plm_buffer_discard_read_bytes(self);
01367   }
01368
01369   size_t bytes_available = self->capacity - self->length;
01370   size_t bytes_read = vexFileRead((char *)self->bytes + self->length, 1, bytes_available, self->fh);
01371   self->length += bytes_read;
01372
01373   if (bytes_read == 0) {
01374     self->has_ended = TRUE;
01375   }
01376 }
01377
01378 int plm_buffer_has_ended(plm_buffer_t *self) { return self->has_ended; }
01379
01380 int plm_buffer_has(plm_buffer_t *self, size_t count) {
01381   if (((self->length « 3) - self->bit_index) >= count) {
01382     return TRUE;
01383   }
01384
01385   if (self->load_callback) {
01386     self->load_callback(self, self->load_callback_user_data);
01387
01388     if (((self->length « 3) - self->bit_index) >= count) {
01389       return TRUE;
01390     }
01391   }
01392
01393   if (self->total_size != 0 && self->length == self->total_size) {
01394     self->has_ended = TRUE;
01395   }
01396   return FALSE;
01397 }
01398
01399 int plm_buffer_read(plm_buffer_t *self, int count) {
01400   if (!plm_buffer_has(self, count)) {
```

```
01401     return 0;
01402   }
01403
01404   int value = 0;
01405   while (count) {
01406     int current_byte = self->bytes[self->bit_index » 3];
01407
01408     int remaining = 8 - (self->bit_index & 7);        // Remaining bits in byte
01409     int read = remaining < count ? remaining : count; // Bits in self run
01410     int shift = remaining - read;
01411     int mask = (0xff » (8 - read));
01412
01413     value = (value « read) | ((current_byte & (mask « shift)) » shift);
01414
01415     self->bit_index += read;
01416     count -= read;
01417   }
01418
01419   return value;
01420 }
01421
01422 void plm_buffer_align(plm_buffer_t *self) {
01423   self->bit_index = ((self->bit_index + 7) » 3) « 3; // Align to next byte
01424 }
01425
01426 void plm_buffer_skip(plm_buffer_t *self, size_t count) {
01427   if (plm_buffer_has(self, count)) {
01428     self->bit_index += count;
01429   }
01430 }
01431
01432 int plm_buffer_skip_bytes(plm_buffer_t *self, uint8_t v) {
01433   plm_buffer_align(self);
01434   int skipped = 0;
01435   while (plm_buffer_has(self, 8) && self->bytes[self->bit_index » 3] == v) {
01436     self->bit_index += 8;
01437     skipped++;
01438   }
01439   return skipped;
01440 }
01441
01442 int plm_buffer_next_start_code(plm_buffer_t *self) {
01443   plm_buffer_align(self);
01444
01445   while (plm_buffer_has(self, (5 « 3))) {
01446     size_t byte_index = (self->bit_index) » 3;
01447     if (self->bytes[byte_index] == 0x00 && self->bytes[byte_index + 1] == 0x00 &&
      self->bytes[byte_index + 2] == 0x01) {
01448       self->bit_index = (byte_index + 4) « 3;
01449       return self->bytes[byte_index + 3];
01450     }
01451     self->bit_index += 8;
01452   }
01453   return -1;
01454 }
01455
01456 int plm_buffer_find_start_code(plm_buffer_t *self, int code) {
01457   int current = 0;
01458   while (TRUE) {
01459     current = plm_buffer_next_start_code(self);
01460     if (current == code || current == -1) {
01461       return current;
01462     }
01463   }
01464   return -1;
01465 }
01466
01467 int plm_buffer_has_start_code(plm_buffer_t *self, int code) {
01468   size_t previous_bit_index = self->bit_index;
01469   int previous_discard_read_bytes = self->discard_read_bytes;
01470
01471   self->discard_read_bytes = FALSE;
01472   int current = plm_buffer_find_start_code(self, code);
01473
01474   self->bit_index = previous_bit_index;
01475   self->discard_read_bytes = previous_discard_read_bytes;
01476   return current;
01477 }
01478
01479 int plm_buffer_peek_non_zero(plm_buffer_t *self, int bit_count) {
01480   if (!plm_buffer_has(self, bit_count)) {
01481     return FALSE;
01482   }
01483
01484   int val = plm_buffer_read(self, bit_count);
01485   self->bit_index -= bit_count;
01486   return val != 0;
```

```
01487 }
01488
01489 int16_t plm_buffer_read_vlc(plm_buffer_t *self, const plm_vlc_t *table) {
01490   plm_vlc_t state = {0, 0};
01491   do {
01492     state = table[state.index + plm_buffer_read(self, 1)];
01493   } while (state.index > 0);
01494   return state.value;
01495 }
01496
01497 uint16_t plm_buffer_read_vlc_uint(plm_buffer_t *self, const plm_vlc_uint_t *table) {
01498   return (uint16_t)plm_buffer_read_vlc(self, (const plm_vlc_t *)table);
01499 }
01500
01501 // -----------------------------------------------------------------------------
01502 // plm_demux implementation
01503
01504 static const int PLM_START_PACK = 0xBA;
01505 static const int PLM_START_END = 0xB9;
01506 static const int PLM_START_SYSTEM = 0xBB;
01507
01508 struct plm_demux_t {
01509   plm_buffer_t *buffer;
01510   int destroy_buffer_when_done;
01511   double system_clock_ref;
01512
01513   size_t last_file_size;
01514   double last_decoded_pts;
01515   double start_time;
01516   double duration;
01517
01518   int start_code;
01519   int has_pack_header;
01520   int has_system_header;
01521   int has_headers;
01522
01523   int num_audio_streams;
01524   int num_video_streams;
01525   plm_packet_t current_packet;
01526   plm_packet_t next_packet;
01527 };
01528
01529 void plm_demux_buffer_seek(plm_demux_t *self, size_t pos);
01530 double plm_demux_decode_time(plm_demux_t *self);
01531 plm_packet_t *plm_demux_decode_packet(plm_demux_t *self, int type);
01532 plm_packet_t *plm_demux_get_packet(plm_demux_t *self);
01533
01534 plm_demux_t *plm_demux_create(plm_buffer_t *buffer, int destroy_when_done) {
01535   plm_demux_t *self = (plm_demux_t *)PLM_MALLOC(sizeof(plm_demux_t));
01536   memset(self, 0, sizeof(plm_demux_t));
01537
01538   self->buffer = buffer;
01539   self->destroy_buffer_when_done = destroy_when_done;
01540
01541   self->start_time = PLM_PACKET_INVALID_TS;
01542   self->duration = PLM_PACKET_INVALID_TS;
01543   self->start_code = -1;
01544
01545   plm_demux_has_headers(self);
01546   return self;
01547 }
01548
01549 void plm_demux_destroy(plm_demux_t *self) {
01550   if (self->destroy_buffer_when_done) {
01551     plm_buffer_destroy(self->buffer);
01552   }
01553   PLM_FREE(self);
01554 }
01555
01556 int plm_demux_has_headers(plm_demux_t *self) {
01557   if (self->has_headers) {
01558     return TRUE;
01559   }
01560
01561   // Decode pack header
01562   if (!self->has_pack_header) {
01563     if (self->start_code != PLM_START_PACK && plm_buffer_find_start_code(self->buffer, PLM_START_PACK)
      == -1) {
01564       return FALSE;
01565     }
01566
01567     self->start_code = PLM_START_PACK;
01568     if (!plm_buffer_has(self->buffer, 64)) {
01569       return FALSE;
01570     }
01571     self->start_code = -1;
01572
```

```
01573     if (plm_buffer_read(self->buffer, 4) != 0x02) {
01574       return FALSE;
01575     }
01576
01577     self->system_clock_ref = plm_demux_decode_time(self);
01578     plm_buffer_skip(self->buffer, 1);
01579     plm_buffer_skip(self->buffer, 22); // mux_rate * 50
01580     plm_buffer_skip(self->buffer, 1);
01581
01582     self->has_pack_header = TRUE;
01583   }
01584
01585   // Decode system header
01586   if (!self->has_system_header) {
01587     if (self->start_code != PLM_START_SYSTEM && plm_buffer_find_start_code(self->buffer,
      PLM_START_SYSTEM) == -1) {
01588       return FALSE;
01589     }
01590
01591     self->start_code = PLM_START_SYSTEM;
01592     if (!plm_buffer_has(self->buffer, 56)) {
01593       return FALSE;
01594     }
01595     self->start_code = -1;
01596
01597     plm_buffer_skip(self->buffer, 16); // header_length
01598     plm_buffer_skip(self->buffer, 24); // rate bound
01599     self->num_audio_streams = plm_buffer_read(self->buffer, 6);
01600     plm_buffer_skip(self->buffer, 5); // misc flags
01601     self->num_video_streams = plm_buffer_read(self->buffer, 5);
01602
01603     self->has_system_header = TRUE;
01604   }
01605
01606   self->has_headers = TRUE;
01607   return TRUE;
01608 }
01609
01610 int plm_demux_get_num_video_streams(plm_demux_t *self) {
01611   return plm_demux_has_headers(self) ? self->num_video_streams : 0;
01612 }
01613
01614 int plm_demux_get_num_audio_streams(plm_demux_t *self) {
01615   return plm_demux_has_headers(self) ? self->num_audio_streams : 0;
01616 }
01617
01618 void plm_demux_rewind(plm_demux_t *self) {
01619   plm_buffer_rewind(self->buffer);
01620   self->current_packet.length = 0;
01621   self->next_packet.length = 0;
01622   self->start_code = -1;
01623 }
01624
01625 int plm_demux_has_ended(plm_demux_t *self) { return plm_buffer_has_ended(self->buffer); }
01626
01627 void plm_demux_buffer_seek(plm_demux_t *self, size_t pos) {
01628   plm_buffer_seek(self->buffer, pos);
01629   self->current_packet.length = 0;
01630   self->next_packet.length = 0;
01631   self->start_code = -1;
01632 }
01633
01634 double plm_demux_get_start_time(plm_demux_t *self, int type) {
01635   if (self->start_time != PLM_PACKET_INVALID_TS) {
01636     return self->start_time;
01637   }
01638
01639   int previous_pos = plm_buffer_tell(self->buffer);
01640   int previous_start_code = self->start_code;
01641
01642   // Find first video PTS
01643   plm_demux_rewind(self);
01644   do {
01645     plm_packet_t *packet = plm_demux_decode(self);
01646     if (!packet) {
01647       break;
01648     }
01649     if (packet->type == type) {
01650       self->start_time = packet->pts;
01651     }
01652   } while (self->start_time == PLM_PACKET_INVALID_TS);
01653
01654   plm_demux_buffer_seek(self, previous_pos);
01655   self->start_code = previous_start_code;
01656   return self->start_time;
01657 }
01658
```

```
01659 double plm_demux_get_duration(plm_demux_t *self, int type) {
01660   size_t file_size = plm_buffer_get_size(self->buffer);
01661
01662   if (self->duration != PLM_PACKET_INVALID_TS && self->last_file_size == file_size) {
01663     return self->duration;
01664   }
01665
01666   size_t previous_pos = plm_buffer_tell(self->buffer);
01667   int previous_start_code = self->start_code;
01668
01669   // Find last video PTS. Start searching 64kb from the end and go further
01670   // back if needed.
01671   long start_range = 64 * 1024;
01672   long max_range = 4096 * 1024;
01673   for (long range = start_range; range <= max_range; range *= 2) {
01674     long seek_pos = file_size - range;
01675     if (seek_pos < 0) {
01676       seek_pos = 0;
01677       range = max_range; // Make sure to bail after this round
01678     }
01679     plm_demux_buffer_seek(self, seek_pos);
01680     self->current_packet.length = 0;
01681
01682     double last_pts = PLM_PACKET_INVALID_TS;
01683     plm_packet_t *packet = NULL;
01684     while ((packet = plm_demux_decode(self))) {
01685       if (packet->pts != PLM_PACKET_INVALID_TS && packet->type == type) {
01686         last_pts = packet->pts;
01687       }
01688     }
01689     if (last_pts != PLM_PACKET_INVALID_TS) {
01690       self->duration = last_pts - plm_demux_get_start_time(self, type);
01691       break;
01692     }
01693   }
01694
01695   plm_demux_buffer_seek(self, previous_pos);
01696   self->start_code = previous_start_code;
01697   self->last_file_size = file_size;
01698   return self->duration;
01699 }
01700
01701 plm_packet_t *plm_demux_seek(plm_demux_t *self, double seek_time, int type, int force_intra) {
01702   if (!plm_demux_has_headers(self)) {
01703     return NULL;
01704   }
01705
01706   // Using the current time, current byte position and the average bytes per
01707   // second for this file, try to jump to a byte position that hopefully has
01708   // packets containing timestamps within one second before to the desired
01709   // seek_time.
01710
01711   // If we hit close to the seek_time scan through all packets to find the
01712   // last one (just before the seek_time) containing an intra frame.
01713   // Otherwise we should at least be closer than before. Calculate the bytes
01714   // per second for the jumped range and jump again.
01715
01716   // The number of retries here is hard-limited to a generous amount. Usually
01717   // the correct range is found after 1--5 jumps, even for files with very
01718   // variable bitrates. If significantly more jumps are needed, there's
01719   // probably something wrong with the file and we just avoid getting into an
01720   // infinite loop. 32 retries should be enough for anybody.
01721
01722   double duration = plm_demux_get_duration(self, type);
01723   long file_size = plm_buffer_get_size(self->buffer);
01724   long byterate = file_size / duration;
01725
01726   double cur_time = self->last_decoded_pts;
01727   double scan_span = 1;
01728
01729   if (seek_time > duration) {
01730     seek_time = duration;
01731   } else if (seek_time < 0) {
01732     seek_time = 0;
01733   }
01734   seek_time += self->start_time;
01735
01736   for (int retry = 0; retry < 32; retry++) {
01737     int found_packet_with_pts = FALSE;
01738     int found_packet_in_range = FALSE;
01739     long last_valid_packet_start = -1;
01740     double first_packet_time = PLM_PACKET_INVALID_TS;
01741
01742     long cur_pos = plm_buffer_tell(self->buffer);
01743
01744     // Estimate byte offset and jump to it.
01745     long offset = (seek_time - cur_time - scan_span) * byterate;
```

```
01746     long seek_pos = cur_pos + offset;
01747     if (seek_pos < 0) {
01748       seek_pos = 0;
01749     } else if (seek_pos > file_size - 256) {
01750       seek_pos = file_size - 256;
01751     }
01752
01753     plm_demux_buffer_seek(self, seek_pos);
01754
01755     // Scan through all packets up to the seek_time to find the last packet
01756     // containing an intra frame.
01757     while (plm_buffer_find_start_code(self->buffer, type) != -1) {
01758       long packet_start = plm_buffer_tell(self->buffer);
01759       plm_packet_t *packet = plm_demux_decode_packet(self, type);
01760
01761       // Skip packet if it has no PTS
01762       if (!packet || packet->pts == PLM_PACKET_INVALID_TS) {
01763         continue;
01764       }
01765
01766       // Bail scanning through packets if we hit one that is outside
01767       // seek_time - scan_span.
01768       // We also adjust the cur_time and byterate values here so the next
01769       // iteration can be a bit more precise.
01770       if (packet->pts > seek_time || packet->pts < seek_time - scan_span) {
01771         found_packet_with_pts = TRUE;
01772         byterate = (seek_pos - cur_pos) / (packet->pts - cur_time);
01773         cur_time = packet->pts;
01774         break;
01775       }
01776
01777       // If we are still here, it means this packet is in close range to
01778       // the seek_time. If this is the first packet for this jump position
01779       // record the PTS. If we later have to back off, when there was no
01780       // intra frame in this range, we can lower the seek_time to not scan
01781       // this range again.
01782       if (!found_packet_in_range) {
01783         found_packet_in_range = TRUE;
01784         first_packet_time = packet->pts;
01785       }
01786
01787       // Check if this is an intra frame packet. If so, record the buffer
01788       // position of the start of this packet. We want to jump back to it
01789       // later, when we know it's the last intra frame before desired
01790       // seek time.
01791       if (force_intra) {
01792         for (size_t i = 0; i < packet->length - 6; i++) {
01793           // Find the START_PICTURE code
01794           if (packet->data[i] == 0x00 && packet->data[i + 1] == 0x00 && packet->data[i + 2] == 0x01 &&
01795               packet->data[i + 3] == 0x00) {
01796             // Bits 11--13 in the picture header contain the frame
01797             // type, where 1=Intra
01798             if ((packet->data[i + 5] & 0x38) == 8) {
01799               last_valid_packet_start = packet_start;
01800             }
01801             break;
01802           }
01803         }
01804       }
01805
01806       // If we don't want intra frames, just use the last PTS found.
01807       else {
01808         last_valid_packet_start = packet_start;
01809       }
01810     }
01811
01812     // If there was at least one intra frame in the range scanned above,
01813     // our search is over. Jump back to the packet and decode it again.
01814     if (last_valid_packet_start != -1) {
01815       plm_demux_buffer_seek(self, last_valid_packet_start);
01816       return plm_demux_decode_packet(self, type);
01817     }
01818
01819     // If we hit the right range, but still found no intra frame, we have
01820     // to increases the scan_span. This is done exponentially to also handle
01821     // video files with very few intra frames.
01822     else if (found_packet_in_range) {
01823       scan_span *= 2;
01824       seek_time = first_packet_time;
01825     }
01826
01827     // If we didn't find any packet with a PTS, it probably means we reached
01828     // the end of the file. Estimate byterate and cur_time accordingly.
01829     else if (!found_packet_with_pts) {
01830       byterate = (seek_pos - cur_pos) / (duration - cur_time);
01831       cur_time = duration;
01832     }
```

```
01833   }
01834
01835   return NULL;
01836 }
01837
01838 plm_packet_t *plm_demux_decode(plm_demux_t *self) {
01839   if (!plm_demux_has_headers(self)) {
01840     return NULL;
01841   }
01842
01843   if (self->current_packet.length) {
01844     size_t bits_till_next_packet = self->current_packet.length << 3;
01845     if (!plm_buffer_has(self->buffer, bits_till_next_packet)) {
01846       return NULL;
01847     }
01848     plm_buffer_skip(self->buffer, bits_till_next_packet);
01849     self->current_packet.length = 0;
01850   }
01851
01852   // Pending packet waiting for data?
01853   if (self->next_packet.length) {
01854     return plm_demux_get_packet(self);
01855   }
01856
01857   // Pending packet waiting for header?
01858   if (self->start_code != -1) {
01859     return plm_demux_decode_packet(self, self->start_code);
01860   }
01861
01862   do {
01863     self->start_code = plm_buffer_next_start_code(self->buffer);
01864     if (self->start_code == PLM_DEMUX_PACKET_VIDEO_1 || self->start_code == PLM_DEMUX_PACKET_PRIVATE
      ||
01865         (self->start_code >= PLM_DEMUX_PACKET_AUDIO_1 && self->start_code <=
      PLM_DEMUX_PACKET_AUDIO_4)) {
01866       return plm_demux_decode_packet(self, self->start_code);
01867     }
01868   } while (self->start_code != -1);
01869
01870   return NULL;
01871 }
01872
01873 double plm_demux_decode_time(plm_demux_t *self) {
01874   int64_t clock = plm_buffer_read(self->buffer, 3) << 30;
01875   plm_buffer_skip(self->buffer, 1);
01876   clock |= plm_buffer_read(self->buffer, 15) << 15;
01877   plm_buffer_skip(self->buffer, 1);
01878   clock |= plm_buffer_read(self->buffer, 15);
01879   plm_buffer_skip(self->buffer, 1);
01880   return (double)clock / 90000.0;
01881 }
01882
01883 plm_packet_t *plm_demux_decode_packet(plm_demux_t *self, int type) {
01884   if (!plm_buffer_has(self->buffer, 16 << 3)) {
01885     return NULL;
01886   }
01887
01888   self->start_code = -1;
01889
01890   self->next_packet.type = type;
01891   self->next_packet.length = plm_buffer_read(self->buffer, 16);
01892   self->next_packet.length -= plm_buffer_skip_bytes(self->buffer, 0xff); // stuffing
01893
01894   // skip P-STD
01895   if (plm_buffer_read(self->buffer, 2) == 0x01) {
01896     plm_buffer_skip(self->buffer, 16);
01897     self->next_packet.length -= 2;
01898   }
01899
01900   int pts_dts_marker = plm_buffer_read(self->buffer, 2);
01901   if (pts_dts_marker == 0x03) {
01902     self->next_packet.pts = plm_demux_decode_time(self);
01903     self->last_decoded_pts = self->next_packet.pts;
01904     plm_buffer_skip(self->buffer, 40); // skip dts
01905     self->next_packet.length -= 10;
01906   } else if (pts_dts_marker == 0x02) {
01907     self->next_packet.pts = plm_demux_decode_time(self);
01908     self->last_decoded_pts = self->next_packet.pts;
01909     self->next_packet.length -= 5;
01910   } else if (pts_dts_marker == 0x00) {
01911     self->next_packet.pts = PLM_PACKET_INVALID_TS;
01912     plm_buffer_skip(self->buffer, 4);
01913     self->next_packet.length -= 1;
01914   } else {
01915     return NULL; // invalid
01916   }
01917
```

```
01918   return plm_demux_get_packet(self);
01919 }
01920
01921 plm_packet_t *plm_demux_get_packet(plm_demux_t *self) {
01922   if (!plm_buffer_has(self->buffer, self->next_packet.length << 3)) {
01923     return NULL;
01924   }
01925
01926   self->current_packet.data = self->buffer->bytes + (self->buffer->bit_index >> 3);
01927   self->current_packet.length = self->next_packet.length;
01928   self->current_packet.type = self->next_packet.type;
01929   self->current_packet.pts = self->next_packet.pts;
01930
01931   self->next_packet.length = 0;
01932   return &self->current_packet;
01933 }
01934
01935 // -----------------------------------------------------------------------------
01936 // plm_video implementation
01937
01938 // Inspired by Java MPEG-1 Video Decoder and Player by Zoltan Korandi
01939 // https://sourceforge.net/projects/javampeg1video/
01940
01941 static const int PLM_VIDEO_PICTURE_TYPE_INTRA = 1;
01942 static const int PLM_VIDEO_PICTURE_TYPE_PREDICTIVE = 2;
01943 static const int PLM_VIDEO_PICTURE_TYPE_B = 3;
01944
01945 static const int PLM_START_SEQUENCE = 0xB3;
01946 static const int PLM_START_SLICE_FIRST = 0x01;
01947 static const int PLM_START_SLICE_LAST = 0xAF;
01948 static const int PLM_START_PICTURE = 0x00;
01949 static const int PLM_START_EXTENSION = 0xB5;
01950 static const int PLM_START_USER_DATA = 0xB2;
01951
01952 #define PLM_START_IS_SLICE(c) (c >= PLM_START_SLICE_FIRST && c <= PLM_START_SLICE_LAST)
01953
01954 static const double PLM_VIDEO_PICTURE_RATE[] = {0.000,  23.976, 24.000, 25.000, 29.970, 30.000,
      50.000, 59.940,
01955                                                 60.000, 0.000,  0.000,  0.000,  0.000,  0.000,  0.000,
      0.000};
01956
01957 static const uint8_t PLM_VIDEO_ZIG_ZAG[] = {0,  1,  8,  16, 9,  2,  3,  10, 17, 24, 32, 25, 18, 11, 4,
      5,
01958                                              12, 19, 26, 33, 40, 48, 41, 34, 27, 20, 13, 6,  7,  14,
      21, 28,
01959                                              35, 42, 49, 56, 57, 50, 43, 36, 29, 22, 15, 23, 30, 37,
      44, 51,
01960                                              58, 59, 52, 45, 38, 31, 39, 46, 53, 60, 61, 54, 47, 55,
      62, 63};
01961
01962 static const uint8_t PLM_VIDEO_INTRA_QUANT_MATRIX[] = {8,  16, 19, 22, 26, 27, 29, 34, 16, 16, 22, 24,
      27, 29, 34, 37,
01963                                                       19, 22, 26, 27, 29, 34, 34, 38, 22, 22, 26, 27,
      29, 34, 37, 40,
01964                                                       22, 26, 27, 29, 32, 35, 40, 48, 26, 27, 29, 32,
      35, 40, 48, 58,
01965                                                       26, 27, 29, 34, 38, 46, 56, 69, 27, 29, 35, 38,
      46, 56, 69, 83};
01966
01967 static const uint8_t PLM_VIDEO_NON_INTRA_QUANT_MATRIX[] = {
01968     16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
01969     16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16,
01970     16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16, 16};
01971
01972 static const uint8_t PLM_VIDEO_PREMULTIPLIER_MATRIX[] = {32, 44, 42, 38, 32, 25, 17, 9,  44, 62, 58,
      52, 44, 35, 24, 12,
01973                                                         42, 58, 55, 49, 42, 33, 23, 12, 38, 52, 49,
      44, 38, 30, 20, 10,
01974                                                         32, 44, 42, 38, 32, 25, 17, 9,  25, 35, 33,
      30, 25, 20, 14, 7,
01975                                                         17, 24, 23, 20, 17, 14, 9,  5,  9,  12, 12,
      10, 9,  7,  5,  2};
01976
01977 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT[] = {
01978     {1 << 1, 0},  {0, 1},       //   0: x
01979     {2 << 1, 0},  {3 << 1, 0},  //   1: 0x
01980     {4 << 1, 0},  {5 << 1, 0},  //   2: 00x
01981     {0, 3},       {0, 2},       //   3: 01x
01982     {6 << 1, 0},  {7 << 1, 0},  //   4: 000x
01983     {0, 5},       {0, 4},       //   5: 001x
01984     {8 << 1, 0},  {9 << 1, 0},  //   6: 0000x
01985     {0, 7},       {0, 6},       //   7: 0001x
01986     {10 << 1, 0}, {11 << 1, 0}, //   8: 0000 0x
01987     {12 << 1, 0}, {13 << 1, 0}, //   9: 0000 1x
01988     {14 << 1, 0}, {15 << 1, 0}, //  10: 0000 00x
01989     {16 << 1, 0}, {17 << 1, 0}, //  11: 0000 01x
01990     {18 << 1, 0}, {19 << 1, 0}, //  12: 0000 10x
```

```
01991     {0, 9},       {0, 8},       //  13: 0000 11x
01992     {-1, 0},      {20 « 1, 0}, //  14: 0000 000x
01993     {-1, 0},      {21 « 1, 0}, //  15: 0000 001x
01994     {22 « 1, 0}, {23 « 1, 0}, //  16: 0000 010x
01995     {0, 15},      {0, 14},      //  17: 0000 011x
01996     {0, 13},      {0, 12},      //  18: 0000 100x
01997     {0, 11},      {0, 10},      //  19: 0000 101x
01998     {24 « 1, 0}, {25 « 1, 0}, //  20: 0000 0001x
01999     {26 « 1, 0}, {27 « 1, 0}, //  21: 0000 0011x
02000     {28 « 1, 0}, {29 « 1, 0}, //  22: 0000 0100x
02001     {30 « 1, 0}, {31 « 1, 0}, //  23: 0000 0101x
02002     {32 « 1, 0}, {-1, 0},      //  24: 0000 0001 0x
02003     {-1, 0},      {33 « 1, 0}, //  25: 0000 0001 1x
02004     {34 « 1, 0}, {35 « 1, 0}, //  26: 0000 0011 0x
02005     {36 « 1, 0}, {37 « 1, 0}, //  27: 0000 0011 1x
02006     {38 « 1, 0}, {39 « 1, 0}, //  28: 0000 0100 0x
02007     {0, 21},      {0, 20},      //  29: 0000 0100 1x
02008     {0, 19},      {0, 18},      //  30: 0000 0101 0x
02009     {0, 17},      {0, 16},      //  31: 0000 0101 1x
02010     {0, 35},      {-1, 0},      //  32: 0000 0001 00x
02011     {-1, 0},      {0, 34},      //  33: 0000 0001 11x
02012     {0, 33},      {0, 32},      //  34: 0000 0011 00x
02013     {0, 31},      {0, 30},      //  35: 0000 0011 01x
02014     {0, 29},      {0, 28},      //  36: 0000 0011 10x
02015     {0, 27},      {0, 26},      //  37: 0000 0011 11x
02016     {0, 25},      {0, 24},      //  38: 0000 0100 00x
02017     {0, 23},      {0, 22},      //  39: 0000 0100 01x
02018 };
02019
02020 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_TYPE_INTRA[] = {
02021     {1 « 1, 0},
02022     {0, 0x01}, //   0: x
02023     {-1, 0},
02024     {0, 0x11}, //   1: 0x
02025 };
02026
02027 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_TYPE_PREDICTIVE[] = {
02028     {1 « 1, 0}, {0, 0x0a},   //   0: x
02029     {2 « 1, 0}, {0, 0x02},   //   1: 0x
02030     {3 « 1, 0}, {0, 0x08},   //   2: 00x
02031     {4 « 1, 0}, {5 « 1, 0}, //   3: 000x
02032     {6 « 1, 0}, {0, 0x12},   //   4: 0000x
02033     {0, 0x1a},   {0, 0x01},   //   5: 0001x
02034     {-1, 0},     {0, 0x11},   //   6: 0000 0x
02035 };
02036
02037 static const plm_vlc_t PLM_VIDEO_MACROBLOCK_TYPE_B[] = {
02038     {1 « 1, 0}, {2 « 1, 0},  //   0: x
02039     {3 « 1, 0}, {4 « 1, 0},  //   1: 0x
02040     {0, 0x0c},   {0, 0x0e},    //   2: 1x
02041     {5 « 1, 0}, {6 « 1, 0},  //   3: 00x
02042     {0, 0x04},   {0, 0x06},    //   4: 01x
02043     {7 « 1, 0}, {8 « 1, 0},  //   5: 000x
02044     {0, 0x08},   {0, 0x0a},    //   6: 001x
02045     {9 « 1, 0}, {10 « 1, 0}, //   7: 0000x
02046     {0, 0x1e},   {0, 0x01},    //   8: 0001x
02047     {-1, 0},     {0, 0x11},    //   9: 0000 0x
02048     {0, 0x16},   {0, 0x1a},    //  10: 0000 1x
02049 };
02050
02051 static const plm_vlc_t *PLM_VIDEO_MACROBLOCK_TYPE[] = {
02052     NULL, PLM_VIDEO_MACROBLOCK_TYPE_INTRA, PLM_VIDEO_MACROBLOCK_TYPE_PREDICTIVE,
      PLM_VIDEO_MACROBLOCK_TYPE_B};
02053
02054 static const plm_vlc_t PLM_VIDEO_CODE_BLOCK_PATTERN[] = {
02055     {1 « 1, 0},  {2 « 1, 0},  //   0: x
02056     {3 « 1, 0},  {4 « 1, 0},  //   1: 0x
02057     {5 « 1, 0},  {6 « 1, 0},  //   2: 1x
02058     {7 « 1, 0},  {8 « 1, 0},  //   3: 00x
02059     {9 « 1, 0},  {10 « 1, 0}, //   4: 01x
02060     {11 « 1, 0}, {12 « 1, 0}, //   5: 10x
02061     {13 « 1, 0}, {0, 60},      //   6: 11x
02062     {14 « 1, 0}, {15 « 1, 0}, //   7: 000x
02063     {16 « 1, 0}, {17 « 1, 0}, //   8: 001x
02064     {18 « 1, 0}, {19 « 1, 0}, //   9: 010x
02065     {20 « 1, 0}, {21 « 1, 0}, //  10: 011x
02066     {22 « 1, 0}, {23 « 1, 0}, //  11: 100x
02067     {0, 32},      {0, 16},      //  12: 101x
02068     {0, 8},       {0, 4},       //  13: 110x
02069     {24 « 1, 0}, {25 « 1, 0}, //  14: 0000x
02070     {26 « 1, 0}, {27 « 1, 0}, //  15: 0001x
02071     {28 « 1, 0}, {29 « 1, 0}, //  16: 0010x
02072     {30 « 1, 0}, {31 « 1, 0}, //  17: 0011x
02073     {0, 62},      {0, 2},       //  18: 0100x
02074     {0, 61},      {0, 1},       //  19: 0101x
02075     {0, 56},      {0, 52},      //  20: 0110x
02076     {0, 44},      {0, 28},      //  21: 0111x
```

```
02077     {0, 40},      {0, 20},      //  22: 1000x
02078     {0, 48},      {0, 12},      //  23: 1001x
02079     {32 « 1, 0}, {33 « 1, 0}, //  24: 0000 0x
02080     {34 « 1, 0}, {35 « 1, 0}, //  25: 0000 1x
02081     {36 « 1, 0}, {37 « 1, 0}, //  26: 0001 0x
02082     {38 « 1, 0}, {39 « 1, 0}, //  27: 0001 1x
02083     {40 « 1, 0}, {41 « 1, 0}, //  28: 0010 0x
02084     {42 « 1, 0}, {43 « 1, 0}, //  29: 0010 1x
02085     {0, 63},      {0, 3},       //  30: 0011 0x
02086     {0, 36},      {0, 24},      //  31: 0011 1x
02087     {44 « 1, 0}, {45 « 1, 0}, //  32: 0000 00x
02088     {46 « 1, 0}, {47 « 1, 0}, //  33: 0000 01x
02089     {48 « 1, 0}, {49 « 1, 0}, //  34: 0000 10x
02090     {50 « 1, 0}, {51 « 1, 0}, //  35: 0000 11x
02091     {52 « 1, 0}, {53 « 1, 0}, //  36: 0001 00x
02092     {54 « 1, 0}, {55 « 1, 0}, //  37: 0001 01x
02093     {56 « 1, 0}, {57 « 1, 0}, //  38: 0001 10x
02094     {58 « 1, 0}, {59 « 1, 0}, //  39: 0001 11x
02095     {0, 34},      {0, 18},      //  40: 0010 00x
02096     {0, 10},      {0, 6},       //  41: 0010 01x
02097     {0, 33},      {0, 17},      //  42: 0010 10x
02098     {0, 9},       {0, 5},       //  43: 0010 11x
02099     {-1, 0},      {60 « 1, 0}, //  44: 0000 000x
02100     {61 « 1, 0}, {62 « 1, 0}, //  45: 0000 001x
02101     {0, 58},      {0, 54},      //  46: 0000 010x
02102     {0, 46},      {0, 30},      //  47: 0000 011x
02103     {0, 57},      {0, 53},      //  48: 0000 100x
02104     {0, 45},      {0, 29},      //  49: 0000 101x
02105     {0, 38},      {0, 26},      //  50: 0000 110x
02106     {0, 37},      {0, 25},      //  51: 0000 111x
02107     {0, 43},      {0, 23},      //  52: 0001 000x
02108     {0, 51},      {0, 15},      //  53: 0001 001x
02109     {0, 42},      {0, 22},      //  54: 0001 010x
02110     {0, 50},      {0, 14},      //  55: 0001 011x
02111     {0, 41},      {0, 21},      //  56: 0001 100x
02112     {0, 49},      {0, 13},      //  57: 0001 101x
02113     {0, 35},      {0, 19},      //  58: 0001 110x
02114     {0, 11},      {0, 7},       //  59: 0001 111x
02115     {0, 39},      {0, 27},      //  60: 0000 0001x
02116     {0, 59},      {0, 55},      //  61: 0000 0010x
02117     {0, 47},      {0, 31},      //  62: 0000 0011x
02118 };
02119
02120 static const plm_vlc_t PLM_VIDEO_MOTION[] = {
02121     {1 « 1, 0},  {0, 0},       //   0: x
02122     {2 « 1, 0},  {3 « 1, 0},  //   1: 0x
02123     {4 « 1, 0},  {5 « 1, 0},  //   2: 00x
02124     {0, 1},       {0, -1},      //   3: 01x
02125     {6 « 1, 0},  {7 « 1, 0},  //   4: 000x
02126     {0, 2},       {0, -2},      //   5: 001x
02127     {8 « 1, 0},  {9 « 1, 0},  //   6: 0000x
02128     {0, 3},       {0, -3},      //   7: 0001x
02129     {10 « 1, 0}, {11 « 1, 0}, //   8: 0000 0x
02130     {12 « 1, 0}, {13 « 1, 0}, //   9: 0000 1x
02131     {-1, 0},      {14 « 1, 0}, //  10: 0000 00x
02132     {15 « 1, 0}, {16 « 1, 0}, //  11: 0000 01x
02133     {17 « 1, 0}, {18 « 1, 0}, //  12: 0000 10x
02134     {0, 4},       {0, -4},      //  13: 0000 11x
02135     {-1, 0},      {19 « 1, 0}, //  14: 0000 001x
02136     {20 « 1, 0}, {21 « 1, 0}, //  15: 0000 010x
02137     {0, 7},       {0, -7},      //  16: 0000 011x
02138     {0, 6},       {0, -6},      //  17: 0000 100x
02139     {0, 5},       {0, -5},      //  18: 0000 101x
02140     {22 « 1, 0}, {23 « 1, 0}, //  19: 0000 0011x
02141     {24 « 1, 0}, {25 « 1, 0}, //  20: 0000 0100x
02142     {26 « 1, 0}, {27 « 1, 0}, //  21: 0000 0101x
02143     {28 « 1, 0}, {29 « 1, 0}, //  22: 0000 0011 0x
02144     {30 « 1, 0}, {31 « 1, 0}, //  23: 0000 0011 1x
02145     {32 « 1, 0}, {33 « 1, 0}, //  24: 0000 0100 0x
02146     {0, 10},      {0, -10},     //  25: 0000 0100 1x
02147     {0, 9},       {0, -9},      //  26: 0000 0101 0x
02148     {0, 8},       {0, -8},      //  27: 0000 0101 1x
02149     {0, 16},      {0, -16},     //  28: 0000 0011 00x
02150     {0, 15},      {0, -15},     //  29: 0000 0011 01x
02151     {0, 14},      {0, -14},     //  30: 0000 0011 10x
02152     {0, 13},      {0, -13},     //  31: 0000 0011 11x
02153     {0, 12},      {0, -12},     //  32: 0000 0100 00x
02154     {0, 11},      {0, -11},     //  33: 0000 0100 01x
02155 };
02156
02157 static const plm_vlc_t PLM_VIDEO_DCT_SIZE_LUMINANCE[] = {
02158     {1 « 1, 0}, {2 « 1, 0}, //   0: x
02159     {0, 1},       {0, 2},       //   1: 0x
02160     {3 « 1, 0}, {4 « 1, 0}, //   2: 1x
02161     {0, 0},       {0, 3},       //   3: 10x
02162     {0, 4},       {5 « 1, 0}, //   4: 11x
02163     {0, 5},       {6 « 1, 0}, //   5: 111x
```

```
02164     {0, 6},      {7 « 1, 0}, //   6: 1111x
02165     {0, 7},      {8 « 1, 0}, //   7: 1111 1x
02166     {0, 8},      {-1, 0},     //   8: 1111 11x
02167 };
02168
02169 static const plm_vlc_t PLM_VIDEO_DCT_SIZE_CHROMINANCE[] = {
02170     {1 « 1, 0}, {2 « 1, 0}, //   0: x
02171     {0, 0},      {0, 1},      //   1: 0x
02172     {0, 2},      {3 « 1, 0}, //   2: 1x
02173     {0, 3},      {4 « 1, 0}, //   3: 11x
02174     {0, 4},      {5 « 1, 0}, //   4: 111x
02175     {0, 5},      {6 « 1, 0}, //   5: 1111x
02176     {0, 6},      {7 « 1, 0}, //   6: 1111 1x
02177     {0, 7},      {8 « 1, 0}, //   7: 1111 11x
02178     {0, 8},      {-1, 0},     //   8: 1111 111x
02179 };
02180
02181 static const plm_vlc_t *PLM_VIDEO_DCT_SIZE[] = {PLM_VIDEO_DCT_SIZE_LUMINANCE,
      PLM_VIDEO_DCT_SIZE_CHROMINANCE,
02182                                                  PLM_VIDEO_DCT_SIZE_CHROMINANCE};
02183
02184 //  dct_coeff bitmap:
02185 //    0xff00  run
02186 //    0x00ff  level
02187
02188 //  Decoded values are unsigned. Sign bit follows in the stream.
02189
02190 static const plm_vlc_uint_t PLM_VIDEO_DCT_COEFF[] = {
02191     {1 « 1, 0},   {0, 0x0001},   //   0: x
02192     {2 « 1, 0},   {3 « 1, 0},   //   1: 0x
02193     {4 « 1, 0},   {5 « 1, 0},   //   2: 00x
02194     {6 « 1, 0},   {0, 0x0101},   //   3: 01x
02195     {7 « 1, 0},   {8 « 1, 0},   //   4: 000x
02196     {9 « 1, 0},   {10 « 1, 0},  //   5: 001x
02197     {0, 0x0002},   {0, 0x0201},   //   6: 010x
02198     {11 « 1, 0},  {12 « 1, 0},  //   7: 0000x
02199     {13 « 1, 0},  {14 « 1, 0},  //   8: 0001x
02200     {15 « 1, 0},  {0, 0x0003},   //   9: 0010x
02201     {0, 0x0401},   {0, 0x0301},   //  10: 0011x
02202     {16 « 1, 0},  {0, 0xffff},   //  11: 0000 0x
02203     {17 « 1, 0},  {18 « 1, 0},  //  12: 0000 1x
02204     {0, 0x0701},   {0, 0x0601},   //  13: 0001 0x
02205     {0, 0x0102},   {0, 0x0501},   //  14: 0001 1x
02206     {19 « 1, 0},  {20 « 1, 0},  //  15: 0010 0x
02207     {21 « 1, 0},  {22 « 1, 0},  //  16: 0000 00x
02208     {0, 0x0202},   {0, 0x0901},   //  17: 0000 10x
02209     {0, 0x0004},   {0, 0x0801},   //  18: 0000 11x
02210     {23 « 1, 0},  {24 « 1, 0},  //  19: 0010 00x
02211     {25 « 1, 0},  {26 « 1, 0},  //  20: 0010 01x
02212     {27 « 1, 0},  {28 « 1, 0},  //  21: 0000 000x
02213     {29 « 1, 0},  {30 « 1, 0},  //  22: 0000 001x
02214     {0, 0x0d01},   {0, 0x0006},   //  23: 0010 000x
02215     {0, 0x0c01},   {0, 0x0b01},   //  24: 0010 001x
02216     {0, 0x0302},   {0, 0x0103},   //  25: 0010 010x
02217     {0, 0x0005},   {0, 0x0a01},   //  26: 0010 011x
02218     {31 « 1, 0},  {32 « 1, 0},  //  27: 0000 0000x
02219     {33 « 1, 0},  {34 « 1, 0},  //  28: 0000 0001x
02220     {35 « 1, 0},  {36 « 1, 0},  //  29: 0000 0010x
02221     {37 « 1, 0},  {38 « 1, 0},  //  30: 0000 0011x
02222     {39 « 1, 0},  {40 « 1, 0},  //  31: 0000 0000 0x
02223     {41 « 1, 0},  {42 « 1, 0},  //  32: 0000 0000 1x
02224     {43 « 1, 0},  {44 « 1, 0},  //  33: 0000 0001 0x
02225     {45 « 1, 0},  {46 « 1, 0},  //  34: 0000 0001 1x
02226     {0, 0x1001},   {0, 0x0502},   //  35: 0000 0010 0x
02227     {0, 0x0007},   {0, 0x0203},   //  36: 0000 0010 1x
02228     {0, 0x0104},   {0, 0x0f01},   //  37: 0000 0011 0x
02229     {0, 0x0e01},   {0, 0x0402},   //  38: 0000 0011 1x
02230     {47 « 1, 0},  {48 « 1, 0},  //  39: 0000 0000 00x
02231     {49 « 1, 0},  {50 « 1, 0},  //  40: 0000 0000 01x
02232     {51 « 1, 0},  {52 « 1, 0},  //  41: 0000 0000 10x
02233     {53 « 1, 0},  {54 « 1, 0},  //  42: 0000 0000 11x
02234     {55 « 1, 0},  {56 « 1, 0},  //  43: 0000 0001 00x
02235     {57 « 1, 0},  {58 « 1, 0},  //  44: 0000 0001 01x
02236     {59 « 1, 0},  {60 « 1, 0},  //  45: 0000 0001 10x
02237     {61 « 1, 0},  {62 « 1, 0},  //  46: 0000 0001 11x
02238     {-1, 0},       {63 « 1, 0},  //  47: 0000 0000 000x
02239     {64 « 1, 0},  {65 « 1, 0},  //  48: 0000 0000 001x
02240     {66 « 1, 0},  {67 « 1, 0},  //  49: 0000 0000 010x
02241     {68 « 1, 0},  {69 « 1, 0},  //  50: 0000 0000 011x
02242     {70 « 1, 0},  {71 « 1, 0},  //  51: 0000 0000 100x
02243     {72 « 1, 0},  {73 « 1, 0},  //  52: 0000 0000 101x
02244     {74 « 1, 0},  {75 « 1, 0},  //  53: 0000 0000 110x
02245     {76 « 1, 0},  {77 « 1, 0},  //  54: 0000 0000 111x
02246     {0, 0x000b},   {0, 0x0802},   //  55: 0000 0001 000x
02247     {0, 0x0403},   {0, 0x000a},   //  56: 0000 0001 001x
02248     {0, 0x0204},   {0, 0x0702},   //  57: 0000 0001 010x
02249     {0, 0x1501},   {0, 0x1401},   //  58: 0000 0001 011x
```

```
02250     {0, 0x0009},   {0, 0x1301},   //  59: 0000 0001 100x
02251     {0, 0x1201},   {0, 0x0105},   //  60: 0000 0001 101x
02252     {0, 0x0303},   {0, 0x0008},   //  61: 0000 0001 110x
02253     {0, 0x0602},   {0, 0x1101},   //  62: 0000 0001 111x
02254     {78 « 1, 0},  {79 « 1, 0},  //  63: 0000 0000 0001x
02255     {80 « 1, 0},  {81 « 1, 0},  //  64: 0000 0000 0010x
02256     {82 « 1, 0},  {83 « 1, 0},  //  65: 0000 0000 0011x
02257     {84 « 1, 0},  {85 « 1, 0},  //  66: 0000 0000 0100x
02258     {86 « 1, 0},  {87 « 1, 0},  //  67: 0000 0000 0101x
02259     {88 « 1, 0},  {89 « 1, 0},  //  68: 0000 0000 0110x
02260     {90 « 1, 0},  {91 « 1, 0},  //  69: 0000 0000 0111x
02261     {0, 0x0a02},   {0, 0x0902},   //  70: 0000 0000 1000x
02262     {0, 0x0503},   {0, 0x0304},   //  71: 0000 0000 1001x
02263     {0, 0x0205},   {0, 0x0107},   //  72: 0000 0000 1010x
02264     {0, 0x0106},   {0, 0x000f},   //  73: 0000 0000 1011x
02265     {0, 0x000e},   {0, 0x000d},   //  74: 0000 0000 1100x
02266     {0, 0x000c},   {0, 0x1a01},   //  75: 0000 0000 1101x
02267     {0, 0x1901},   {0, 0x1801},   //  76: 0000 0000 1110x
02268     {0, 0x1701},   {0, 0x1601},   //  77: 0000 0000 1111x
02269     {92 « 1, 0},  {93 « 1, 0},  //  78: 0000 0000 0001 0x
02270     {94 « 1, 0},  {95 « 1, 0},  //  79: 0000 0000 0001 1x
02271     {96 « 1, 0},  {97 « 1, 0},  //  80: 0000 0000 0010 0x
02272     {98 « 1, 0},  {99 « 1, 0},  //  81: 0000 0000 0010 1x
02273     {100 « 1, 0}, {101 « 1, 0}, //  82: 0000 0000 0011 0x
02274     {102 « 1, 0}, {103 « 1, 0}, //  83: 0000 0000 0011 1x
02275     {0, 0x001f},   {0, 0x001e},   //  84: 0000 0000 0100 0x
02276     {0, 0x001d},   {0, 0x001c},   //  85: 0000 0000 0100 1x
02277     {0, 0x001b},   {0, 0x001a},   //  86: 0000 0000 0101 0x
02278     {0, 0x0019},   {0, 0x0018},   //  87: 0000 0000 0101 1x
02279     {0, 0x0017},   {0, 0x0016},   //  88: 0000 0000 0110 0x
02280     {0, 0x0015},   {0, 0x0014},   //  89: 0000 0000 0110 1x
02281     {0, 0x0013},   {0, 0x0012},   //  90: 0000 0000 0111 0x
02282     {0, 0x0011},   {0, 0x0010},   //  91: 0000 0000 0111 1x
02283     {104 « 1, 0}, {105 « 1, 0}, //  92: 0000 0000 0001 00x
02284     {106 « 1, 0}, {107 « 1, 0}, //  93: 0000 0000 0001 01x
02285     {108 « 1, 0}, {109 « 1, 0}, //  94: 0000 0000 0001 10x
02286     {110 « 1, 0}, {111 « 1, 0}, //  95: 0000 0000 0001 11x
02287     {0, 0x0028},   {0, 0x0027},   //  96: 0000 0000 0010 00x
02288     {0, 0x0026},   {0, 0x0025},   //  97: 0000 0000 0010 01x
02289     {0, 0x0024},   {0, 0x0023},   //  98: 0000 0000 0010 10x
02290     {0, 0x0022},   {0, 0x0021},   //  99: 0000 0000 0010 11x
02291     {0, 0x0020},   {0, 0x010e},   // 100: 0000 0000 0011 00x
02292     {0, 0x010d},   {0, 0x010c},   // 101: 0000 0000 0011 01x
02293     {0, 0x010b},   {0, 0x010a},   // 102: 0000 0000 0011 10x
02294     {0, 0x0109},   {0, 0x0108},   // 103: 0000 0000 0011 11x
02295     {0, 0x0112},   {0, 0x0111},   // 104: 0000 0000 0001 000x
02296     {0, 0x0110},   {0, 0x010f},   // 105: 0000 0000 0001 001x
02297     {0, 0x0603},   {0, 0x1002},   // 106: 0000 0000 0001 010x
02298     {0, 0x0f02},   {0, 0x0e02},   // 107: 0000 0000 0001 011x
02299     {0, 0x0d02},   {0, 0x0c02},   // 108: 0000 0000 0001 100x
02300     {0, 0x0b02},   {0, 0x1f01},   // 109: 0000 0000 0001 101x
02301     {0, 0x1e01},   {0, 0x1d01},   // 110: 0000 0000 0001 110x
02302     {0, 0x1c01},   {0, 0x1b01},   // 111: 0000 0000 0001 111x
02303 };
02304
02305 typedef struct {
02306   int full_px;
02307   int is_set;
02308   int r_size;
02309   int h;
02310   int v;
02311 } plm_video_motion_t;
02312
02313 struct plm_video_t {
02314   double framerate;
02315   double time;
02316   int frames_decoded;
02317   int width;
02318   int height;
02319   int mb_width;
02320   int mb_height;
02321   int mb_size;
02322
02323   int luma_width;
02324   int luma_height;
02325
02326   int chroma_width;
02327   int chroma_height;
02328
02329   int start_code;
02330   int picture_type;
02331
02332   plm_video_motion_t motion_forward;
02333   plm_video_motion_t motion_backward;
02334
02335   int has_sequence_header;
02336
```

```
02337   int quantizer_scale;
02338   int slice_begin;
02339   int macroblock_address;
02340
02341   int mb_row;
02342   int mb_col;
02343
02344   int macroblock_type;
02345   int macroblock_intra;
02346
02347   int dc_predictor[3];
02348
02349   plm_buffer_t *buffer;
02350   int destroy_buffer_when_done;
02351
02352   plm_frame_t frame_current;
02353   plm_frame_t frame_forward;
02354   plm_frame_t frame_backward;
02355
02356   uint8_t *frames_data;
02357
02358   int block_data[64];
02359   uint8_t intra_quant_matrix[64];
02360   uint8_t non_intra_quant_matrix[64];
02361
02362   int has_reference_frame;
02363   int assume_no_b_frames;
02364 };
02365
02366 static inline uint8_t plm_clamp(int n) {
02367   if (n > 255) {
02368     n = 255;
02369   } else if (n < 0) {
02370     n = 0;
02371   }
02372   return n;
02373 }
02374
02375 int plm_video_decode_sequence_header(plm_video_t *self);
02376 void plm_video_init_frame(plm_video_t *self, plm_frame_t *frame, uint8_t *base);
02377 void plm_video_decode_picture(plm_video_t *self);
02378 void plm_video_decode_slice(plm_video_t *self, int slice);
02379 void plm_video_decode_macroblock(plm_video_t *self);
02380 void plm_video_decode_motion_vectors(plm_video_t *self);
02381 int plm_video_decode_motion_vector(plm_video_t *self, int r_size, int motion);
02382 void plm_video_predict_macroblock(plm_video_t *self);
02383 void plm_video_copy_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v);
02384 void plm_video_interpolate_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v);
02385 void plm_video_process_macroblock(plm_video_t *self, uint8_t *s, uint8_t *d, int mh, int mb, int bs,
      int interp);
02386 void plm_video_decode_block(plm_video_t *self, int block);
02387 void plm_video_idct(int *block);
02388
02389 plm_video_t *plm_video_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done) {
02390   plm_video_t *self = (plm_video_t *)PLM_MALLOC(sizeof(plm_video_t));
02391   memset(self, 0, sizeof(plm_video_t));
02392
02393   self->buffer = buffer;
02394   self->destroy_buffer_when_done = destroy_when_done;
02395
02396   // Attempt to decode the sequence header
02397   self->start_code = plm_buffer_find_start_code(self->buffer, PLM_START_SEQUENCE);
02398   if (self->start_code != -1) {
02399     plm_video_decode_sequence_header(self);
02400   }
02401   return self;
02402 }
02403
02404 void plm_video_destroy(plm_video_t *self) {
02405   if (self->destroy_buffer_when_done) {
02406     plm_buffer_destroy(self->buffer);
02407   }
02408
02409   if (self->has_sequence_header) {
02410     PLM_FREE(self->frames_data);
02411   }
02412
02413   PLM_FREE(self);
02414 }
02415
02416 double plm_video_get_framerate(plm_video_t *self) { return plm_video_has_header(self) ?
      self->framerate : 0; }
02417
02418 int plm_video_get_width(plm_video_t *self) { return plm_video_has_header(self) ? self->width : 0; }
02419
02420 int plm_video_get_height(plm_video_t *self) { return plm_video_has_header(self) ? self->height : 0; }
02421
```

```
02422 void plm_video_set_no_delay(plm_video_t *self, int no_delay) { self->assume_no_b_frames = no_delay; }
02423
02424 double plm_video_get_time(plm_video_t *self) { return self->time; }
02425
02426 void plm_video_set_time(plm_video_t *self, double time) {
02427   self->frames_decoded = self->framerate * time;
02428   self->time = time;
02429 }
02430
02431 void plm_video_rewind(plm_video_t *self) {
02432   plm_buffer_rewind(self->buffer);
02433   self->time = 0;
02434   self->frames_decoded = 0;
02435   self->has_reference_frame = FALSE;
02436   self->start_code = -1;
02437 }
02438
02439 int plm_video_has_ended(plm_video_t *self) { return plm_buffer_has_ended(self->buffer); }
02440
02441 plm_frame_t *plm_video_decode(plm_video_t *self) {
02442   if (!plm_video_has_header(self)) {
02443     return NULL;
02444   }
02445
02446   plm_frame_t *frame = NULL;
02447   do {
02448     if (self->start_code != PLM_START_PICTURE) {
02449       self->start_code = plm_buffer_find_start_code(self->buffer, PLM_START_PICTURE);
02450
02451       if (self->start_code == -1) {
02452         // If we reached the end of the file and the previously decoded
02453         // frame was a reference frame, we still have to return it.
02454         if (self->has_reference_frame && !self->assume_no_b_frames &&
    plm_buffer_has_ended(self->buffer) &&
02455             (self->picture_type == PLM_VIDEO_PICTURE_TYPE_INTRA ||
02456              self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE)) {
02457           self->has_reference_frame = FALSE;
02458           frame = &self->frame_backward;
02459           break;
02460         }
02461
02462         return NULL;
02463       }
02464     }
02465
02466     // Make sure we have a full picture in the buffer before attempting to
02467     // decode it. Sadly, this can only be done by seeking for the start code
02468     // of the next picture. Also, if we didn't find the start code for the
02469     // next picture, but the source has ended, we assume that this last
02470     // picture is in the buffer.
02471     if (plm_buffer_has_start_code(self->buffer, PLM_START_PICTURE) == -1 &&
    !plm_buffer_has_ended(self->buffer)) {
02472       return NULL;
02473     }
02474     plm_buffer_discard_read_bytes(self->buffer);
02475
02476     plm_video_decode_picture(self);
02477
02478     if (self->assume_no_b_frames) {
02479       frame = &self->frame_backward;
02480     } else if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_B) {
02481       frame = &self->frame_current;
02482     } else if (self->has_reference_frame) {
02483       frame = &self->frame_forward;
02484     } else {
02485       self->has_reference_frame = TRUE;
02486     }
02487   } while (!frame);
02488
02489   frame->time = self->time;
02490   self->frames_decoded++;
02491   self->time = (double)self->frames_decoded / self->framerate;
02492
02493   return frame;
02494 }
02495
02496 int plm_video_has_header(plm_video_t *self) {
02497   if (self->has_sequence_header) {
02498     return TRUE;
02499   }
02500
02501   if (self->start_code != PLM_START_SEQUENCE) {
02502     self->start_code = plm_buffer_find_start_code(self->buffer, PLM_START_SEQUENCE);
02503   }
02504   if (self->start_code == -1) {
02505     return FALSE;
02506   }
```

```
02507
02508   if (!plm_video_decode_sequence_header(self)) {
02509     return FALSE;
02510   }
02511
02512   return TRUE;
02513 }
02514
02515 int plm_video_decode_sequence_header(plm_video_t *self) {
02516   int max_header_size = 64 + 2 * 64 * 8; // 64 bit header + 2x 64 byte matrix
02517   if (!plm_buffer_has(self->buffer, max_header_size)) {
02518     return FALSE;
02519   }
02520
02521   self->width = plm_buffer_read(self->buffer, 12);
02522   self->height = plm_buffer_read(self->buffer, 12);
02523
02524   if (self->width <= 0 || self->height <= 0) {
02525     return FALSE;
02526   }
02527
02528   // Skip pixel aspect ratio
02529   plm_buffer_skip(self->buffer, 4);
02530
02531   self->framerate = PLM_VIDEO_PICTURE_RATE[plm_buffer_read(self->buffer, 4)];
02532
02533   // Skip bit_rate, marker, buffer_size and constrained bit
02534   plm_buffer_skip(self->buffer, 18 + 1 + 10 + 1);
02535
02536   // Load custom intra quant matrix?
02537   if (plm_buffer_read(self->buffer, 1)) {
02538     for (int i = 0; i < 64; i++) {
02539       int idx = PLM_VIDEO_ZIG_ZAG[i];
02540       self->intra_quant_matrix[idx] = plm_buffer_read(self->buffer, 8);
02541     }
02542   } else {
02543     memcpy(self->intra_quant_matrix, PLM_VIDEO_INTRA_QUANT_MATRIX, 64);
02544   }
02545
02546   // Load custom non intra quant matrix?
02547   if (plm_buffer_read(self->buffer, 1)) {
02548     for (int i = 0; i < 64; i++) {
02549       int idx = PLM_VIDEO_ZIG_ZAG[i];
02550       self->non_intra_quant_matrix[idx] = plm_buffer_read(self->buffer, 8);
02551     }
02552   } else {
02553     memcpy(self->non_intra_quant_matrix, PLM_VIDEO_NON_INTRA_QUANT_MATRIX, 64);
02554   }
02555
02556   self->mb_width = (self->width + 15) >> 4;
02557   self->mb_height = (self->height + 15) >> 4;
02558   self->mb_size = self->mb_width * self->mb_height;
02559
02560   self->luma_width = self->mb_width << 4;
02561   self->luma_height = self->mb_height << 4;
02562
02563   self->chroma_width = self->mb_width << 3;
02564   self->chroma_height = self->mb_height << 3;
02565
02566   // Allocate one big chunk of data for all 3 frames = 9 planes
02567   size_t luma_plane_size = self->luma_width * self->luma_height;
02568   size_t chroma_plane_size = self->chroma_width * self->chroma_height;
02569   size_t frame_data_size = (luma_plane_size + 2 * chroma_plane_size);
02570
02571   self->frames_data = (uint8_t *)PLM_MALLOC(frame_data_size * 3);
02572   plm_video_init_frame(self, &self->frame_current, self->frames_data + frame_data_size * 0);
02573   plm_video_init_frame(self, &self->frame_forward, self->frames_data + frame_data_size * 1);
02574   plm_video_init_frame(self, &self->frame_backward, self->frames_data + frame_data_size * 2);
02575
02576   self->has_sequence_header = TRUE;
02577   return TRUE;
02578 }
02579
02580 void plm_video_init_frame(plm_video_t *self, plm_frame_t *frame, uint8_t *base) {
02581   size_t luma_plane_size = self->luma_width * self->luma_height;
02582   size_t chroma_plane_size = self->chroma_width * self->chroma_height;
02583
02584   frame->width = self->width;
02585   frame->height = self->height;
02586   frame->y.width = self->luma_width;
02587   frame->y.height = self->luma_height;
02588   frame->y.data = base;
02589
02590   frame->cr.width = self->chroma_width;
02591   frame->cr.height = self->chroma_height;
02592   frame->cr.data = base + luma_plane_size;
02593
```

```
02594   frame->cb.width = self->chroma_width;
02595   frame->cb.height = self->chroma_height;
02596   frame->cb.data = base + luma_plane_size + chroma_plane_size;
02597 }
02598
02599 void plm_video_decode_picture(plm_video_t *self) {
02600   plm_buffer_skip(self->buffer, 10); // skip temporalReference
02601   self->picture_type = plm_buffer_read(self->buffer, 3);
02602   plm_buffer_skip(self->buffer, 16); // skip vbv_delay
02603
02604   // D frames or unknown coding type
02605   if (self->picture_type <= 0 || self->picture_type > PLM_VIDEO_PICTURE_TYPE_B) {
02606     return;
02607   }
02608
02609   // Forward full_px, f_code
02610   if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE || self->picture_type ==
      PLM_VIDEO_PICTURE_TYPE_B) {
02611     self->motion_forward.full_px = plm_buffer_read(self->buffer, 1);
02612     int f_code = plm_buffer_read(self->buffer, 3);
02613     if (f_code == 0) {
02614       // Ignore picture with zero f_code
02615       return;
02616     }
02617     self->motion_forward.r_size = f_code - 1;
02618   }
02619
02620   // Backward full_px, f_code
02621   if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_B) {
02622     self->motion_backward.full_px = plm_buffer_read(self->buffer, 1);
02623     int f_code = plm_buffer_read(self->buffer, 3);
02624     if (f_code == 0) {
02625       // Ignore picture with zero f_code
02626       return;
02627     }
02628     self->motion_backward.r_size = f_code - 1;
02629   }
02630
02631   plm_frame_t frame_temp = self->frame_forward;
02632   if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_INTRA || self->picture_type ==
      PLM_VIDEO_PICTURE_TYPE_PREDICTIVE) {
02633     self->frame_forward = self->frame_backward;
02634   }
02635
02636   // Find first slice start code; skip extension and user data
02637   do {
02638     self->start_code = plm_buffer_next_start_code(self->buffer);
02639   } while (self->start_code == PLM_START_EXTENSION || self->start_code == PLM_START_USER_DATA);
02640
02641   // Decode all slices
02642   while (PLM_START_IS_SLICE(self->start_code)) {
02643     plm_video_decode_slice(self, self->start_code & 0x000000FF);
02644     if (self->macroblock_address >= self->mb_size - 2) {
02645       break;
02646     }
02647     self->start_code = plm_buffer_next_start_code(self->buffer);
02648   }
02649
02650   // If this is a reference picture rotate the prediction pointers
02651   if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_INTRA || self->picture_type ==
      PLM_VIDEO_PICTURE_TYPE_PREDICTIVE) {
02652     self->frame_backward = self->frame_current;
02653     self->frame_current = frame_temp;
02654   }
02655 }
02656
02657 void plm_video_decode_slice(plm_video_t *self, int slice) {
02658   self->slice_begin = TRUE;
02659   self->macroblock_address = (slice - 1) * self->mb_width - 1;
02660
02661   // Reset motion vectors and DC predictors
02662   self->motion_backward.h = self->motion_forward.h = 0;
02663   self->motion_backward.v = self->motion_forward.v = 0;
02664   self->dc_predictor[0] = 128;
02665   self->dc_predictor[1] = 128;
02666   self->dc_predictor[2] = 128;
02667
02668   self->quantizer_scale = plm_buffer_read(self->buffer, 5);
02669
02670   // Skip extra
02671   while (plm_buffer_read(self->buffer, 1)) {
02672     plm_buffer_skip(self->buffer, 8);
02673   }
02674
02675   do {
02676     plm_video_decode_macroblock(self);
02677   } while (self->macroblock_address < self->mb_size - 1 && plm_buffer_peek_non_zero(self->buffer,
```

```
    23));
02678 }
02679
02680 void plm_video_decode_macroblock(plm_video_t *self) {
02681   // Decode increment
02682   int increment = 0;
02683   int t = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT);
02684
02685   while (t == 34) {
02686     // macroblock_stuffing
02687     t = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT);
02688   }
02689   while (t == 35) {
02690     // macroblock_escape
02691     increment += 33;
02692     t = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MACROBLOCK_ADDRESS_INCREMENT);
02693   }
02694   increment += t;
02695
02696   // Process any skipped macroblocks
02697   if (self->slice_begin) {
02698     // The first increment of each slice is relative to beginning of the
02699     // previous row, not the previous macroblock
02700     self->slice_begin = FALSE;
02701     self->macroblock_address += increment;
02702   } else {
02703     if (self->macroblock_address + increment >= self->mb_size) {
02704       return; // invalid
02705     }
02706     if (increment > 1) {
02707       // Skipped macroblocks reset DC predictors
02708       self->dc_predictor[0] = 128;
02709       self->dc_predictor[1] = 128;
02710       self->dc_predictor[2] = 128;
02711
02712       // Skipped macroblocks in P-pictures reset motion vectors
02713       if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE) {
02714         self->motion_forward.h = 0;
02715         self->motion_forward.v = 0;
02716       }
02717     }
02718
02719     // Predict skipped macroblocks
02720     while (increment > 1) {
02721       self->macroblock_address++;
02722       self->mb_row = self->macroblock_address / self->mb_width;
02723       self->mb_col = self->macroblock_address % self->mb_width;
02724
02725       plm_video_predict_macroblock(self);
02726       increment--;
02727     }
02728     self->macroblock_address++;
02729   }
02730
02731   self->mb_row = self->macroblock_address / self->mb_width;
02732   self->mb_col = self->macroblock_address % self->mb_width;
02733
02734   if (self->mb_col >= self->mb_width || self->mb_row >= self->mb_height) {
02735     return; // corrupt stream;
02736   }
02737
02738   // Process the current macroblock
02739   const plm_vlc_t *table = PLM_VIDEO_MACROBLOCK_TYPE[self->picture_type];
02740   self->macroblock_type = plm_buffer_read_vlc(self->buffer, table);
02741
02742   self->macroblock_intra = (self->macroblock_type & 0x01);
02743   self->motion_forward.is_set = (self->macroblock_type & 0x08);
02744   self->motion_backward.is_set = (self->macroblock_type & 0x04);
02745
02746   // Quantizer scale
02747   if ((self->macroblock_type & 0x10) != 0) {
02748     self->quantizer_scale = plm_buffer_read(self->buffer, 5);
02749   }
02750
02751   if (self->macroblock_intra) {
02752     // Intra-coded macroblocks reset motion vectors
02753     self->motion_backward.h = self->motion_forward.h = 0;
02754     self->motion_backward.v = self->motion_forward.v = 0;
02755   } else {
02756     // Non-intra macroblocks reset DC predictors
02757     self->dc_predictor[0] = 128;
02758     self->dc_predictor[1] = 128;
02759     self->dc_predictor[2] = 128;
02760
02761     plm_video_decode_motion_vectors(self);
02762     plm_video_predict_macroblock(self);
02763   }
```

```
02764
02765   // Decode blocks
02766   int cbp = ((self->macroblock_type & 0x02) != 0) ? plm_buffer_read_vlc(self->buffer,
      PLM_VIDEO_CODE_BLOCK_PATTERN)
02767                                                   : (self->macroblock_intra ? 0x3f : 0);
02768
02769   for (int block = 0, mask = 0x20; block < 6; block++) {
02770     if ((cbp & mask) != 0) {
02771       plm_video_decode_block(self, block);
02772     }
02773     mask »= 1;
02774   }
02775 }
02776
02777 void plm_video_decode_motion_vectors(plm_video_t *self) {
02778
02779   // Forward
02780   if (self->motion_forward.is_set) {
02781     int r_size = self->motion_forward.r_size;
02782     self->motion_forward.h = plm_video_decode_motion_vector(self, r_size, self->motion_forward.h);
02783     self->motion_forward.v = plm_video_decode_motion_vector(self, r_size, self->motion_forward.v);
02784   } else if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_PREDICTIVE) {
02785     // No motion information in P-picture, reset vectors
02786     self->motion_forward.h = 0;
02787     self->motion_forward.v = 0;
02788   }
02789
02790   if (self->motion_backward.is_set) {
02791     int r_size = self->motion_backward.r_size;
02792     self->motion_backward.h = plm_video_decode_motion_vector(self, r_size, self->motion_backward.h);
02793     self->motion_backward.v = plm_video_decode_motion_vector(self, r_size, self->motion_backward.v);
02794   }
02795 }
02796
02797 int plm_video_decode_motion_vector(plm_video_t *self, int r_size, int motion) {
02798   int fscale = 1 « r_size;
02799   int m_code = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_MOTION);
02800   int r = 0;
02801   int d;
02802
02803   if ((m_code != 0) && (fscale != 1)) {
02804     r = plm_buffer_read(self->buffer, r_size);
02805     d = ((abs(m_code) - 1) « r_size) + r + 1;
02806     if (m_code < 0) {
02807       d = -d;
02808     }
02809   } else {
02810     d = m_code;
02811   }
02812
02813   motion += d;
02814   if (motion > (fscale « 4) - 1) {
02815     motion -= fscale « 5;
02816   } else if (motion < ((-fscale) « 4)) {
02817     motion += fscale « 5;
02818   }
02819
02820   return motion;
02821 }
02822
02823 void plm_video_predict_macroblock(plm_video_t *self) {
02824   int fw_h = self->motion_forward.h;
02825   int fw_v = self->motion_forward.v;
02826
02827   if (self->motion_forward.full_px) {
02828     fw_h «= 1;
02829     fw_v «= 1;
02830   }
02831
02832   if (self->picture_type == PLM_VIDEO_PICTURE_TYPE_B) {
02833     int bw_h = self->motion_backward.h;
02834     int bw_v = self->motion_backward.v;
02835
02836     if (self->motion_backward.full_px) {
02837       bw_h «= 1;
02838       bw_v «= 1;
02839     }
02840
02841     if (self->motion_forward.is_set) {
02842       plm_video_copy_macroblock(self, &self->frame_forward, fw_h, fw_v);
02843       if (self->motion_backward.is_set) {
02844         plm_video_interpolate_macroblock(self, &self->frame_backward, bw_h, bw_v);
02845       }
02846     } else {
02847       plm_video_copy_macroblock(self, &self->frame_backward, bw_h, bw_v);
02848     }
02849   } else {
```

```
02850     plm_video_copy_macroblock(self, &self->frame_forward, fw_h, fw_v);
02851   }
02852 }
02853
02854 void plm_video_copy_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v) {
02855   plm_frame_t *d = &self->frame_current;
02856   plm_video_process_macroblock(self, s->y.data, d->y.data, motion_h, motion_v, 16, FALSE);
02857   plm_video_process_macroblock(self, s->cr.data, d->cr.data, motion_h / 2, motion_v / 2, 8, FALSE);
02858   plm_video_process_macroblock(self, s->cb.data, d->cb.data, motion_h / 2, motion_v / 2, 8, FALSE);
02859 }
02860
02861 void plm_video_interpolate_macroblock(plm_video_t *self, plm_frame_t *s, int motion_h, int motion_v) {
02862   plm_frame_t *d = &self->frame_current;
02863   plm_video_process_macroblock(self, s->y.data, d->y.data, motion_h, motion_v, 16, TRUE);
02864   plm_video_process_macroblock(self, s->cr.data, d->cr.data, motion_h / 2, motion_v / 2, 8, TRUE);
02865   plm_video_process_macroblock(self, s->cb.data, d->cb.data, motion_h / 2, motion_v / 2, 8, TRUE);
02866 }
02867
02868 #define PLM_BLOCK_SET(DEST, DEST_INDEX, DEST_WIDTH, SOURCE_INDEX, SOURCE_WIDTH, BLOCK_SIZE, OP)
    \
02869   do {
    \
02870     int dest_scan = DEST_WIDTH - BLOCK_SIZE;
    \
02871     int source_scan = SOURCE_WIDTH - BLOCK_SIZE;
    \
02872     for (int y = 0; y < BLOCK_SIZE; y++) {
    \
02873       for (int x = 0; x < BLOCK_SIZE; x++) {
    \
02874         DEST[DEST_INDEX] = OP;
    \
02875         SOURCE_INDEX++;
    \
02876         DEST_INDEX++;
    \
02877       }
    \
02878       SOURCE_INDEX += source_scan;
    \
02879       DEST_INDEX += dest_scan;
    \
02880     }
    \
02881   } while (FALSE)
02882
02883 void plm_video_process_macroblock(plm_video_t *self, uint8_t *s, uint8_t *d, int motion_h, int
      motion_v, int block_size,
02884                                   int interpolate) {
02885   int dw = self->mb_width * block_size;
02886
02887   int hp = motion_h >> 1;
02888   int vp = motion_v >> 1;
02889   int odd_h = (motion_h & 1) == 1;
02890   int odd_v = (motion_v & 1) == 1;
02891
02892   unsigned int si = ((self->mb_row * block_size) + vp) * dw + (self->mb_col * block_size) + hp;
02893   unsigned int di = (self->mb_row * dw + self->mb_col) * block_size;
02894
02895   unsigned int max_address = (dw * (self->mb_height * block_size - block_size + 1) - block_size);
02896   if (si > max_address || di > max_address) {
02897     return; // corrupt video
02898   }
02899
02900 #define PLM_MB_CASE(INTERPOLATE, ODD_H, ODD_V, OP)
    \
02901   case ((INTERPOLATE << 2) | (ODD_H << 1) | (ODD_V)):
    \
02902     PLM_BLOCK_SET(d, di, dw, si, dw, block_size, OP);
    \
02903     break
02904
02905   switch ((interpolate << 2) | (odd_h << 1) | (odd_v)) {
02906     PLM_MB_CASE(0, 0, 0, (s[si]));
02907     PLM_MB_CASE(0, 0, 1, (s[si] + s[si + dw] + 1) >> 1);
02908     PLM_MB_CASE(0, 1, 0, (s[si] + s[si + 1] + 1) >> 1);
02909     PLM_MB_CASE(0, 1, 1, (s[si] + s[si + 1] + s[si + dw] + s[si + dw + 1] + 2) >> 2);
02910
02911     PLM_MB_CASE(1, 0, 0, (d[di] + (s[si]) + 1) >> 1);
02912     PLM_MB_CASE(1, 0, 1, (d[di] + ((s[si] + s[si + dw] + 1) >> 1) + 1) >> 1);
02913     PLM_MB_CASE(1, 1, 0, (d[di] + ((s[si] + s[si + 1] + 1) >> 1) + 1) >> 1);
02914     PLM_MB_CASE(1, 1, 1, (d[di] + ((s[si] + s[si + 1] + s[si + dw] + s[si + dw + 1] + 2) >> 2) + 1) >>
      1);
02915   }
02916
02917 #undef PLM_MB_CASE
02918 }
```

```
02919
02920 void plm_video_decode_block(plm_video_t *self, int block) {
02921
02922   int n = 0;
02923   uint8_t *quant_matrix;
02924
02925   // Decode DC coefficient of intra-coded blocks
02926   if (self->macroblock_intra) {
02927     int predictor;
02928     int dct_size;
02929
02930     // DC prediction
02931     int plane_index = block > 3 ? block - 3 : 0;
02932     predictor = self->dc_predictor[plane_index];
02933     dct_size = plm_buffer_read_vlc(self->buffer, PLM_VIDEO_DCT_SIZE[plane_index]);
02934
02935     // Read DC coeff
02936     if (dct_size > 0) {
02937       int differential = plm_buffer_read(self->buffer, dct_size);
02938       if ((differential & (1 << (dct_size - 1))) != 0) {
02939         self->block_data[0] = predictor + differential;
02940       } else {
02941         self->block_data[0] = predictor + (-(1 << dct_size) | (differential + 1));
02942       }
02943     } else {
02944       self->block_data[0] = predictor;
02945     }
02946
02947     // Save predictor value
02948     self->dc_predictor[plane_index] = self->block_data[0];
02949
02950     // Dequantize + premultiply
02951     self->block_data[0] <<= (3 + 5);
02952
02953     quant_matrix = self->intra_quant_matrix;
02954     n = 1;
02955   } else {
02956     quant_matrix = self->non_intra_quant_matrix;
02957   }
02958
02959   // Decode AC coefficients (+DC for non-intra)
02960   int level = 0;
02961   while (TRUE) {
02962     int run = 0;
02963     uint16_t coeff = plm_buffer_read_vlc_uint(self->buffer, PLM_VIDEO_DCT_COEFF);
02964
02965     if ((coeff == 0x0001) && (n > 0) && (plm_buffer_read(self->buffer, 1) == 0)) {
02966       // end_of_block
02967       break;
02968     }
02969     if (coeff == 0xffff) {
02970       // escape
02971       run = plm_buffer_read(self->buffer, 6);
02972       level = plm_buffer_read(self->buffer, 8);
02973       if (level == 0) {
02974         level = plm_buffer_read(self->buffer, 8);
02975       } else if (level == 128) {
02976         level = plm_buffer_read(self->buffer, 8) - 256;
02977       } else if (level > 128) {
02978         level = level - 256;
02979       }
02980     } else {
02981       run = coeff >> 8;
02982       level = coeff & 0xff;
02983       if (plm_buffer_read(self->buffer, 1)) {
02984         level = -level;
02985       }
02986     }
02987
02988     n += run;
02989     if (n < 0 || n >= 64) {
02990       return; // invalid
02991     }
02992
02993     int de_zig_zagged = PLM_VIDEO_ZIG_ZAG[n];
02994     n++;
02995
02996     // Dequantize, oddify, clip
02997     level <<= 1;
02998     if (!self->macroblock_intra) {
02999       level += (level < 0 ? -1 : 1);
03000     }
03001     level = (level * self->quantizer_scale * quant_matrix[de_zig_zagged]) >> 4;
03002     if ((level & 1) == 0) {
03003       level -= level > 0 ? 1 : -1;
03004     }
03005     if (level > 2047) {
```

```
03006       level = 2047;
03007     } else if (level < -2048) {
03008       level = -2048;
03009     }
03010
03011     // Save premultiplied coefficient
03012     self->block_data[de_zig_zagged] = level * PLM_VIDEO_PREMULTIPLIER_MATRIX[de_zig_zagged];
03013   }
03014
03015   // Move block to its place
03016   uint8_t *d;
03017   int dw;
03018   int di;
03019
03020   if (block < 4) {
03021     d = self->frame_current.y.data;
03022     dw = self->luma_width;
03023     di = (self->mb_row * self->luma_width + self->mb_col) << 4;
03024     if ((block & 1) != 0) {
03025       di += 8;
03026     }
03027     if ((block & 2) != 0) {
03028       di += self->luma_width << 3;
03029     }
03030   } else {
03031     d = (block == 4) ? self->frame_current.cb.data : self->frame_current.cr.data;
03032     dw = self->chroma_width;
03033     di = ((self->mb_row * self->luma_width) << 2) + (self->mb_col << 3);
03034   }
03035
03036   int *s = self->block_data;
03037   int si = 0;
03038   if (self->macroblock_intra) {
03039     // Overwrite (no prediction)
03040     if (n == 1) {
03041       int clamped = plm_clamp((s[0] + 128) >> 8);
03042       PLM_BLOCK_SET(d, di, dw, si, 8, 8, clamped);
03043       s[0] = 0;
03044     } else {
03045       plm_video_idct(s);
03046       PLM_BLOCK_SET(d, di, dw, si, 8, 8, plm_clamp(s[si]));
03047       memset(self->block_data, 0, sizeof(self->block_data));
03048     }
03049   } else {
03050     // Add data to the predicted macroblock
03051     if (n == 1) {
03052       int value = (s[0] + 128) >> 8;
03053       PLM_BLOCK_SET(d, di, dw, si, 8, 8, plm_clamp(d[di] + value));
03054       s[0] = 0;
03055     } else {
03056       plm_video_idct(s);
03057       PLM_BLOCK_SET(d, di, dw, si, 8, 8, plm_clamp(d[di] + s[si]));
03058       memset(self->block_data, 0, sizeof(self->block_data));
03059     }
03060   }
03061 }
03062
03063 void plm_video_idct(int *block) {
03064   int b1, b3, b4, b6, b7, tmp1, tmp2, m0, x0, x1, x2, x3, x4, y3, y4, y5, y6, y7;
03065
03066   // Transform columns
03067   for (int i = 0; i < 8; ++i) {
03068     b1 = block[4 * 8 + i];
03069     b3 = block[2 * 8 + i] + block[6 * 8 + i];
03070     b4 = block[5 * 8 + i] - block[3 * 8 + i];
03071     tmp1 = block[1 * 8 + i] + block[7 * 8 + i];
03072     tmp2 = block[3 * 8 + i] + block[5 * 8 + i];
03073     b6 = block[1 * 8 + i] - block[7 * 8 + i];
03074     b7 = tmp1 + tmp2;
03075     m0 = block[0 * 8 + i];
03076     x4 = ((b6 * 473 - b4 * 196 + 128) >> 8) - b7;
03077     x0 = x4 - (((tmp1 - tmp2) * 362 + 128) >> 8);
03078     x1 = m0 - b1;
03079     x2 = (((block[2 * 8 + i] - block[6 * 8 + i]) * 362 + 128) >> 8) - b3;
03080     x3 = m0 + b1;
03081     y3 = x1 + x2;
03082     y4 = x3 + b3;
03083     y5 = x1 - x2;
03084     y6 = x3 - b3;
03085     y7 = -x0 - ((b4 * 473 + b6 * 196 + 128) >> 8);
03086     block[0 * 8 + i] = b7 + y4;
03087     block[1 * 8 + i] = x4 + y3;
03088     block[2 * 8 + i] = y5 - x0;
03089     block[3 * 8 + i] = y6 - y7;
03090     block[4 * 8 + i] = y6 + y7;
03091     block[5 * 8 + i] = x0 + y5;
03092     block[6 * 8 + i] = y3 - x4;
```

```
03093     block[7 * 8 + i] = y4 - b7;
03094   }
03095
03096   // Transform rows
03097   for (int i = 0; i < 64; i += 8) {
03098     b1 = block[4 + i];
03099     b3 = block[2 + i] + block[6 + i];
03100     b4 = block[5 + i] - block[3 + i];
03101     tmp1 = block[1 + i] + block[7 + i];
03102     tmp2 = block[3 + i] + block[5 + i];
03103     b6 = block[1 + i] - block[7 + i];
03104     b7 = tmp1 + tmp2;
03105     m0 = block[0 + i];
03106     x4 = ((b6 * 473 - b4 * 196 + 128) >> 8) - b7;
03107     x0 = x4 - (((tmp1 - tmp2) * 362 + 128) >> 8);
03108     x1 = m0 - b1;
03109     x2 = (((block[2 + i] - block[6 + i]) * 362 + 128) >> 8) - b3;
03110     x3 = m0 + b1;
03111     y3 = x1 + x2;
03112     y4 = x3 + b3;
03113     y5 = x1 - x2;
03114     y6 = x3 - b3;
03115     y7 = -x0 - ((b4 * 473 + b6 * 196 + 128) >> 8);
03116     block[0 + i] = (b7 + y4 + 128) >> 8;
03117     block[1 + i] = (x4 + y3 + 128) >> 8;
03118     block[2 + i] = (y5 - x0 + 128) >> 8;
03119     block[3 + i] = (y6 - y7 + 128) >> 8;
03120     block[4 + i] = (y6 + y7 + 128) >> 8;
03121     block[5 + i] = (x0 + y5 + 128) >> 8;
03122     block[6 + i] = (y3 - x4 + 128) >> 8;
03123     block[7 + i] = (y4 - b7 + 128) >> 8;
03124   }
03125 }
03126
03127 // YCbCr conversion following the BT.601 standard:
03128 // https://infogalactic.com/info/YCbCr#ITU-R_BT.601_conversion
03129
03130 #define PLM_PUT_PIXEL(RI, GI, BI, Y_OFFSET, DEST_OFFSET)
    \
03131   y = ((frame->y.data[y_index + Y_OFFSET] - 16) * 76309) >> 16;
    \
03132   dest[d_index + DEST_OFFSET + RI] = plm_clamp(y + r);
    \
03133   dest[d_index + DEST_OFFSET + GI] = plm_clamp(y - g);
    \
03134   dest[d_index + DEST_OFFSET + BI] = plm_clamp(y + b);
03135
03136 #define PLM_DEFINE_FRAME_CONVERT_FUNCTION(NAME, BYTES_PER_PIXEL, RI, GI, BI)
    \
03137   void NAME(plm_frame_t *frame, uint8_t *dest, int stride) {
    \
03138     int cols = frame->width >> 1;
    \
03139     int rows = frame->height >> 1;
    \
03140     int yw = frame->y.width;
    \
03141     int cw = frame->cb.width;
    \
03142     for (int row = 0; row < rows; row++) {
    \
03143       int c_index = row * cw;
    \
03144       int y_index = row * 2 * yw;
    \
03145       int d_index = row * 2 * stride;
    \
03146       for (int col = 0; col < cols; col++) {
    \
03147         int y;
    \
03148         int cr = frame->cr.data[c_index] - 128;
    \
03149         int cb = frame->cb.data[c_index] - 128;
    \
03150         int r = (cr * 104597) >> 16;
    \
03151         int g = (cb * 25674 + cr * 53278) >> 16;
    \
03152         int b = (cb * 132201) >> 16;
    \
03153         PLM_PUT_PIXEL(RI, GI, BI, 0, 0);
    \
03154         PLM_PUT_PIXEL(RI, GI, BI, 1, BYTES_PER_PIXEL);
    \
03155         PLM_PUT_PIXEL(RI, GI, BI, yw, stride);
    \
```

```
03156         PLM_PUT_PIXEL(RI, GI, BI, yw + 1, stride + BYTES_PER_PIXEL);
      \
03157         c_index += 1;
      \
03158         y_index += 2;
      \
03159         d_index += 2 * BYTES_PER_PIXEL;
      \
03160       }
      \
03161     }
      \
03162   }
03163
03164 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_rgb, 3, 0, 1, 2)
03165 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_bgr, 3, 2, 1, 0)
03166 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_rgba, 4, 0, 1, 2)
03167 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_bgra, 4, 2, 1, 0)
03168 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_argb, 4, 1, 2, 3)
03169 PLM_DEFINE_FRAME_CONVERT_FUNCTION(plm_frame_to_abgr, 4, 3, 2, 1)
03170
03171 #undef PLM_PUT_PIXEL
03172 #undef PLM_DEFINE_FRAME_CONVERT_FUNCTION
03173
03174 // -----------------------------------------------------------------------------
03175 // plm_audio implementation
03176
03177 // Based on kjmp2 by Martin J. Fiedler
03178 // http://keyj.emphy.de/kjmp2/
03179
03180 static const int PLM_AUDIO_FRAME_SYNC = 0x7ff;
03181
03182 static const int PLM_AUDIO_MPEG_2_5 = 0x0;
03183 static const int PLM_AUDIO_MPEG_2 = 0x2;
03184 static const int PLM_AUDIO_MPEG_1 = 0x3;
03185
03186 static const int PLM_AUDIO_LAYER_III = 0x1;
03187 static const int PLM_AUDIO_LAYER_II = 0x2;
03188 static const int PLM_AUDIO_LAYER_I = 0x3;
03189
03190 static const int PLM_AUDIO_MODE_STEREO = 0x0;
03191 static const int PLM_AUDIO_MODE_JOINT_STEREO = 0x1;
03192 static const int PLM_AUDIO_MODE_DUAL_CHANNEL = 0x2;
03193 static const int PLM_AUDIO_MODE_MONO = 0x3;
03194
03195 static const unsigned short PLM_AUDIO_SAMPLE_RATE[] = {
03196     44100, 48000, 32000, 0, // MPEG-1
03197     22050, 24000, 16000, 0  // MPEG-2
03198 };
03199
03200 static const short PLM_AUDIO_BIT_RATE[] = {
03201     32, 48, 56, 64, 80, 96, 112, 128, 160, 192, 224, 256, 320, 384, // MPEG-1
03202     8,  16, 24, 32, 40, 48, 56,  64,  80,  96,  112, 128, 144, 160  // MPEG-2
03203 };
03204
03205 static const int PLM_AUDIO_SCALEFACTOR_BASE[] = {0x02000000, 0x01965FEA, 0x01428A30};
03206
03207 static const float PLM_AUDIO_SYNTHESIS_WINDOW[] = {
03208     0.0,      -0.5,     -0.5,     -0.5,     -0.5,     -0.5,     -0.5,     -1.0,     -1.0,     -1.0,
      -1.0,
03209     -1.5,     -1.5,     -2.0,     -2.0,     -2.5,     -2.5,     -3.0,     -3.5,     -3.5,     -4.0,
      -4.5,
03210     -5.0,     -5.5,     -6.5,     -7.0,     -8.0,     -8.5,     -9.5,     -10.5,    -12.0,    -13.0,
      -14.5,
03211     -15.5,    -17.5,    -19.0,    -20.5,    -22.5,    -24.5,    -26.5,    -29.0,    -31.5,    -34.0,
      -36.5,
03212     -39.5,    -42.5,    -45.5,    -48.5,    -52.0,    -55.5,    -58.5,    -62.5,    -66.0,    -69.5,
      -73.5,
03213     -77.0,    -80.5,    -84.5,    -88.0,    -91.5,    -95.0,    -98.0,    -101.0,   -104.0,   106.5,
      109.0,
03214     111.0,    112.5,    113.5,    114.0,    114.0,    113.5,    112.0,    110.5,    107.5,    104.0,
      100.0,
03215     94.5,     88.5,     81.5,     73.0,     63.5,     53.0,     41.5,     28.5,     14.5,     -1.0,
      -18.0,
03216     -36.0,    -55.5,    -76.5,    -98.5,    -122.0,   -147.0,   -173.5,   -200.5,   -229.5,   -259.5,
      -290.5,
03217     -322.5,   -355.5,   -389.5,   -424.0,   -459.5,   -495.5,   -532.0,   -568.5,   -605.0,   -641.5,
      -678.0,
03218     -714.0,   -749.0,   -783.5,   -817.0,   -849.0,   -879.5,   -908.5,   -935.0,   -959.5,   -981.0,
      -1000.5,
03219     -1016.0,  -1028.5,  -1037.5,  -1042.5,  -1043.5,  -1040.0,  -1031.5,  1018.5,   1000.0,   976.0,
      946.5,
03220     911.0,    869.5,    822.0,    767.5,    707.0,    640.0,    565.5,    485.0,    397.0,    302.5,
      201.0,
03221     92.5,     -22.5,    -144.0,   -272.5,   -407.0,   -547.5,   -694.0,   -846.0,   -1003.0,  -1165.0,
      -1331.5,
03222     -1502.0,  -1675.5,  -1852.5,  -2031.5,  -2212.5,  -2394.0,  -2576.5,  -2758.5,  -2939.5,  -3118.5,
```

```
      -3294.5,
03223     -3467.5,  -3635.5,  -3798.5,  -3955.0,  -4104.5,  -4245.5,  -4377.5,  -4499.0,  -4609.5,  -4708.0,
      -4792.5,
03224     -4863.5,  -4919.0,  -4958.0,  -4979.5,  -4983.0,  -4967.5,  -4931.5,  -4875.0,  -4796.0,  -4694.5,
      -4569.5,
03225     -4420.0,  -4246.0,  -4046.0,  -3820.0,  -3567.0,  3287.0,   2979.5,   2644.0,   2280.5,   1888.0,
      1467.5,
03226     1018.5,   541.0,    35.0,     -499.0,   -1061.0,  -1650.0,  -2266.5,  -2909.0,  -3577.0,  -4270.0,
      -4987.5,
03227     -5727.5,  -6490.0,  -7274.0,  -8077.5,  -8899.5,  -9739.0,  -10594.5, -11464.5, -12347.0,
      -13241.0, -14144.5,
03228     -15056.0, -15973.5, -16895.5, -17820.0, -18744.5, -19668.0, -20588.0, -21503.0, -22410.5,
      -23308.5, -24195.0,
03229     -25068.5, -25926.5, -26767.0, -27589.0, -28389.0, -29166.5, -29919.0, -30644.5, -31342.0,
      -32009.5, -32645.0,
03230     -33247.0, -33814.5, -34346.0, -34839.5, -35295.0, -35710.0, -36084.5, -36417.5, -36707.5,
      -36954.0, -37156.5,
03231     -37315.0, -37428.0, -37496.0, 37519.0,  37496.0,  37428.0,  37315.0,  37156.5,  36954.0,  36707.5,
      36417.5,
03232     36084.5,  35710.0,  35295.0,  34839.5,  34346.0,  33814.5,  33247.0,  32645.0,  32009.5,  31342.0,
      30644.5,
03233     29919.0,  29166.5,  28389.0,  27589.0,  26767.0,  25926.5,  25068.5,  24195.0,  23308.5,  22410.5,
      21503.0,
03234     20588.0,  19668.0,  18744.5,  17820.0,  16895.5,  15973.5,  15056.0,  14144.5,  13241.0,  12347.0,
      11464.5,
03235     10594.5,  9739.0,   8899.5,   8077.5,   7274.0,   6490.0,   5727.5,   4987.5,   4270.0,   3577.0,
      2909.0,
03236     2266.5,   1650.0,   1061.0,   499.0,    -35.0,    -541.0,   -1018.5,  -1467.5,  -1888.0,  -2280.5,
      -2644.0,
03237     -2979.5,  3287.0,   3567.0,   3820.0,   4046.0,   4246.0,   4420.0,   4569.5,   4694.5,   4796.0,
      4875.0,
03238     4931.5,   4967.5,   4983.0,   4979.5,   4958.0,   4919.0,   4863.5,   4792.5,   4708.0,   4609.5,
      4499.0,
03239     4377.5,   4245.5,   4104.5,   3955.0,   3798.5,   3635.5,   3467.5,   3294.5,   3118.5,   2939.5,
      2758.5,
03240     2576.5,   2394.0,   2212.5,   2031.5,   1852.5,   1675.5,   1502.0,   1331.5,   1165.0,   1003.0,
      846.0,
03241     694.0,    547.5,    407.0,    272.5,    144.0,    22.5,     -92.5,    -201.0,   -302.5,   -397.0,
      -485.0,
03242     -565.5,   -640.0,   -707.0,   -767.5,   -822.0,   -869.5,   -911.0,   -946.5,   -976.0,   -1000.0,
      1018.5,
03243     1031.5,   1040.0,   1043.5,   1042.5,   1037.5,   1028.5,   1016.0,   1000.5,   981.0,    959.5,
      935.0,
03244     908.5,    879.5,    849.0,    817.0,    783.5,    749.0,    714.0,    678.0,    641.5,    605.0,
      568.5,
03245     532.0,    495.5,    459.5,    424.0,    389.5,    355.5,    322.5,    290.5,    259.5,    229.5,
      200.5,
03246     173.5,    147.0,    122.0,    98.5,     76.5,     55.5,     36.0,     18.0,     1.0,      -14.5,
      -28.5,
03247     -41.5,    -53.0,    -63.5,    -73.0,    -81.5,    -88.5,    -94.5,    -100.0,   -104.0,   -107.5,
      -110.5,
03248     -112.0,   -113.5,   -114.0,   -114.0,   -113.5,   -112.5,   -111.0,   -109.0,   106.5,    104.0,
      101.0,
03249     98.0,     95.0,     91.5,     88.0,     84.5,     80.5,     77.0,     73.5,     69.5,     66.0,
      62.5,
03250     58.5,     55.5,     52.0,     48.5,     45.5,     42.5,     39.5,     36.5,     34.0,     31.5,
      29.0,
03251     26.5,     24.5,     22.5,     20.5,     19.0,     17.5,     15.5,     14.5,     13.0,     12.0,
      10.5,
03252     9.5,      8.5,      8.0,      7.0,      6.5,      5.5,      5.0,      4.5,      4.0,      3.5,
      3.5,
03253     3.0,      2.5,      2.5,      2.0,      2.0,      1.5,      1.5,      1.0,      1.0,      1.0,
      1.0,
03254     0.5,      0.5,      0.5,      0.5,      0.5,      0.5};
03255
03256 // Quantizer lookup, step 1: bitrate classes
03257 static const uint8_t PLM_AUDIO_QUANT_LUT_STEP_1[2][16] = {
03258     // 32, 48, 56, 64, 80, 96,112,128,160,192,224,256,320,384 <- bitrate
03259     {0, 0, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2}, // mono
03260     // 16, 24, 28, 32, 40, 48, 56, 64, 80, 96,112,128,160,192 <- bitrate / chan
03261     {0, 0, 0, 0, 0, 0, 1, 1, 1, 2, 2, 2, 2, 2} // stereo
03262 };
03263
03264 // Quantizer lookup, step 2: bitrate class, sample rate -> B2 table idx, sblimit
03265 #define PLM_AUDIO_QUANT_TAB_A (27 | 64) // Table 3-B.2a: high-rate, sblimit = 27
03266 #define PLM_AUDIO_QUANT_TAB_B (30 | 64) // Table 3-B.2b: high-rate, sblimit = 30
03267 #define PLM_AUDIO_QUANT_TAB_C 8         // Table 3-B.2c:  low-rate, sblimit =  8
03268 #define PLM_AUDIO_QUANT_TAB_D 12        // Table 3-B.2d:  low-rate, sblimit = 12
03269
03270 static const uint8_t QUANT_LUT_STEP_2[3][3] = {
03271     // 44.1 kHz,              48 kHz,               32 kHz
03272     {PLM_AUDIO_QUANT_TAB_C, PLM_AUDIO_QUANT_TAB_C, PLM_AUDIO_QUANT_TAB_D}, // 32 - 48 kbit/sec/ch
03273     {PLM_AUDIO_QUANT_TAB_A, PLM_AUDIO_QUANT_TAB_A, PLM_AUDIO_QUANT_TAB_A}, // 56 - 80 kbit/sec/ch
03274     {PLM_AUDIO_QUANT_TAB_B, PLM_AUDIO_QUANT_TAB_A, PLM_AUDIO_QUANT_TAB_B}  // 96+    kbit/sec/ch
03275 };
03276
03277 // Quantizer lookup, step 3: B2 table, subband -> nbal, row index
```

```
03278 // (upper 4 bits: nbal, lower 4 bits: row index)
03279 static const uint8_t PLM_AUDIO_QUANT_LUT_STEP_3[3][32] = {
03280     // Low-rate table (3-B.2c and 3-B.2d)
03281     {0x44, 0x44, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34},
03282     // High-rate table (3-B.2a and 3-B.2b)
03283     {0x43, 0x43, 0x43, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x31, 0x31, 0x31, 0x31,
03284      0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x31, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20, 0x20},
03285     // MPEG-2 LSR table (B.2 in ISO 13818-3)
03286     {0x45, 0x45, 0x45, 0x45, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x34, 0x24, 0x24, 0x24, 0x24,
03287      0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24, 0x24}};
03288
03289 // Quantizer lookup, step 4: table row, allocation[] value -> quant table index
03290 static const uint8_t PLM_AUDIO_QUANT_LUT_STEP_4[6][16] = {{0, 1, 2, 17},
03291                                                           {0, 1, 2, 3, 4, 5, 6, 17},
03292                                                           {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
      13, 14, 17},
03293                                                           {0, 1, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
      15, 16, 17},
03294                                                           {0, 1, 2, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
      14, 15, 17},
03295                                                           {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
      13, 14, 15}};
03296
03297 typedef struct plm_quantizer_spec_t {
03298   unsigned short levels;
03299   unsigned char group;
03300   unsigned char bits;
03301 } plm_quantizer_spec_t;
03302
03303 static const plm_quantizer_spec_t PLM_AUDIO_QUANT_TAB[] = {
03304     {3, 1, 5},       //  1
03305     {5, 1, 7},       //  2
03306     {7, 0, 3},       //  3
03307     {9, 1, 10},      //  4
03308     {15, 0, 4},      //  5
03309     {31, 0, 5},      //  6
03310     {63, 0, 6},      //  7
03311     {127, 0, 7},     //  8
03312     {255, 0, 8},     //  9
03313     {511, 0, 9},     // 10
03314     {1023, 0, 10},   // 11
03315     {2047, 0, 11},   // 12
03316     {4095, 0, 12},   // 13
03317     {8191, 0, 13},   // 14
03318     {16383, 0, 14}, // 15
03319     {32767, 0, 15}, // 16
03320     {65535, 0, 16}   // 17
03321 };
03322
03323 struct plm_audio_t {
03324   double time;
03325   int samples_decoded;
03326   int samplerate_index;
03327   int bitrate_index;
03328   int version;
03329   int layer;
03330   int mode;
03331   int bound;
03332   int v_pos;
03333   int next_frame_data_size;
03334   int has_header;
03335
03336   plm_buffer_t *buffer;
03337   int destroy_buffer_when_done;
03338
03339   const plm_quantizer_spec_t *allocation[2][32];
03340   uint8_t scale_factor_info[2][32];
03341   int scale_factor[2][32][3];
03342   int sample[2][32][3];
03343
03344   plm_samples_t samples;
03345   float D[1024];
03346   float V[2][1024];
03347   float U[32];
03348 };
03349
03350 int plm_audio_find_frame_sync(plm_audio_t *self);
03351 int plm_audio_decode_header(plm_audio_t *self);
03352 void plm_audio_decode_frame(plm_audio_t *self);
03353 const plm_quantizer_spec_t *plm_audio_read_allocation(plm_audio_t *self, int sb, int tab3);
03354 void plm_audio_read_samples(plm_audio_t *self, int ch, int sb, int part);
03355 void plm_audio_idct36(int s[32][3], int ss, float *d, int dp);
03356
03357 plm_audio_t *plm_audio_create_with_buffer(plm_buffer_t *buffer, int destroy_when_done) {
03358   plm_audio_t *self = (plm_audio_t *)PLM_MALLOC(sizeof(plm_audio_t));
03359   memset(self, 0, sizeof(plm_audio_t));
03360
```

```
03361   self->samples.count = PLM_AUDIO_SAMPLES_PER_FRAME;
03362   self->buffer = buffer;
03363   self->destroy_buffer_when_done = destroy_when_done;
03364   self->samplerate_index = 3; // Indicates 0
03365
03366   memcpy(self->D, PLM_AUDIO_SYNTHESIS_WINDOW, 512 * sizeof(float));
03367   memcpy(self->D + 512, PLM_AUDIO_SYNTHESIS_WINDOW, 512 * sizeof(float));
03368
03369   // Attempt to decode first header
03370   self->next_frame_data_size = plm_audio_decode_header(self);
03371
03372   return self;
03373 }
03374
03375 void plm_audio_destroy(plm_audio_t *self) {
03376   if (self->destroy_buffer_when_done) {
03377     plm_buffer_destroy(self->buffer);
03378   }
03379   PLM_FREE(self);
03380 }
03381
03382 int plm_audio_has_header(plm_audio_t *self) {
03383   if (self->has_header) {
03384     return TRUE;
03385   }
03386
03387   self->next_frame_data_size = plm_audio_decode_header(self);
03388   return self->has_header;
03389 }
03390
03391 int plm_audio_get_samplerate(plm_audio_t *self) {
03392   return plm_audio_has_header(self) ? PLM_AUDIO_SAMPLE_RATE[self->samplerate_index] : 0;
03393 }
03394
03395 double plm_audio_get_time(plm_audio_t *self) { return self->time; }
03396
03397 void plm_audio_set_time(plm_audio_t *self, double time) {
03398   self->samples_decoded = time * (double)PLM_AUDIO_SAMPLE_RATE[self->samplerate_index];
03399   self->time = time;
03400 }
03401
03402 void plm_audio_rewind(plm_audio_t *self) {
03403   plm_buffer_rewind(self->buffer);
03404   self->time = 0;
03405   self->samples_decoded = 0;
03406   self->next_frame_data_size = 0;
03407 }
03408
03409 int plm_audio_has_ended(plm_audio_t *self) { return plm_buffer_has_ended(self->buffer); }
03410
03411 plm_samples_t *plm_audio_decode(plm_audio_t *self) {
03412   // Do we have at least enough information to decode the frame header?
03413   if (!self->next_frame_data_size) {
03414     if (!plm_buffer_has(self->buffer, 48)) {
03415       return NULL;
03416     }
03417     self->next_frame_data_size = plm_audio_decode_header(self);
03418   }
03419
03420   if (self->next_frame_data_size == 0 || !plm_buffer_has(self->buffer, self->next_frame_data_size «
      3)) {
03421     return NULL;
03422   }
03423
03424   plm_audio_decode_frame(self);
03425   self->next_frame_data_size = 0;
03426
03427   self->samples.time = self->time;
03428
03429   self->samples_decoded += PLM_AUDIO_SAMPLES_PER_FRAME;
03430   self->time = (double)self->samples_decoded / (double)PLM_AUDIO_SAMPLE_RATE[self->samplerate_index];
03431
03432   return &self->samples;
03433 }
03434
03435 int plm_audio_find_frame_sync(plm_audio_t *self) {
03436   size_t i;
03437   for (i = self->buffer->bit_index » 3; i < self->buffer->length - 1; i++) {
03438     if (self->buffer->bytes[i] == 0xFF && (self->buffer->bytes[i + 1] & 0xFE) == 0xFC) {
03439       self->buffer->bit_index = ((i + 1) « 3) + 3;
03440       return TRUE;
03441     }
03442   }
03443   self->buffer->bit_index = (i + 1) « 3;
03444   return FALSE;
03445 }
03446
```

```
03447 int plm_audio_decode_header(plm_audio_t *self) {
03448   if (!plm_buffer_has(self->buffer, 48)) {
03449     return 0;
03450   }
03451
03452   plm_buffer_skip_bytes(self->buffer, 0x00);
03453   int sync = plm_buffer_read(self->buffer, 11);
03454
03455   // Attempt to resync if no syncword was found. This sucks balls. The MP2
03456   // stream contains a syncword just before every frame (11 bits set to 1).
03457   // However, this syncword is not guaranteed to not occur elsewhere in the
03458   // stream. So, if we have to resync, we also have to check if the header
03459   // (samplerate, bitrate) differs from the one we had before. This all
03460   // may still lead to garbage data being decoded :/
03461
03462   if (sync != PLM_AUDIO_FRAME_SYNC && !plm_audio_find_frame_sync(self)) {
03463     return 0;
03464   }
03465
03466   self->version = plm_buffer_read(self->buffer, 2);
03467   self->layer = plm_buffer_read(self->buffer, 2);
03468   int hasCRC = !plm_buffer_read(self->buffer, 1);
03469
03470   if (self->version != PLM_AUDIO_MPEG_1 || self->layer != PLM_AUDIO_LAYER_II) {
03471     return 0;
03472   }
03473
03474   int bitrate_index = plm_buffer_read(self->buffer, 4) - 1;
03475   if (bitrate_index > 13) {
03476     return 0;
03477   }
03478
03479   int samplerate_index = plm_buffer_read(self->buffer, 2);
03480   if (samplerate_index == 3) {
03481     return 0;
03482   }
03483
03484   int padding = plm_buffer_read(self->buffer, 1);
03485   plm_buffer_skip(self->buffer, 1); // f_private
03486   int mode = plm_buffer_read(self->buffer, 2);
03487
03488   // If we already have a header, make sure the samplerate, bitrate and mode
03489   // are still the same, otherwise we might have missed sync.
03490   if (self->has_header &&
03491       (self->bitrate_index != bitrate_index || self->samplerate_index != samplerate_index ||
      self->mode != mode)) {
03492     return 0;
03493   }
03494
03495   self->bitrate_index = bitrate_index;
03496   self->samplerate_index = samplerate_index;
03497   self->mode = mode;
03498   self->has_header = TRUE;
03499
03500   // Parse the mode_extension, set up the stereo bound
03501   if (mode == PLM_AUDIO_MODE_JOINT_STEREO) {
03502     self->bound = (plm_buffer_read(self->buffer, 2) + 1) « 2;
03503   } else {
03504     plm_buffer_skip(self->buffer, 2);
03505     self->bound = (mode == PLM_AUDIO_MODE_MONO) ? 0 : 32;
03506   }
03507
03508   // Discard the last 4 bits of the header and the CRC value, if present
03509   plm_buffer_skip(self->buffer, 4); // copyright(1), original(1), emphasis(2)
03510   if (hasCRC) {
03511     plm_buffer_skip(self->buffer, 16);
03512   }
03513
03514   // Compute frame size, check if we have enough data to decode the whole
03515   // frame.
03516   int bitrate = PLM_AUDIO_BIT_RATE[self->bitrate_index];
03517   int samplerate = PLM_AUDIO_SAMPLE_RATE[self->samplerate_index];
03518   int frame_size = (144000 * bitrate / samplerate) + padding;
03519   return frame_size - (hasCRC ? 6 : 4);
03520 }
03521
03522 void plm_audio_decode_frame(plm_audio_t *self) {
03523   // Prepare the quantizer table lookups
03524   int tab3 = 0;
03525   int sblimit = 0;
03526
03527   int tab1 = (self->mode == PLM_AUDIO_MODE_MONO) ? 0 : 1;
03528   int tab2 = PLM_AUDIO_QUANT_LUT_STEP_1[tab1][self->bitrate_index];
03529   tab3 = QUANT_LUT_STEP_2[tab2][self->samplerate_index];
03530   sblimit = tab3 & 63;
03531   tab3 »= 6;
03532
```

```
03533   if (self->bound > sblimit) {
03534     self->bound = sblimit;
03535   }
03536
03537   // Read the allocation information
03538   for (int sb = 0; sb < self->bound; sb++) {
03539     self->allocation[0][sb] = plm_audio_read_allocation(self, sb, tab3);
03540     self->allocation[1][sb] = plm_audio_read_allocation(self, sb, tab3);
03541   }
03542
03543   for (int sb = self->bound; sb < sblimit; sb++) {
03544     self->allocation[0][sb] = self->allocation[1][sb] = plm_audio_read_allocation(self, sb, tab3);
03545   }
03546
03547   // Read scale factor selector information
03548   int channels = (self->mode == PLM_AUDIO_MODE_MONO) ? 1 : 2;
03549   for (int sb = 0; sb < sblimit; sb++) {
03550     for (int ch = 0; ch < channels; ch++) {
03551       if (self->allocation[ch][sb]) {
03552         self->scale_factor_info[ch][sb] = plm_buffer_read(self->buffer, 2);
03553       }
03554     }
03555     if (self->mode == PLM_AUDIO_MODE_MONO) {
03556       self->scale_factor_info[1][sb] = self->scale_factor_info[0][sb];
03557     }
03558   }
03559
03560   // Read scale factors
03561   for (int sb = 0; sb < sblimit; sb++) {
03562     for (int ch = 0; ch < channels; ch++) {
03563       if (self->allocation[ch][sb]) {
03564         int *sf = self->scale_factor[ch][sb];
03565         switch (self->scale_factor_info[ch][sb]) {
03566         case 0:
03567           sf[0] = plm_buffer_read(self->buffer, 6);
03568           sf[1] = plm_buffer_read(self->buffer, 6);
03569           sf[2] = plm_buffer_read(self->buffer, 6);
03570           break;
03571         case 1:
03572           sf[0] = sf[1] = plm_buffer_read(self->buffer, 6);
03573           sf[2] = plm_buffer_read(self->buffer, 6);
03574           break;
03575         case 2:
03576           sf[0] = sf[1] = sf[2] = plm_buffer_read(self->buffer, 6);
03577           break;
03578         case 3:
03579           sf[0] = plm_buffer_read(self->buffer, 6);
03580           sf[1] = sf[2] = plm_buffer_read(self->buffer, 6);
03581           break;
03582         }
03583       }
03584     }
03585     if (self->mode == PLM_AUDIO_MODE_MONO) {
03586       self->scale_factor[1][sb][0] = self->scale_factor[0][sb][0];
03587       self->scale_factor[1][sb][1] = self->scale_factor[0][sb][1];
03588       self->scale_factor[1][sb][2] = self->scale_factor[0][sb][2];
03589     }
03590   }
03591
03592   // Coefficient input and reconstruction
03593   int out_pos = 0;
03594   for (int part = 0; part < 3; part++) {
03595     for (int granule = 0; granule < 4; granule++) {
03596
03597       // Read the samples
03598       for (int sb = 0; sb < self->bound; sb++) {
03599         plm_audio_read_samples(self, 0, sb, part);
03600         plm_audio_read_samples(self, 1, sb, part);
03601       }
03602       for (int sb = self->bound; sb < sblimit; sb++) {
03603         plm_audio_read_samples(self, 0, sb, part);
03604         self->sample[1][sb][0] = self->sample[0][sb][0];
03605         self->sample[1][sb][1] = self->sample[0][sb][1];
03606         self->sample[1][sb][2] = self->sample[0][sb][2];
03607       }
03608       for (int sb = sblimit; sb < 32; sb++) {
03609         self->sample[0][sb][0] = 0;
03610         self->sample[0][sb][1] = 0;
03611         self->sample[0][sb][2] = 0;
03612         self->sample[1][sb][0] = 0;
03613         self->sample[1][sb][1] = 0;
03614         self->sample[1][sb][2] = 0;
03615       }
03616
03617       // Synthesis loop
03618       for (int p = 0; p < 3; p++) {
03619         // Shifting step
```

```
03620         self->v_pos = (self->v_pos - 64) & 1023;
03621
03622         for (int ch = 0; ch < 2; ch++) {
03623           plm_audio_idct36(self->sample[ch], p, self->V[ch], self->v_pos);
03624
03625           // Build U, windowing, calculate output
03626           memset(self->U, 0, sizeof(self->U));
03627
03628           int d_index = 512 - (self->v_pos >> 1);
03629           int v_index = (self->v_pos % 128) >> 1;
03630           while (v_index < 1024) {
03631             for (int i = 0; i < 32; ++i) {
03632               self->U[i] += self->D[d_index++] * self->V[ch][v_index++];
03633             }
03634
03635             v_index += 128 - 32;
03636             d_index += 64 - 32;
03637           }
03638
03639           d_index -= (512 - 32);
03640           v_index = (128 - 32 + 1024) - v_index;
03641           while (v_index < 1024) {
03642             for (int i = 0; i < 32; ++i) {
03643               self->U[i] += self->D[d_index++] * self->V[ch][v_index++];
03644             }
03645
03646             v_index += 128 - 32;
03647             d_index += 64 - 32;
03648           }
03649
03650 // Output samples
03651 #ifdef PLM_AUDIO_SEPARATE_CHANNELS
03652           float *out_channel = ch == 0 ? self->samples.left : self->samples.right;
03653           for (int j = 0; j < 32; j++) {
03654             out_channel[out_pos + j] = self->U[j] / 2147418112.0f;
03655           }
03656 #else
03657           for (int j = 0; j < 32; j++) {
03658             self->samples.interleaved[((out_pos + j) << 1) + ch] = self->U[j] / 2147418112.0f;
03659           }
03660 #endif
03661         } // End of synthesis channel loop
03662         out_pos += 32;
03663       } // End of synthesis sub-block loop
03664
03665     } // Decoding of the granule finished
03666   }
03667
03668   plm_buffer_align(self->buffer);
03669 }
03670
03671 const plm_quantizer_spec_t *plm_audio_read_allocation(plm_audio_t *self, int sb, int tab3) {
03672   int tab4 = PLM_AUDIO_QUANT_LUT_STEP_3[tab3][sb];
03673   int qtab = PLM_AUDIO_QUANT_LUT_STEP_4[tab4 & 15][plm_buffer_read(self->buffer, tab4 >> 4)];
03674   return qtab ? (&PLM_AUDIO_QUANT_TAB[qtab - 1]) : 0;
03675 }
03676
03677 void plm_audio_read_samples(plm_audio_t *self, int ch, int sb, int part) {
03678   const plm_quantizer_spec_t *q = self->allocation[ch][sb];
03679   int sf = self->scale_factor[ch][sb][part];
03680   int *sample = self->sample[ch][sb];
03681   int val = 0;
03682
03683   if (!q) {
03684     // No bits allocated for this subband
03685     sample[0] = sample[1] = sample[2] = 0;
03686     return;
03687   }
03688
03689   // Resolve scalefactor
03690   if (sf == 63) {
03691     sf = 0;
03692   } else {
03693     int shift = (sf / 3) | 0;
03694     sf = (PLM_AUDIO_SCALEFACTOR_BASE[sf % 3] + ((1 << shift) >> 1)) >> shift;
03695   }
03696
03697   // Decode samples
03698   int adj = q->levels;
03699   if (q->group) {
03700     // Decode grouped samples
03701     val = plm_buffer_read(self->buffer, q->bits);
03702     sample[0] = val % adj;
03703     val /= adj;
03704     sample[1] = val % adj;
03705     sample[2] = val / adj;
03706   } else {
```

```
03707     // Decode direct samples
03708     sample[0] = plm_buffer_read(self->buffer, q->bits);
03709     sample[1] = plm_buffer_read(self->buffer, q->bits);
03710     sample[2] = plm_buffer_read(self->buffer, q->bits);
03711   }
03712
03713   // Postmultiply samples
03714   int scale = 65536 / (adj + 1);
03715   adj = ((adj + 1) » 1) - 1;
03716
03717   val = (adj - sample[0]) * scale;
03718   sample[0] = (val * (sf » 12) + ((val * (sf & 4095) + 2048) » 12)) » 12;
03719
03720   val = (adj - sample[1]) * scale;
03721   sample[1] = (val * (sf » 12) + ((val * (sf & 4095) + 2048) » 12)) » 12;
03722
03723   val = (adj - sample[2]) * scale;
03724   sample[2] = (val * (sf » 12) + ((val * (sf & 4095) + 2048) » 12)) » 12;
03725 }
03726
03727 void plm_audio_idct36(int s[32][3], int ss, float *d, int dp) {
03728   float t01, t02, t03, t04, t05, t06, t07, t08, t09, t10, t11, t12, t13, t14, t15, t16, t17, t18, t19,
      t20, t21, t22,
03729         t23, t24, t25, t26, t27, t28, t29, t30, t31, t32, t33;
03730
03731   t01 = (float)(s[0][ss] + s[31][ss]);
03732   t02 = (float)(s[0][ss] - s[31][ss]) * 0.500602998235f;
03733   t03 = (float)(s[1][ss] + s[30][ss]);
03734   t04 = (float)(s[1][ss] - s[30][ss]) * 0.505470959898f;
03735   t05 = (float)(s[2][ss] + s[29][ss]);
03736   t06 = (float)(s[2][ss] - s[29][ss]) * 0.515447309923f;
03737   t07 = (float)(s[3][ss] + s[28][ss]);
03738   t08 = (float)(s[3][ss] - s[28][ss]) * 0.53104259109f;
03739   t09 = (float)(s[4][ss] + s[27][ss]);
03740   t10 = (float)(s[4][ss] - s[27][ss]) * 0.553103896034f;
03741   t11 = (float)(s[5][ss] + s[26][ss]);
03742   t12 = (float)(s[5][ss] - s[26][ss]) * 0.582934968206f;
03743   t13 = (float)(s[6][ss] + s[25][ss]);
03744   t14 = (float)(s[6][ss] - s[25][ss]) * 0.622504123036f;
03745   t15 = (float)(s[7][ss] + s[24][ss]);
03746   t16 = (float)(s[7][ss] - s[24][ss]) * 0.674808341455f;
03747   t17 = (float)(s[8][ss] + s[23][ss]);
03748   t18 = (float)(s[8][ss] - s[23][ss]) * 0.744536271002f;
03749   t19 = (float)(s[9][ss] + s[22][ss]);
03750   t20 = (float)(s[9][ss] - s[22][ss]) * 0.839349645416f;
03751   t21 = (float)(s[10][ss] + s[21][ss]);
03752   t22 = (float)(s[10][ss] - s[21][ss]) * 0.972568237862f;
03753   t23 = (float)(s[11][ss] + s[20][ss]);
03754   t24 = (float)(s[11][ss] - s[20][ss]) * 1.16943993343f;
03755   t25 = (float)(s[12][ss] + s[19][ss]);
03756   t26 = (float)(s[12][ss] - s[19][ss]) * 1.48416461631f;
03757   t27 = (float)(s[13][ss] + s[18][ss]);
03758   t28 = (float)(s[13][ss] - s[18][ss]) * 2.05778100995f;
03759   t29 = (float)(s[14][ss] + s[17][ss]);
03760   t30 = (float)(s[14][ss] - s[17][ss]) * 3.40760841847f;
03761   t31 = (float)(s[15][ss] + s[16][ss]);
03762   t32 = (float)(s[15][ss] - s[16][ss]) * 10.1900081235f;
03763
03764   t33 = t01 + t31;
03765   t31 = (t01 - t31) * 0.502419286188f;
03766   t01 = t03 + t29;
03767   t29 = (t03 - t29) * 0.52249861494f;
03768   t03 = t05 + t27;
03769   t27 = (t05 - t27) * 0.566944034816f;
03770   t05 = t07 + t25;
03771   t25 = (t07 - t25) * 0.64682178336f;
03772   t07 = t09 + t23;
03773   t23 = (t09 - t23) * 0.788154623451f;
03774   t09 = t11 + t21;
03775   t21 = (t11 - t21) * 1.06067768599f;
03776   t11 = t13 + t19;
03777   t19 = (t13 - t19) * 1.72244709824f;
03778   t13 = t15 + t17;
03779   t17 = (t15 - t17) * 5.10114861869f;
03780   t15 = t33 + t13;
03781   t13 = (t33 - t13) * 0.509795579104f;
03782   t33 = t01 + t11;
03783   t01 = (t01 - t11) * 0.601344886935f;
03784   t11 = t03 + t09;
03785   t09 = (t03 - t09) * 0.899976223136f;
03786   t03 = t05 + t07;
03787   t07 = (t05 - t07) * 2.56291544774f;
03788   t05 = t15 + t03;
03789   t15 = (t15 - t03) * 0.541196100146f;
03790   t03 = t33 + t11;
03791   t11 = (t33 - t11) * 1.30656296488f;
03792   t33 = t05 + t03;
```

```
03793   t05 = (t05 - t03) * 0.707106781187f;
03794   t03 = t15 + t11;
03795   t15 = (t15 - t11) * 0.707106781187f;
03796   t03 += t15;
03797   t11 = t13 + t07;
03798   t13 = (t13 - t07) * 0.541196100146f;
03799   t07 = t01 + t09;
03800   t09 = (t01 - t09) * 1.30656296488f;
03801   t01 = t11 + t07;
03802   t07 = (t11 - t07) * 0.707106781187f;
03803   t11 = t13 + t09;
03804   t13 = (t13 - t09) * 0.707106781187f;
03805   t11 += t13;
03806   t01 += t11;
03807   t11 += t07;
03808   t07 += t13;
03809   t09 = t31 + t17;
03810   t31 = (t31 - t17) * 0.509795579104f;
03811   t17 = t29 + t19;
03812   t29 = (t29 - t19) * 0.601344886935f;
03813   t19 = t27 + t21;
03814   t21 = (t27 - t21) * 0.899976223136f;
03815   t27 = t25 + t23;
03816   t23 = (t25 - t23) * 2.56291544774f;
03817   t25 = t09 + t27;
03818   t09 = (t09 - t27) * 0.541196100146f;
03819   t27 = t17 + t19;
03820   t19 = (t17 - t19) * 1.30656296488f;
03821   t17 = t25 + t27;
03822   t27 = (t25 - t27) * 0.707106781187f;
03823   t25 = t09 + t19;
03824   t19 = (t09 - t19) * 0.707106781187f;
03825   t25 += t19;
03826   t09 = t31 + t23;
03827   t31 = (t31 - t23) * 0.541196100146f;
03828   t23 = t29 + t21;
03829   t21 = (t29 - t21) * 1.30656296488f;
03830   t29 = t09 + t23;
03831   t23 = (t09 - t23) * 0.707106781187f;
03832   t09 = t31 + t21;
03833   t31 = (t31 - t21) * 0.707106781187f;
03834   t09 += t31;
03835   t29 += t09;
03836   t09 += t23;
03837   t23 += t31;
03838   t17 += t29;
03839   t29 += t25;
03840   t25 += t09;
03841   t09 += t27;
03842   t27 += t23;
03843   t23 += t19;
03844   t19 += t31;
03845   t21 = t02 + t32;
03846   t02 = (t02 - t32) * 0.502419286188f;
03847   t32 = t04 + t30;
03848   t04 = (t04 - t30) * 0.52249861494f;
03849   t30 = t06 + t28;
03850   t28 = (t06 - t28) * 0.566944034816f;
03851   t06 = t08 + t26;
03852   t08 = (t08 - t26) * 0.64682178336f;
03853   t26 = t10 + t24;
03854   t10 = (t10 - t24) * 0.788154623451f;
03855   t24 = t12 + t22;
03856   t22 = (t12 - t22) * 1.06067768599f;
03857   t12 = t14 + t20;
03858   t20 = (t14 - t20) * 1.72244709824f;
03859   t14 = t16 + t18;
03860   t16 = (t16 - t18) * 5.10114861869f;
03861   t18 = t21 + t14;
03862   t14 = (t21 - t14) * 0.509795579104f;
03863   t21 = t32 + t12;
03864   t32 = (t32 - t12) * 0.601344886935f;
03865   t12 = t30 + t24;
03866   t24 = (t30 - t24) * 0.899976223136f;
03867   t30 = t06 + t26;
03868   t26 = (t06 - t26) * 2.56291544774f;
03869   t06 = t18 + t30;
03870   t18 = (t18 - t30) * 0.541196100146f;
03871   t30 = t21 + t12;
03872   t12 = (t21 - t12) * 1.30656296488f;
03873   t21 = t06 + t30;
03874   t30 = (t06 - t30) * 0.707106781187f;
03875   t06 = t18 + t12;
03876   t12 = (t18 - t12) * 0.707106781187f;
03877   t06 += t12;
03878   t18 = t14 + t26;
03879   t26 = (t14 - t26) * 0.541196100146f;
```

```
03880   t14 = t32 + t24;
03881   t24 = (t32 - t24) * 1.30656296488f;
03882   t32 = t18 + t14;
03883   t14 = (t18 - t14) * 0.707106781187f;
03884   t18 = t26 + t24;
03885   t24 = (t26 - t24) * 0.707106781187f;
03886   t18 += t24;
03887   t32 += t18;
03888   t18 += t14;
03889   t26 = t14 + t24;
03890   t14 = t02 + t16;
03891   t02 = (t02 - t16) * 0.509795579104f;
03892   t16 = t04 + t20;
03893   t04 = (t04 - t20) * 0.601344886935f;
03894   t20 = t28 + t22;
03895   t22 = (t28 - t22) * 0.899976223136f;
03896   t28 = t08 + t10;
03897   t10 = (t08 - t10) * 2.56291544774f;
03898   t08 = t14 + t28;
03899   t14 = (t14 - t28) * 0.541196100146f;
03900   t28 = t16 + t20;
03901   t20 = (t16 - t20) * 1.30656296488f;
03902   t16 = t08 + t28;
03903   t28 = (t08 - t28) * 0.707106781187f;
03904   t08 = t14 + t20;
03905   t20 = (t14 - t20) * 0.707106781187f;
03906   t08 += t20;
03907   t14 = t02 + t10;
03908   t02 = (t02 - t10) * 0.541196100146f;
03909   t10 = t04 + t22;
03910   t22 = (t04 - t22) * 1.30656296488f;
03911   t04 = t14 + t10;
03912   t10 = (t14 - t10) * 0.707106781187f;
03913   t14 = t02 + t22;
03914   t02 = (t02 - t22) * 0.707106781187f;
03915   t14 += t02;
03916   t04 += t14;
03917   t14 += t10;
03918   t10 += t02;
03919   t16 += t04;
03920   t04 += t08;
03921   t08 += t14;
03922   t14 += t28;
03923   t28 += t10;
03924   t10 += t20;
03925   t20 += t02;
03926   t21 += t16;
03927   t16 += t32;
03928   t32 += t04;
03929   t04 += t06;
03930   t06 += t08;
03931   t08 += t18;
03932   t18 += t14;
03933   t14 += t30;
03934   t30 += t28;
03935   t28 += t26;
03936   t26 += t10;
03937   t10 += t12;
03938   t12 += t20;
03939   t20 += t24;
03940   t24 += t02;
03941
03942   d[dp + 48] = -t33;
03943   d[dp + 49] = d[dp + 47] = -t21;
03944   d[dp + 50] = d[dp + 46] = -t17;
03945   d[dp + 51] = d[dp + 45] = -t16;
03946   d[dp + 52] = d[dp + 44] = -t01;
03947   d[dp + 53] = d[dp + 43] = -t32;
03948   d[dp + 54] = d[dp + 42] = -t29;
03949   d[dp + 55] = d[dp + 41] = -t04;
03950   d[dp + 56] = d[dp + 40] = -t03;
03951   d[dp + 57] = d[dp + 39] = -t06;
03952   d[dp + 58] = d[dp + 38] = -t25;
03953   d[dp + 59] = d[dp + 37] = -t08;
03954   d[dp + 60] = d[dp + 36] = -t11;
03955   d[dp + 61] = d[dp + 35] = -t18;
03956   d[dp + 62] = d[dp + 34] = -t09;
03957   d[dp + 63] = d[dp + 33] = -t14;
03958   d[dp + 32] = -t05;
03959   d[dp + 0] = t05;
03960   d[dp + 31] = -t30;
03961   d[dp + 1] = t30;
03962   d[dp + 30] = -t27;
03963   d[dp + 2] = t27;
03964   d[dp + 29] = -t28;
03965   d[dp + 3] = t28;
03966   d[dp + 28] = -t07;
```

```
03967   d[dp + 4] = t07;
03968   d[dp + 27] = -t26;
03969   d[dp + 5] = t26;
03970   d[dp + 26] = -t23;
03971   d[dp + 6] = t23;
03972   d[dp + 25] = -t10;
03973   d[dp + 7] = t10;
03974   d[dp + 24] = -t15;
03975   d[dp + 8] = t15;
03976   d[dp + 23] = -t12;
03977   d[dp + 9] = t12;
03978   d[dp + 22] = -t19;
03979   d[dp + 10] = t19;
03980   d[dp + 21] = -t20;
03981   d[dp + 11] = t20;
03982   d[dp + 20] = -t13;
03983   d[dp + 12] = t13;
03984   d[dp + 19] = -t24;
03985   d[dp + 13] = t24;
03986   d[dp + 18] = -t31;
03987   d[dp + 14] = t31;
03988   d[dp + 17] = -t02;
03989   d[dp + 15] = t02;
03990   d[dp + 16] = 0.0;
03991 }
03992
03993 #endif // PL_MPEG_IMPLEMENTATION
```

6.5 video.h

```
00001 #include "../core/include/subsystems/screen.h"
00002 #include "pl_mpeg.h"
00003 #include <string>
00004
00006 void set_video(const std::string &filename);
00008 void video_restart();
00009 // plays the video set by set_video()
00010 // because of memory constraints we're limited to one video at a time
00011 class VideoPlayer : public screen::Page {
00012 public:
00013   VideoPlayer();
00014   void update(bool was_pressed, int x, int y) override;
00015
00016   void draw(vex::brain::lcd &screen, bool first_draw, unsigned int frame_number) override;
00017 };
```

6.6 layout.h

```
00001 #include <cmath>
00002 #include <functional>
00003
00004 struct SliderCfg {
00005   double &val;
00006   double min;
00007   double max;
00008 };
```

6.7 lift.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/controls/pid.h"
00004 #include "vex.h"
00005 #include <atomic>
00006 #include <iostream>
00007 #include <map>
00008 #include <vector>
00009
00010 using namespace vex;
00011 using namespace std;
00012
00020 template <typename T> class Lift {
00021 public:
00028   struct lift_cfg_t {
00029     double up_speed, down_speed;
```

```
00030     double softstop_up, softstop_down;
00031
00032     PID::pid_config_t lift_pid_cfg;
00033   };
00034
00056   Lift(motor_group &lift_motors, lift_cfg_t &lift_cfg, map<T, double> &setpoint_map, limit
      *homing_switch = NULL)
00057       : lift_motors(lift_motors), cfg(lift_cfg), lift_pid(cfg.lift_pid_cfg),
      setpoint_map(setpoint_map),
00058         homing_switch(homing_switch) {
00059
00060     is_async = true;
00061     setpoint = 0;
00062
00063     // Create a background task that is constantly updating the lift PID, if requested.
00064     // Set once, and forget.
00065     task t(
00066         [](void *ptr) {
00067           Lift &lift = *((Lift *)ptr);
00068
00069           while (true) {
00070             if (lift.get_async())
00071               lift.hold();
00072
00073             vexDelay(50);
00074           }
00075
00076           return 0;
00077         },
00078         this);
00079   }
00080
00089   void control_continuous(bool up_ctrl, bool down_ctrl) {
00090     static timer tmr;
00091
00092     double cur_pos = 0;
00093
00094     // Check if there's a hook for a custom sensor. If not, use the motors.
00095     if (get_sensor == NULL)
00096       cur_pos = lift_motors.position(rev);
00097     else
00098       cur_pos = get_sensor();
00099
00100     if (up_ctrl && cur_pos < cfg.softstop_up) {
00101       lift_motors.spin(directionType::fwd, cfg.up_speed, volt);
00102       setpoint = cur_pos + .3;
00103
00104       // std::cout << "DEBUG OUT: UP " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
      "\n";
00105
00106       // Disable the PID while going UP.
00107       is_async = false;
00108     } else if (down_ctrl && cur_pos > cfg.softstop_down) {
00109       // Lower the lift slowly, at a rate defined by down_speed
00110       if (setpoint > cfg.softstop_down)
00111         setpoint = setpoint - (tmr.time(sec) * cfg.down_speed);
00112       // std::cout << "DEBUG OUT: DOWN " << setpoint << ", " << tmr.time(sec) << ", " << cfg.down_speed <<
      "\n";
00113       is_async = true;
00114     } else {
00115       // Hold the lift at the last setpoint
00116       is_async = true;
00117     }
00118
00119     tmr.reset();
00120   }
00121
00130   void control_manual(bool up_btn, bool down_btn, int volt_up, int volt_down) {
00131     static bool down_hold = false;
00132     static bool init = true;
00133
00134     // Allow for setting position while still calling this function
00135     if (init || up_btn || down_btn) {
00136       init = false;
00137       is_async = false;
00138     }
00139
00140     double rev = lift_motors.position(rotationUnits::rev);
00141
00142     if (rev < cfg.softstop_down && down_btn)
00143       down_hold = true;
00144     else if (!down_btn)
00145       down_hold = false;
00146
00147     if (up_btn && rev < cfg.softstop_up)
00148       lift_motors.spin(directionType::fwd, volt_up, voltageUnits::volt);
00149     else if (down_btn && rev > cfg.softstop_down && !down_hold)
```

```
00150       lift_motors.spin(directionType::rev, volt_down, voltageUnits::volt);
00151     else
00152       lift_motors.spin(directionType::fwd, 0, voltageUnits::volt);
00153   }
00154
00166   void control_setpoints(bool up_step, bool down_step, vector<T> pos_list) {
00167     // Make sure inputs are only processed on the rising edge of the button
00168     static bool up_last = up_step, down_last = down_step;
00169
00170     bool up_rising = up_step && !up_last;
00171     bool down_rising = down_step && !down_last;
00172
00173     up_last = up_step;
00174     down_last = down_step;
00175
00176     static int cur_index = 0;
00177
00178     // Avoid an index overflow. Shouldn't happen unless the user changes pos_list between calls.
00179     if (cur_index >= pos_list.size())
00180       cur_index = pos_list.size() - 1;
00181
00182     // Increment or decrement the index of the list, bringing it up or down.
00183     if (up_rising && cur_index < (pos_list.size() - 1))
00184       cur_index++;
00185     else if (down_rising && cur_index > 0)
00186       cur_index--;
00187
00188     // Set the lift to hold the position in the background with the PID loop
00189     set_position(pos_list[cur_index]);
00190     is_async = true;
00191   }
00192
00201   bool set_position(T pos) {
00202     this->setpoint = setpoint_map[pos];
00203     is_async = true;
00204
00205     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00206   }
00207
00214   bool set_setpoint(double val) {
00215     this->setpoint = val;
00216     return (lift_pid.get_target() == this->setpoint) && lift_pid.is_on_target();
00217   }
00218
00222   double get_setpoint() { return this->setpoint; }
00223
00228   void hold() {
00229     lift_pid.set_target(setpoint);
00230     // std::cout << "DEBUG OUT: SETPOINT " << setpoint << "\n";
00231
00232     if (get_sensor != NULL)
00233       lift_pid.update(get_sensor());
00234     else
00235       lift_pid.update(lift_motors.position(rev));
00236
00237     // std::cout << "DEBUG OUT: ROTATION " << lift_motors.rotation(rev) << "\n\n";
00238
00239     lift_motors.spin(fwd, lift_pid.get(), volt);
00240   }
00241
00246   void home() {
00247     static timer tmr;
00248     tmr.reset();
00249
00250     while (tmr.time(sec) < 3) {
00251       lift_motors.spin(directionType::rev, 6, volt);
00252
00253       if (homing_switch == NULL && lift_motors.current(currentUnits::amp) > 1.5)
00254         break;
00255       else if (homing_switch != NULL && homing_switch->pressing())
00256         break;
00257     }
00258
00259     if (reset_sensor != NULL)
00260       reset_sensor();
00261
00262     lift_motors.resetPosition();
00263     lift_motors.stop();
00264   }
00265
00269   bool get_async() { return is_async; }
00270
00276   void set_async(bool val) { this->is_async = val; }
00277
00287   void set_sensor_function(double (*fn_ptr)(void)) { this->get_sensor = fn_ptr; }
00288
00295   void set_sensor_reset(void (*fn_ptr)(void)) { this->reset_sensor = fn_ptr; }
```

```
00296
00297 private:
00298   motor_group &lift_motors;
00299   lift_cfg_t &cfg;
00300   PID lift_pid;
00301   map<T, double> &setpoint_map;
00302   limit *homing_switch;
00303
00304   atomic<double> setpoint;
00305   atomic<bool> is_async;
00306
00307   double (*get_sensor)(void) = NULL;
00308   void (*reset_sensor)(void) = NULL;
00309 };
```

6.8 mecanum_drive.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/controls/pid.h"
00004 #include "vex.h"
00005
00006 #ifndef PI
00007 #define PI 3.141592654
00008 #endif
00009
00014 class MecanumDrive {
00015
00016 public:
00020   struct mecanumdrive_config_t {
00021     // PID configurations for autonomous driving
00022     PID::pid_config_t drive_pid_conf;
00023     PID::pid_config_t drive_gyro_pid_conf;
00024     PID::pid_config_t turn_pid_conf;
00025
00026     // Diameter of the mecanum wheels
00027     double drive_wheel_diam;
00028
00029     // Diameter of the perpendicular undriven encoder wheel
00030     double lateral_wheel_diam;
00031
00032     // Width between the center of the left and right wheels
00033     double wheelbase_width;
00034   };
00035
00039   MecanumDrive(vex::motor &left_front, vex::motor &right_front, vex::motor &left_rear, vex::motor
      &right_rear,
00040                vex::rotation *lateral_wheel = NULL, vex::inertial *imu = NULL, mecanumdrive_config_t
      *config = NULL);
00041
00050   void drive_raw(double direction_deg, double magnitude, double rotation);
00051
00062   void drive(double left_y, double left_x, double right_x, int power = 2);
00063
00076   bool auto_drive(double inches, double direction, double speed, bool gyro_correction = true);
00077
00088   bool auto_turn(double degrees, double speed, bool ignore_imu = false);
00089
00090 private:
00091   vex::motor &left_front, &right_front, &left_rear, &right_rear;
00092
00093   mecanumdrive_config_t *config;
00094   vex::rotation *lateral_wheel;
00095   vex::inertial *imu;
00096
00097   PID *drive_pid = NULL;
00098   PID *drive_gyro_pid = NULL;
00099   PID *turn_pid = NULL;
00100
00101   bool init = true;
00102 };
```

6.9 odometry_3wheel.h

```
00001 #pragma once
00002 #include "../core/include/subsystems/custom_encoder.h"
00003 #include "../core/include/subsystems/odometry/odometry_base.h"
00004 #include "../core/include/subsystems/tank_drive.h"
00005
```

```
00032 class Odometry3Wheel : public OdometryBase {
00033 public:
00038   typedef struct {
00039     double wheelbase_dist;
00040     double off_axis_center_dist;
00041     double wheel_diam;
00043   } odometry3wheel_cfg_t;
00044
00054   Odometry3Wheel(CustomEncoder &lside_fwd, CustomEncoder &rside_fwd, CustomEncoder &off_axis,
      odometry3wheel_cfg_t &cfg,
00055                  bool is_async = true);
00056
00063   pose_t update() override;
00064
00073   void tune(vex::controller &con, TankDrive &drive);
00074
00075 private:
00088   static pose_t calculate_new_pos(double lside_delta_deg, double rside_delta_deg, double
      offax_delta_deg,
00089                                   pose_t old_pos, odometry3wheel_cfg_t cfg);
00090
00091   CustomEncoder &lside_fwd, &rside_fwd, &off_axis;
00092   odometry3wheel_cfg_t &cfg;
00093 };
```

6.10 odometry_base.h

```
00001 #pragma once
00002
00003 #include "../core/include/robot_specs.h"
00004 #include "../core/include/utils/command_structure/auto_command.h"
00005 #include "../core/include/utils/geometry.h"
00006 #include "vex.h"
00007
00008 #ifndef PI
00009 #define PI 3.141592654
00010 #endif
00011
00024 class OdometryBase {
00025 public:
00031   OdometryBase(bool is_async);
00032
00037   pose_t get_position(void);
00038
00043   virtual void set_position(const pose_t &newpos = zero_pos);
00044   AutoCommand *SetPositionCmd(const pose_t &newpos = zero_pos);
00049   virtual pose_t update() = 0;
00050
00058   static int background_task(void *ptr);
00059
00065   void end_async();
00066
00073   static double pos_diff(pose_t start_pos, pose_t end_pos);
00074
00081   static double rot_diff(pose_t pos1, pose_t pos2);
00082
00092   static double smallest_angle(double start_deg, double end_deg);
00093
00095   bool end_task = false;
00096
00101   double get_speed();
00102
00107   double get_accel();
00108
00113   double get_angular_speed_deg();
00114
00119   double get_angular_accel_deg();
00120
00124   inline static constexpr pose_t zero_pos = {.x = 0.0L, .y = 0.0L, .rot = 90.0L};
00125
00126 protected:
00130   vex::task *handle;
00131
00135   vex::mutex mut;
00136
00140   pose_t current_pos;
00141
00142   double speed;
00143   double accel;
00144   double ang_speed_deg;
00145   double ang_accel_deg;
00146 };
```

6.11 odometry_tank.h

```
00001 #pragma once
00002
00003 #include "../core/include/subsystems/custom_encoder.h"
00004 #include "../core/include/subsystems/odometry/odometry_base.h"
00005 #include "../core/include/utils/geometry.h"
00006 #include "../core/include/utils/moving_average.h"
00007 #include "../core/include/utils/vector2d.h"
00008
00009 #include "../core/include/robot_specs.h"
00010
00011 static int background_task(void *odom_obj);
00012
00019 class OdometryTank : public OdometryBase {
00020 public:
00031   OdometryTank(vex::motor_group &left_side, vex::motor_group &right_side, robot_specs_t &config,
00032                vex::inertial *imu = NULL, bool is_async = true);
00033
00045   OdometryTank(CustomEncoder &left_custom_enc, CustomEncoder &right_custom_enc, robot_specs_t &config,
00046                vex::inertial *imu = NULL, bool is_async = true);
00047
00059   OdometryTank(vex::encoder &left_vex_enc, vex::encoder &right_vex_enc, robot_specs_t &config,
00060                vex::inertial *imu = NULL, bool is_async = true);
00061
00066   pose_t update() override;
00067
00072   void set_position(const pose_t &newpos = zero_pos) override;
00073
00074 private:
00078   static pose_t calculate_new_pos(robot_specs_t &config, pose_t &stored_info, double lside_diff,
      double rside_diff,
00079                                   double angle_deg);
00080
00081   vex::motor_group *left_side, *right_side;
00082   CustomEncoder *left_custom_enc, *right_custom_enc;
00083   vex::encoder *left_vex_enc, *right_vex_enc;
00084   vex::inertial *imu;
00085   robot_specs_t &config;
00086
00087   double rotation_offset = 0;
00088   ExponentialMovingAverage ema = ExponentialMovingAverage(3);
00089 };
```

6.12 screen.h

```
00001 #pragma once
00002 #include "../core/include/subsystems/odometry/odometry_base.h"
00003 #include "../core/include/utils/controls/pid.h"
00004 #include "../core/include/utils/controls/pidff.h"
00005 #include "../core/include/utils/graph_drawer.h"
00006 #include "vex.h"
00007 #include <cassert>
00008 #include <functional>
00009 #include <map>
00010 #include <vector>
00011
00012 namespace screen {
00014 class ButtonWidget {
00015 public:
00020   ButtonWidget(std::function<void(void)> onpress, Rect rect, std::string name)
00021       : onpress(onpress), rect(rect), name(name) {}
00026   ButtonWidget(void (*onpress)(), Rect rect, std::string name) : onpress(onpress), rect(rect),
      name(name) {}
00027
00033   bool update(bool was_pressed, int x, int y);
00035   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00036
00037 private:
00038   std::function<void(void)> onpress;
00039   Rect rect;
00040   std::string name = "";
00041   bool was_pressed_last = false;
00042 };
00043
00046 class SliderWidget {
00047 public:
00054   SliderWidget(double &val, double low, double high, Rect rect, std::string name)
00055       : value(val), low(low), high(high), rect(rect), name(name) {}
00056
00062   bool update(bool was_pressed, int x, int y);
00064   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00065
```

```
00066 private:
00067   double &value;
00068
00069   double low;
00070   double high;
00071
00072   Rect rect;
00073   std::string name = "";
00074 };
00075
00076 struct WidgetConfig;
00077
00078 struct SliderConfig {
00079   double &val;
00080   double low;
00081   double high;
00082 };
00083 struct ButtonConfig {
00084   std::function<void()> onclick;
00085 };
00086 struct CheckboxConfig {
00087   std::function<void(bool)> onupdate;
00088 };
00089 struct LabelConfig {
00090   std::string label;
00091 };
00092
00093 struct TextConfig {
00094   std::function<std::string()> text;
00095 };
00096 struct SizedWidget {
00097   int size;
00098   WidgetConfig &widget;
00099 };
00100 struct WidgetConfig {
00101   enum Type {
00102     Col,
00103     Row,
00104     Slider,
00105     Button,
00106     Checkbox,
00107     Label,
00108     Text,
00109     Graph,
00110   };
00111   Type type;
00112   union {
00113     std::vector<SizedWidget> widgets;
00114     SliderConfig slider;
00115     ButtonConfig button;
00116     CheckboxConfig checkbox;
00117     LabelConfig label;
00118     TextConfig text;
00119     GraphDrawer *graph;
00120   } config;
00121 };
00122
00123 class Page;
00125 class Page {
00126 public:
00135   virtual void update(bool was_pressed, int x, int y);
00143   virtual void draw(vex::brain::lcd &screen, bool first_draw, unsigned int frame_number);
00144 };
00145
00146 struct ScreenRect {
00147   uint32_t x1;
00148   uint32_t y1;
00149   uint32_t x2;
00150   uint32_t y2;
00151 };
00152 void draw_widget(WidgetConfig &widget, ScreenRect rect);
00153
00154 class WidgetPage : public Page {
00155 public:
00156   WidgetPage(WidgetConfig &cfg) : base_widget(cfg) {}
00157   void update(bool was_pressed, int x, int y) override;
00158
00159   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override {
00160     draw_widget(base_widget, {.x1 = 20, .y1 = 0, .x2 = 440, .y2 = 240});
00161   }
00162
00163 private:
00164   WidgetConfig &base_widget;
00165 };
00166
00173 void start_screen(vex::brain::lcd &screen, std::vector<Page *> pages, int first_page = 0);
00174
```

```
00175 void next_page();
00176 void prev_page();
00177 void goto_page(size_t page);
00178
00180 void stop_screen();
00181
00183 using update_func_t = std::function<void(bool, int, int)>;
00184
00186 using draw_func_t = std::function<void(vex::brain::lcd &screen, bool, unsigned int)>;
00187
00189 class StatsPage : public Page {
00190 public:
00193   StatsPage(std::map<std::string, vex::motor &> motors);
00195   void update(bool was_pressed, int x, int y) override;
00197   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00198
00199 private:
00200   void draw_motor_stats(const std::string &name, vex::motor &mot, unsigned int frame, int x, int y,
00201                         vex::brain::lcd &scr);
00202
00203   std::map<std::string, vex::motor &> motors;
00204   static const int y_start = 0;
00205   static const int per_column = 4;
00206   static const int row_height = 20;
00207   static const int row_width = 200;
00208 };
00209
00213 class OdometryPage : public Page {
00214 public:
00221   OdometryPage(OdometryBase &odom, double robot_width, double robot_height, bool do_trail);
00223   void update(bool was_pressed, int x, int y) override;
00225   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00226
00227 private:
00228   static const int path_len = 40;
00229   static constexpr char const *field_filename = "vex_field_240p.png";
00230
00231   OdometryBase &odom;
00232   double robot_width;
00233   double robot_height;
00234   uint8_t *buf = nullptr;
00235   int buf_size = 0;
00236   pose_t path[path_len];
00237   int path_index = 0;
00238   bool do_trail;
00239   GraphDrawer velocity_graph;
00240 };
00241
00244 class FunctionPage : public Page {
00245 public:
00249   FunctionPage(update_func_t update_f, draw_func_t draw_t);
00251   void update(bool was_pressed, int x, int y) override;
00253   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00254
00255 private:
00256   update_func_t update_f;
00257   draw_func_t draw_f;
00258 };
00259
00261 class PIDPage : public Page {
00262 public:
00268   PIDPage(
00269       PID &pid, std::string name, std::function<void(void)> onchange = [](){});
00270   PIDPage(
00271       PIDFF &pidff, std::string name, std::function<void(void)> onchange = [](){});
00272
00274   void update(bool was_pressed, int x, int y) override;
00276   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number) override;
00277
00278 private:
00280   void zero_d_f() { cfg.d = 0; }
00282   void zero_i_f() { cfg.i = 0; }
00283
00284   PID::pid_config_t &cfg;
00285   PID &pid;
00286   const std::string name;
00287   std::function<void(void)> onchange;
00288
00289   SliderWidget p_slider;
00290   SliderWidget i_slider;
00291   SliderWidget d_slider;
00292   ButtonWidget zero_i;
00293   ButtonWidget zero_d;
00294
00295   GraphDrawer graph;
00296 };
00297
```

```
00298 } // namespace screen
```

6.13 tank_drive.h

```
00001 #pragma once
00002
00003 #ifndef PI
00004 #define PI 3.141592654
00005 #endif
00006
00007 #include "../core/include/robot_specs.h"
00008 #include "../core/include/subsystems/odometry/odometry_tank.h"
00009 #include "../core/include/utils/command_structure/auto_command.h"
00010 #include "../core/include/utils/controls/feedback_base.h"
00011 #include "../core/include/utils/controls/pid.h"
00012 #include "../core/include/utils/pure_pursuit.h"
00013 #include "vex.h"
00014 #include <vector>
00015
00016 using namespace vex;
00017
00023 class TankDrive {
00024 public:
00025   enum class BrakeType {
00026     None,
00027     ZeroVelocity,
00028     Smart,
00029   };
00038   TankDrive(motor_group &left_motors, motor_group &right_motors, robot_specs_t &config, OdometryBase
      *odom = NULL);
00039
00040   AutoCommand *DriveToPointCmd(point_t pt, vex::directionType dir = vex::forward, double max_speed =
      1.0,
00041                                double end_speed = 0.0);
00042   AutoCommand *DriveToPointCmd(Feedback &fb, point_t pt, vex::directionType dir = vex::forward, double
      max_speed = 1.0,
00043                                double end_speed = 0.0);
00044
00045   AutoCommand *DriveForwardCmd(double dist, vex::directionType dir = vex::forward, double max_speed =
      1.0,
00046                                double end_speed = 0.0);
00047   AutoCommand *DriveForwardCmd(Feedback &fb, double dist, vex::directionType dir = vex::forward,
      double max_speed = 1.0,
00048                                double end_speed = 0.0);
00049
00050   AutoCommand *TurnToHeadingCmd(double heading, double max_speed = 1.0, double end_speed = 0.0);
00051   AutoCommand *TurnToHeadingCmd(Feedback &fb, double heading, double max_speed = 1.0, double end_speed
      = 0.0);
00052
00053   AutoCommand *TurnToPointCmd(double x, double y, vex::directionType dir = vex::directionType::fwd,
00054                               double max_speed = 1.0, double end_speed = 0.0);
00055
00056   AutoCommand *TurnDegreesCmd(double degrees, double max_speed = 1.0, double start_speed = 0.0);
00057   AutoCommand *TurnDegreesCmd(Feedback &fb, double degrees, double max_speed = 1.0, double end_speed =
      0.0);
00058
00059   AutoCommand *PurePursuitCmd(PurePursuit::Path path, directionType dir, double max_speed = 1, double
      end_speed = 0);
00060   AutoCommand *PurePursuitCmd(Feedback &feedback, PurePursuit::Path path, directionType dir, double
      max_speed = 1,
00061                               double end_speed = 0);
00062   Condition *DriveStalledCondition(double stall_time);
00063   AutoCommand *DriveTankCmd(double left, double right);
00064
00068   void stop();
00069
00080   void drive_tank(double left, double right, int power = 1, BrakeType bt = BrakeType::None);
00086   void drive_tank_raw(double left, double right);
00087
00099   void drive_arcade(double forward_back, double left_right, int power = 1, BrakeType bt =
      BrakeType::None);
00100
00114   bool drive_forward(double inches, directionType dir, Feedback &feedback, double max_speed = 1,
      double end_speed = 0);
00115
00128   bool drive_forward(double inches, directionType dir, double max_speed = 1, double end_speed = 0);
00129
00141   bool turn_degrees(double degrees, Feedback &feedback, double max_speed = 1, double end_speed = 0);
00142
00156   bool turn_degrees(double degrees, double max_speed = 1, double end_speed = 0);
00157
00171   bool drive_to_point(double x, double y, vex::directionType dir, Feedback &feedback, double max_speed
      = 1,
```

```
00172                       double end_speed = 0);
00173
00188   bool drive_to_point(double x, double y, vex::directionType dir, double max_speed = 1, double
      end_speed = 0);
00189
00200   bool turn_to_heading(double heading_deg, Feedback &feedback, double max_speed = 1, double end_speed
      = 0);
00211   bool turn_to_heading(double heading_deg, double max_speed = 1, double end_speed = 0);
00212
00216   void reset_auto();
00217
00228   static double modify_inputs(double input, int power = 2);
00229
00244   bool pure_pursuit(PurePursuit::Path path, directionType dir, Feedback &feedback, double max_speed =
      1,
00245                     double end_speed = 0);
00246
00262   bool pure_pursuit(PurePursuit::Path path, directionType dir, double max_speed = 1, double end_speed
      = 0);
00263
00264 private:
00265   motor_group &left_motors;
00266   motor_group &right_motors;
00267
00268   PID correction_pid;
00270   Feedback *drive_default_feedback = NULL;
00271   Feedback *turn_default_feedback = NULL;
00272
00273   OdometryBase *odometry;
00275
00276   robot_specs_t
00277       &config;
00278
00279   bool func_initialized = false;
00281   bool is_pure_pursuit = false;
00282 };
```

6.14 auto_chooser.h

```
00001 #pragma once
00002 #include "../core/include/subsystems/screen.h"
00003 #include "../core/include/utils/geometry.h"
00004 #include "vex.h"
00005 #include <string>
00006 #include <vector>
00007
00017 class AutoChooser : public screen::Page {
00018 public:
00024   AutoChooser(std::vector<std::string> paths, size_t def = 0);
00025
00026   void update(bool was_pressed, int x, int y);
00027   void draw(vex::brain::lcd &, bool first_draw, unsigned int frame_number);
00028
00033   size_t get_choice();
00034
00035 protected:
00039   struct entry_t {
00040     Rect rect;
00041     std::string name;
00042   };
00043
00044   static const size_t width = 380;
00045   static const size_t height = 220;
00046
00047   size_t choice;
00048   std::vector<entry_t> list ;
00049 };
```

6.15 auto_command.h

```
00001
00007 #pragma once
00008
00009 #include "vex.h"
00010 #include <atomic>
00011 #include <functional>
00012 #include <queue>
00013 #include <vector>
00014
```

```
00024 class Condition {
00025 public:
00026   Condition *Or(Condition *b);
00027   Condition *And(Condition *b);
00028   virtual bool test() = 0;
00029 };
00030
00031 class AutoCommand {
00032 public:
00033   static constexpr double default_timeout = 10.0;
00039   virtual bool run() { return true; }
00043   virtual void on_timeout() {}
00044   AutoCommand *withTimeout(double t_seconds) {
00045     if (this->timeout_seconds < 0) {
00046       // should never be timed out
00047       return this;
00048     }
00049     this->timeout_seconds = t_seconds;
00050     return this;
00051   }
00052   AutoCommand *withCancelCondition(Condition *true_to_end) {
00053     this->true_to_end = true_to_end;
00054     return this;
00055   }
00065   double timeout_seconds = default_timeout;
00066   Condition *true_to_end = nullptr;
00067 };
00068
00073 class FunctionCommand : public AutoCommand {
00074 public:
00075   FunctionCommand(std::function<bool(void)> f) : f(f) {}
00076   bool run() { return f(); }
00077
00078 private:
00079   std::function<bool(void)> f;
00080 };
00081
00082 // Times tested 3
00083 // Test 1 -> false
00084 // Test 2 -> false
00085 // Test 3 -> true
00086 // Returns false until the Nth time that it is called
00087 // This is pretty much only good for implementing RepeatUntil
00088 class TimesTestedCondition : public Condition {
00089 public:
00090   TimesTestedCondition(size_t N) : max(N) {}
00091   bool test() override {
00092     count++;
00093     if (count >= max) {
00094       return true;
00095     }
00096     return false;
00097   }
00098
00099 private:
00100   size_t count = 0;
00101   size_t max;
00102 };
00103
00105 class FunctionCondition : public Condition {
00106 public:
00107   FunctionCondition(
00108       std::function<bool()> cond, std::function<void(void)> timeout = []() {})
00109       : cond(cond), timeout(timeout) {}
00110   bool test() override;
00111
00112 private:
00113   std::function<bool()> cond;
00114   std::function<void(void)> timeout;
00115 };
00116
00119 class IfTimePassed : public Condition {
00120 public:
00121   IfTimePassed(double time_s);
00122   bool test() override;
00123
00124 private:
00125   double time_s;
00126   vex::timer tmr;
00127 };
00128
00130 class WaitUntilCondition : public AutoCommand {
00131 public:
00132   WaitUntilCondition(Condition *cond) : cond(cond) {}
00133   bool run() override { return cond->test(); }
00134
00135 private:
```

```
00136   Condition *cond;
00137 };
00138
00141
00144 class InOrder : public AutoCommand {
00145 public:
00146   InOrder(const InOrder &other) = default;
00147   InOrder(std::queue<AutoCommand *> cmds);
00148   InOrder(std::initializer_list<AutoCommand *> cmds);
00149   bool run() override;
00150   void on_timeout() override;
00151
00152 private:
00153   AutoCommand *current_command = nullptr;
00154   std::queue<AutoCommand *> cmds;
00155   vex::timer tmr;
00156 };
00157
00160 class Parallel : public AutoCommand {
00161 public:
00162   Parallel(std::initializer_list<AutoCommand *> cmds);
00163   bool run() override;
00164   void on_timeout() override;
00165
00166 private:
00167   std::vector<AutoCommand *> cmds;
00168   std::vector<vex::task *> runners;
00169 };
00170
00174 class Branch : public AutoCommand {
00175 public:
00176   Branch(Condition *cond, AutoCommand *false_choice, AutoCommand *true_choice);
00177   ~Branch();
00178   bool run() override;
00179   void on_timeout() override;
00180
00181 private:
00182   AutoCommand *false_choice;
00183   AutoCommand *true_choice;
00184   Condition *cond;
00185   bool choice = false;
00186   bool chosen = false;
00187   vex::timer tmr;
00188 };
00189
00193 class Async : public AutoCommand {
00194 public:
00195   Async(AutoCommand *cmd) : cmd(cmd) {}
00196   bool run() override;
00197
00198 private:
00199   AutoCommand *cmd = nullptr;
00200 };
00201
00202 class RepeatUntil : public AutoCommand {
00203 public:
00207   RepeatUntil(InOrder cmds, size_t repeats);
00211   RepeatUntil(InOrder cmds, Condition *true_to_end);
00212   bool run() override;
00213   void on_timeout() override;
00214
00215 private:
00216   const InOrder cmds;
00217   InOrder *working_cmds;
00218   Condition *cond;
00219 };
```

6.16 basic_command.h

```
00001
00014 #pragma once
00015
00016 #include "../core/include/utils/command_structure/auto_command.h"
00017
00018 // Basic Motor Classes-----------------------------------------------
00019
00024 class BasicSpinCommand : public AutoCommand {
00025 public:
00026   // Enumurator for the type of power setting in the motor
00027   enum type { percent, voltage, veocity };
00028
00037   BasicSpinCommand(vex::motor &motor, vex::directionType dir, BasicSpinCommand::type setting, double
      power);
```

```
00038
00045   bool run() override;
00046
00047 private:
00048   vex::motor &motor;
00049
00050   type setting;
00051
00052   vex::directionType dir;
00053
00054   double power;
00055 };
00060 class BasicStopCommand : public AutoCommand {
00061 public:
00068   BasicStopCommand(vex::motor &motor, vex::brakeType setting);
00069
00076   bool run() override;
00077
00078 private:
00079   vex::motor &motor;
00080
00081   vex::brakeType setting;
00082 };
00083
00084 // Basic Solenoid Commands---------------------------------
00085
00090 class BasicSolenoidSet : public AutoCommand {
00091 public:
00098   BasicSolenoidSet(vex::pneumatics &solenoid, bool setting);
00099
00106   bool run() override;
00107
00108 private:
00109   vex::pneumatics &solenoid;
00110
00111   bool setting;
00112 };
```

6.17 command_controller.h

```
00001
00010 #pragma once
00011 #include "../core/include/utils/command_structure/auto_command.h"
00012 #include <queue>
00013 #include <vector>
00014
00015 class CommandController {
00016 public:
00019   [[deprecated("Empty constructor is bad. Use list constructor "
00020                "instead.")]] CommandController()
00021       : command_queue({}) {}
00022
00026   CommandController(std::initializer_list<AutoCommand *> cmds) : command_queue(cmds) {}
00033   [[deprecated("Use list constructor instead. If you need to make a decision before adding new
      commands, use Branch "
00034                "(https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]] void
00035   add(std::vector<AutoCommand *> cmds);
00036   void add(AutoCommand *cmd, double timeout_seconds = 10.0);
00037
00049   [[deprecated("Use list constructor instead. If you need to make a decision before adding new
      commands, use Branch "
00050                "(https://github.com/RIT-VEX-U/Core/wiki/3-%7C-Utilites#commandcontroller)")]] void
00051   add(std::vector<AutoCommand *> cmds, double timeout_sec);
00058   void add_delay(int ms);
00059
00064   void add_cancel_func(std::function<bool(void)> true_if_cancel);
00065
00070   void run();
00071
00079   bool last_command_timed_out();
00080
00081 private:
00082   std::queue<AutoCommand *> command_queue;
00083   bool command_timed_out = false;
00084   std::function<bool()> should_cancel = []() { return false; };
00085 };
```

6.18 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 };
```

6.19 drive_commands.h

```
00001
00019 #pragma once
00020
00021 #include "../core/include/subsystems/tank_drive.h"
00022 #include "../core/include/utils/command_structure/auto_command.h"
00023 #include "../core/include/utils/geometry.h"
00024 #include "vex.h"
00025
00026 using namespace vex;
00027
00028 // ==== DRIVING ====
00029
00035 class DriveForwardCommand : public AutoCommand {
00036 public:
00037   DriveForwardCommand(TankDrive &drive_sys, Feedback &feedback, double inches, directionType dir,
      double max_speed = 1,
00038                       double end_speed = 0);
00039
00045   bool run() override;
00049   void on_timeout() override;
00050
00051 private:
00052   // drive system to run the function on
00053   TankDrive &drive_sys;
00054
00055   // feedback controller to use
00056   Feedback &feedback;
00057
00058   // parameters for drive_forward
00059   double inches;
00060   directionType dir;
00061   double max_speed;
00062   double end_speed;
00063 };
00064
00069 class TurnDegreesCommand : public AutoCommand {
00070 public:
00071   TurnDegreesCommand(TankDrive &drive_sys, Feedback &feedback, double degrees, double max_speed = 1,
00072                      double end_speed = 0);
00073
00079   bool run() override;
00083   void on_timeout() override;
00084
00085 private:
00086   // drive system to run the function on
00087   TankDrive &drive_sys;
00088
00089   // feedback controller to use
00090   Feedback &feedback;
00091
00092   // parameters for turn_degrees
00093   double degrees;
00094   double max_speed;
00095   double end_speed;
00096 };
00097
00102 class DriveToPointCommand : public AutoCommand {
00103 public:
00104   DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, double x, double y, directionType dir,
00105                       double max_speed = 1, double end_speed = 0);
00106   DriveToPointCommand(TankDrive &drive_sys, Feedback &feedback, point_t point, directionType dir,
      double max_speed = 1,
00107                       double end_speed = 0);
```

```
00108
00114   bool run() override;
00115
00116 private:
00117   // drive system to run the function on
00118   TankDrive &drive_sys;
00119
00123   void on_timeout() override;
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   double end_speed;
00134 };
00135
00141 class TurnToHeadingCommand : public AutoCommand {
00142 public:
00143   TurnToHeadingCommand(TankDrive &drive_sys, Feedback &feedback, double heading_deg, double speed = 1,
00144                        double end_speed = 0);
00145
00151   bool run() override;
00155   void on_timeout() override;
00156
00157 private:
00158   // drive system to run the function on
00159   TankDrive &drive_sys;
00160
00161   // feedback controller to use
00162   Feedback &feedback;
00163
00164   // parameters for turn_to_heading
00165   double heading_deg;
00166   double max_speed;
00167   double end_speed;
00168 };
00169
00173 class PurePursuitCommand : public AutoCommand {
00174 public:
00183   PurePursuitCommand(TankDrive &drive_sys, Feedback &feedback, PurePursuit::Path path, directionType
      dir,
00184                      double max_speed = 1, double end_speed = 0);
00185
00189   bool run() override;
00190
00194   void on_timeout() override;
00195
00196 private:
00197   TankDrive &drive_sys;
00198   PurePursuit::Path path;
00199   directionType dir;
00200   Feedback &feedback;
00201   double max_speed;
00202   double end_speed;
00203 };
00204
00209 class DriveStopCommand : public AutoCommand {
00210 public:
00211   DriveStopCommand(TankDrive &drive_sys);
00212
00218   bool run() override;
00219   void on_timeout() override;
00220
00221 private:
00222   // drive system to run the function on
00223   TankDrive &drive_sys;
00224 };
00225
00226 // ==== ODOMETRY ====
00227
00232 class OdomSetPosition : public AutoCommand {
00233 public:
00239   OdomSetPosition(OdometryBase &odom, const pose_t &newpos = OdometryBase::zero_pos);
00240
00246   bool run() override;
00247
00248 private:
00249   // drive system with an odometry config
00250   OdometryBase &odom;
00251   pose_t newpos;
00252 };
```

6.20 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 spin_rpm
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;
00088
00089 private:
00090   // Flywheel instance to run the function on
00091   Flywheel &flywheel;
00092 };
00093
00099 class FlywheelStopMotorsCommand : public AutoCommand {
00100 public:
00105   FlywheelStopMotorsCommand(Flywheel &flywheel);
00106
00112   bool run() override;
00113
00114 private:
00115   // Flywheel instance to run the function on
00116   Flywheel &flywheel;
00117 };
00118
00124 class FlywheelStopNonTasksCommand : public AutoCommand {
00125   FlywheelStopNonTasksCommand(Flywheel &flywheel);
00126
00132   bool run() override;
00133
00134 private:
00135   // Flywheel instance to run the function on
00136   Flywheel &flywheel;
00137 };
```

6.21 bang_bang.h

```
00001 #include "../core/include/utils/controls/feedback_base.h"
00002
00003 class BangBang : public Feedback {
00004
00005 public:
00006   BangBang(double thresshold, double low, double high);
00015   void init(double start_pt, double set_pt) override;
00016
00023   double update(double val) override;
00024
00028   double get() override;
00029
00036   void set_limits(double lower, double upper) override;
```

```
00037
00041   bool is_on_target() override;
00042
00043 private:
00044   double setpt;
00045   double sensor_val;
00046   double lower_bound, upper_bound;
00047   double last_output;
00048   double threshhold;
00049 };
```

6.22 feedback_base.h

```
00001 #pragma once
00002
00010 class Feedback {
00011 public:
00020   virtual void init(double start_pt, double set_pt) = 0;
00021
00028   virtual double update(double val) = 0;
00029
00033   virtual double get() = 0;
00034
00041   virtual void set_limits(double lower, double upper) = 0;
00042
00046   virtual bool is_on_target() = 0;
00047 };
```

6.23 feedforward.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/math_util.h"
00004 #include "../core/include/utils/moving_average.h"
00005 #include "vex.h"
00006 #include <math.h>
00007 #include <vector>
00008
00029 class FeedForward {
00030 public:
00039   typedef struct {
00040     double kS;
00041     double kV;
00043     double kA;
00045     double kG;
00047   } ff_config_t;
00048
00053   FeedForward(ff_config_t &cfg) : cfg(cfg) {}
00054
00065   double calculate(double v, double a, double pid_ref = 0.0) {
00066     double ks_sign = 0;
00067     if (v != 0)
00068       ks_sign = sign(v);
00069     else if (pid_ref != 0)
00070       ks_sign = sign(pid_ref);
00071
00072     return (cfg.kS * ks_sign) + (cfg.kV * v) + (cfg.kA * a) + cfg.kG;
00073   }
00074
00075 private:
00076   ff_config_t &cfg;
00077 };
00078
00086 FeedForward::ff_config_t tune_feedforward(vex::motor_group &motor, double pct, double duration);
```

6.24 motion_controller.h

```
00001 #pragma once
00002 #include "../core/include/subsystems/screen.h"
00003 #include "../core/include/subsystems/tank_drive.h"
00004 #include "../core/include/utils/controls/feedback_base.h"
00005 #include "../core/include/utils/controls/feedforward.h"
00006 #include "../core/include/utils/controls/pid.h"
00007 #include "../core/include/utils/controls/trapezoid_profile.h"
00008 #include "vex.h"
```

```
00009
00026 class MotionController : public Feedback {
00027 public:
00034   typedef struct {
00035     double max_v;
00036     double accel;
00037     PID::pid_config_t pid_cfg;
00038     FeedForward::ff_config_t ff_cfg;
00039   } m_profile_cfg_t;
00040
00050   MotionController(m_profile_cfg_t &config);
00051
00056   void init(double start_pt, double end_pt) override;
00057
00064   double update(double sensor_val) override;
00065
00069   double get() override;
00070
00078   void set_limits(double lower, double upper) override;
00079
00084   bool is_on_target() override;
00085
00089   motion_t get_motion() const;
00090
00091   screen::Page *Page();
00092
00111   static FeedForward::ff_config_t tune_feedforward(TankDrive &drive, OdometryTank &odometry, double
      pct = 0.6,
00112                                                    double duration = 2);
00113
00114 private:
00115   m_profile_cfg_t config;
00116
00117   PID pid;
00118   FeedForward ff;
00119   TrapezoidProfile profile;
00120
00121   double current_pos;
00122   double end_pt;
00123
00124   double lower_limit = 0, upper_limit = 0;
00125   double out = 0;
00126   motion_t cur_motion;
00127
00128   vex::timer tmr;
00129   friend class MotionControllerPage;
00130 };
```

6.25 pid.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/controls/feedback_base.h"
00004 #include "vex.h"
00005 #include <cmath>
00006
00007 using namespace vex;
00008
00023 class PID : public Feedback {
00024 public:
00029   enum ERROR_TYPE {
00030     LINEAR,
00031     ANGULAR // assumes degrees
00032   };
00043   struct pid_config_t {
00044     double p;
00045     double i;
00046     double d;
00047     double deadband;
00048     double on_target_time;
00050     ERROR_TYPE error_method;
00052   };
00053
00058   PID(pid_config_t &config);
00059
00072   void init(double start_pt, double set_pt) override;
00073
00081   double update(double sensor_val) override;
00082
00092   double update(double sensor_val, double v_setpt);
00093
00098   double get_sensor_val() const;
00099
```

```
00105   double get() override;
00106
00115   void set_limits(double lower, double upper) override;
00116
00121   bool is_on_target() override;
00122
00126   void reset();
00127
00133   double get_error();
00134
00139   double get_target() const;
00140
00145   void set_target(double target);
00146
00147   pid_config_t &config;
00149
00150 private:
00151   double last_error = 0;
00152   double accum_error = 0;
00153
00154   double last_time = 0;
00155   double on_target_last_time = 0;
00156
00157   double lower_limit = 0;
00158   double upper_limit = 0;
00159
00160   double target = 0;
00162   double target_vel = 0;
00164   double sensor_val = 0;
00166   double out = 0;
00169
00170   bool is_checking_on_target = false;
00171
00172   timer pid_timer;
00175 };
```

6.26 pidff.h

```
00001 #pragma once
00002 #include "../core/include/utils/controls/feedback_base.h"
00003 #include "../core/include/utils/controls/feedforward.h"
00004 #include "../core/include/utils/controls/pid.h"
00005
00006 class PIDFF : public Feedback {
00007 public:
00008   PIDFF(PID::pid_config_t &pid_cfg, FeedForward::ff_config_t &ff_cfg);
00009
00018   void init(double start_pt, double set_pt) override;
00019
00024   void set_target(double set_pt);
00025
00026   double get_target() const;
00027   double get_sensor_val() const;
00035   double update(double val) override;
00036
00045   double update(double val, double vel_setpt, double a_setpt = 0);
00046
00050   double get() override;
00051
00059   void set_limits(double lower, double upper) override;
00060
00064   bool is_on_target() override;
00065
00066   void reset();
00067
00068   PID pid;
00069
00070 private:
00071   FeedForward::ff_config_t &ff_cfg;
00072
00073   FeedForward ff;
00074
00075   double out;
00076   double lower_lim, upper_lim;
00077 };
```

6.27 take_back_half.h

```
00001 #pragma once
```

```
00002 #include "../core/include/utils/controls/feedback_base.h"
00003
00006 class TakeBackHalf : public Feedback {
00007
00008 public:
00009   TakeBackHalf(double TBH_gain, double first_cross_split, double on_target_threshold);
00018   void init(double start_pt, double set_pt);
00025   double update(double val) override;
00026
00030   double get() override;
00031
00038   void set_limits(double lower, double upper) override;
00039
00043   bool is_on_target() override;
00044
00045   double TBH_gain;
00046   double first_cross_split;
00047
00048 private:
00049   double on_target_threshhold;
00050
00051   double target = 0.0;
00052
00053   bool first_cross = true;
00054   double tbh = 0.0;
00055   double prev_error = 0.0;
00056
00057   double output = 0.0;
00058   double lower = 0.0, upper = 0.0;
00059 };
```

6.28 trapezoid_profile.h

```
00001 #pragma once
00002
00006 typedef struct {
00007   double pos;
00008   double vel;
00009   double accel;
00010
00011 } motion_t;
00012
00034 class TrapezoidProfile {
00035 public:
00042   TrapezoidProfile(double max_v, double accel);
00043
00050   motion_t calculate(double time_s);
00051
00057   void set_endpts(double start, double end);
00058
00063   void set_accel(double accel);
00064
00070   void set_max_v(double max_v);
00071
00077   double get_movement_time();
00078
00079 private:
00080   double start, end;
00081   double max_v;
00082   double accel;
00083   double time;
00084 };
```

6.29 generic_auto.h

```
00001 #pragma once
00002
00003 #include "vex.h"
00004 #include <functional>
00005 #include <map>
00006 #include <queue>
00007
00008 typedef std::function<bool(void)> state_ptr;
00009
00014 class GenericAuto {
00015 public:
00029   [[deprecated("Use CommandController instead.")]] bool run(bool blocking);
00030
00035   [[deprecated("Use CommandController instead.")]] void add(state_ptr new_state);
```

```
00036
00041   [[deprecated("Use CommandController instead.")]] void add_async(state_ptr async_state);
00042
00047   [[deprecated("Use CommandController instead.")]] void add_delay(int ms);
00048
00049 private:
00050   std::queue<state_ptr> state_list;
00051 };
```

6.30 geometry.h

```
00001 #pragma once
00002 #include <cmath>
00003
00007 struct point_t {
00008   double x;
00009   double y;
00010
00016   double dist(const point_t other) const {
00017     return std::sqrt(std::pow(this->x - other.x, 2) + pow(this->y - other.y, 2));
00018   }
00019
00025   point_t operator+(const point_t &other) const {
00026     point_t p{.x = this->x + other.x, .y = this->y + other.y};
00027     return p;
00028   }
00029
00035   point_t operator-(const point_t &other) const {
00036     point_t p{.x = this->x - other.x, .y = this->y - other.y};
00037     return p;
00038   }
00039
00040   point_t operator*(double s) const { return {x * s, y * s}; }
00041   point_t operator/(double s) const { return {x / s, y / s}; }
00042
00043   point_t operator-() const { return {-x, -y}; }
00044   point_t operator+() const { return {x, y}; }
00045
00046   bool operator==(const point_t &rhs) { return x == rhs.x && y == rhs.y; }
00047 };
00048
00052 struct pose_t {
00053   double x;
00054   double y;
00055   double rot;
00056
00057   point_t get_point() { return point_t{.x = x, .y = y}; }
00058 };
00059
00060 struct Rect {
00061   point_t min;
00062   point_t max;
00063   static Rect from_min_and_size(point_t min, point_t size) { return {min, min + size}; }
00064   point_t dimensions() const { return max - min; }
00065   point_t center() const { return (min + max) / 2; }
00066   double width() const { return max.x - min.x; }
00067   double height() const { return max.y - min.y; }
00068   bool contains(point_t p) const {
00069     bool xin = p.x > min.x && p.x < max.x;
00070     bool yin = p.y > min.y && p.y < max.y;
00071     return xin && yin;
00072   }
00073 };
00074
00075 struct Mat2 {
00076   double X11, X12;
00077   double X21, X22;
00078   point_t operator*(const point_t p) const {
00079     double outx = p.x * X11 + p.y * X12;
00080     double outy = p.x * X21 + p.y * X22;
00081     return {outx, outy};
00082   }
00083
00084   static Mat2 FromRotationDegrees(double degrees) {
00085     double rad = degrees * (M_PI / 180.0);
00086     double c = cos(rad);
00087     double s = sin(rad);
00088     return {c, -s, s, c};
00089   }
00090 };
```

6.31 graph_drawer.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/geometry.h"
00004 #include "../core/include/utils/vector2d.h"
00005 #include "vex.h"
00006 #include <cmath>
00007 #include <stdio.h>
00008 #include <string>
00009 #include <vector>
00010
00011 class GraphDrawer {
00012 public:
00019   GraphDrawer(int num_samples, double lower_bound, double upper_bound, std::vector<vex::color> colors,
00020               size_t num_series = 1);
00025   void add_samples(std::vector<point_t> sample);
00026
00032   void add_samples(std::vector<double> sample);
00033
00041   void draw(vex::brain::lcd &screen, int x, int y, int width, int height);
00042
00043 private:
00044   std::vector<std::vector<point_t» series;
00045   int sample_index = 0;
00046   std::vector<vex::color> cols;
00047   vex::color bgcol = vex::transparent;
00048   bool border;
00049   double upper;
00050   double lower;
00051   bool auto_fit = false;
00052 };
```

6.32 logger.h

```
00001 #pragma once
00002
00003 #include "vex.h"
00004 #include <cstdarg>
00005 #include <cstdio>
00006 #include <string>
00007
00009 enum LogLevel { DEBUG, NOTICE, WARNING, ERROR, CRITICAL, TIME };
00010
00012 class Logger {
00013 private:
00014   const std::string filename;
00015   vex::brain::sdcard sd;
00016   void write_level(LogLevel l);
00017
00018 public:
00020   static constexpr int MAX_FORMAT_LEN = 512;
00023   explicit Logger(const std::string &filename);
00024
00026   Logger(const Logger &l) = delete;
00028   Logger &operator=(const Logger &l) = delete;
00029
00032   void Log(const std::string &s);
00033
00037   void Log(LogLevel level, const std::string &s);
00038
00041   void Logln(const std::string &s);
00042
00046   void Logln(LogLevel level, const std::string &s);
00047
00051   void Logf(const char *fmt, ...);
00052
00057   void Logf(LogLevel level, const char *fmt, ...);
00058 };
```

6.33 math_util.h

```
00001 #pragma once
00002 #include "../core/include/utils/geometry.h"
00003 #include "math.h"
00004 #include "vex.h"
00005 #include <vector>
00006
00014 double clamp(double value, double low, double high);
```

```
00015
00022 double lerp(double a, double b, double t);
00029 double sign(double x);
00030
00031 double wrap_angle_deg(double input);
00032 double wrap_angle_rad(double input);
00033
00034 /*
00035 Calculates the variance of  a set of numbers (needed for linear regression)
00036 https://en.wikipedia.org/wiki/Variance
00037 @param values   the values for which the variance is taken
00038 @param mean     the average of values
00039 */
00040 double variance(std::vector<double> const &values, double mean);
00041
00042 /*
00043 Calculates the average of a vector of doubles
00044 @param values   the list of values for which the average is taken
00045 */
00046 double mean(std::vector<double> const &values);
00047
00048 /*
00049 Calculates the covariance of a set of points (needed for linear regression)
00050 https://en.wikipedia.org/wiki/Covariance
00051
00052 @param points   the points for which the covariance is taken
00053 @param meanx    the mean value of all x coordinates in points
00054 @param meany    the mean value of all y coordinates in points
00055 */
00056 double covariance(std::vector<std::pair<double, double» const &points, double meanx, double meany);
00057
00058 /*
00059 Calculates the slope and y intercept of the line of best fit for the data
00060 @param points the points for the data
00061 */
00062 std::pair<double, double> calculate_linear_regression(std::vector<std::pair<double, double» const
      &points);
00063
00064 double estimate_path_length(const std::vector<point_t> &points);
```

6.34 moving_average.h

```
00001 #pragma once
00002 #include <vector>
00003
00008 class Filter {
00009 public:
00010   virtual void add_entry(double n) = 0;
00011   virtual double get_value() const = 0;
00012 };
00013
00028 class MovingAverage : public Filter {
00029 public:
00030   /*
00031    * Create a moving average calculator with 0 as the default value
00032    *
00033    * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
      reading
00034    */
00035   MovingAverage(int buffer_size);
00036   /*
00037    * Create a moving average calculator with a specified default value
00038    * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
      reading
00039    * @param starting_value The value that the average will be before any data is added
00040    */
00041   MovingAverage(int buffer_size, double starting_value);
00042
00043   /*
00044    * Add a reading to the buffer
00045    * Before:
00046    * [ 1 1 2 2 3 3] => 2
00047    *   ^
00048    * After:
00049    * [ 2 1 2 2 3 3] => 2.16
00050    *     ^
00051    * @param n  the sample that will be added to the moving average.
00052    */
00053   void add_entry(double n) override;
00054
00059   double get_value() const override;
00060
00065   int get_size() const;
```

```
00066
00067 private:
00068   int buffer_index;           // index of the next value to be overridden
00069   std::vector<double> buffer; // all current data readings we've taken
00070   double current_avg;         // the current value of the data
00071 };
00072
00087 class ExponentialMovingAverage : public Filter {
00088 public:
00089   /*
00090    * Create a moving average calculator with 0 as the default value
00091    *
00092    * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
      reading
00093    */
00094   ExponentialMovingAverage(int buffer_size);
00095   /*
00096    * Create a moving average calculator with a specified default value
00097    * @param buffer_size    The size of the buffer. The number of samples that constitute a valid
      reading
00098    * @param starting_value The value that the average will be before any data is added
00099    */
00100   ExponentialMovingAverage(int buffer_size, double starting_value);
00101
00102   /*
00103    * Add a reading to the buffer
00104    * Before:
00105    * [ 1 1 2 2 3 3] => 2
00106    *   ^
00107    * After:
00108    * [ 2 1 2 2 3 3] => 2.16
00109    *     ^
00110    * @param n  the sample that will be added to the moving average.
00111    */
00112   void add_entry(double n) override;
00113
00118   double get_value() const override;
00119
00124   int get_size();
00125
00126 private:
00127   int buffer_index;           // index of the next value to be overridden
00128   std::vector<double> buffer; // all current data readings we've taken
00129   double current_avg;         // the current value of the data
00130 };
```

6.35 pure_pursuit.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/geometry.h"
00004 #include "../core/include/utils/vector2d.h"
00005 #include "vex.h"
00006 #include <vector>
00007
00008 using namespace vex;
00009
00010 namespace PurePursuit {
00014 class Path {
00015 public:
00021   Path(std::vector<point_t> points, double radius);
00022
00026   std::vector<point_t> get_points();
00027
00031   double get_radius();
00032
00036   bool is_valid();
00037
00038 private:
00039   std::vector<point_t> points;
00040   double radius;
00041   bool valid;
00042 };
00047 struct spline {
00048   double a, b, c, d, x_start, x_end;
00049
00050   double getY(double x) { return a * pow((x - x_start), 3) + b * pow((x - x_start), 2) + c * (x -
      x_start) + d; }
00051 };
00056 struct hermite_point {
00057   double x;
00058   double y;
00059   double dir;
```

```
00060   double mag;
00061
00062   point_t getPoint() const { return {x, y}; }
00063
00064   Vector2D getTangent() const { return Vector2D(dir, mag); }
00065 };
00066
00071 extern std::vector<point_t> line_circle_intersections(point_t center, double r, point_t point1,
      point_t point2);
00075 extern point_t get_lookahead(const std::vector<point_t> &path, pose_t robot_loc, double radius);
00076
00080 extern std::vector<point_t> inject_path(const std::vector<point_t> &path, double spacing);
00081
00093 extern std::vector<point_t> smooth_path(const std::vector<point_t> &path, double weight_data, double
      weight_smooth,
00094                                         double tolerance);
00095
00096 extern std::vector<point_t> smooth_path_cubic(const std::vector<point_t> &path, double res);
00097
00106 extern std::vector<point_t> smooth_path_hermite(const std::vector<hermite_point> &path, double step);
00107
00118 extern double estimate_remaining_dist(const std::vector<point_t> &path, pose_t robot_pose, double
      radius);
00119
00120 } // namespace PurePursuit
```

6.36 serializer.h

```
00001 #pragma once
00002 #include <algorithm>
00003 #include <map>
00004 #include <stdio.h>
00005 #include <string>
00006 #include <vector>
00007 #include <vex.h>
00008
00010 const char serialization_separator = '$';
00012 const std::size_t MAX_FILE_SIZE = 4096;
00013
00015 class Serializer {
00016 private:
00017   bool flush_always;
00018   std::string filename;
00019   std::map<std::string, int> ints;
00020   std::map<std::string, bool> bools;
00021   std::map<std::string, double> doubles;
00022   std::map<std::string, std::string> strings;
00023
00025   bool read_from_disk();
00026
00027 public:
00030   ~Serializer() {
00031     save_to_disk();
00032     printf("Saving %s\n", filename.c_str());
00033     fflush(stdout);
00034   }
00035
00040   explicit Serializer(const std::string &filename, bool flush_always = true)
00041       : flush_always(flush_always), filename(filename), ints({}), bools({}), doubles({}), strings({})
00042
00043   {
00044     read_from_disk();
00045   }
00046
00048   void save_to_disk() const;
00049
00051
00055   void set_int(const std::string &name, int i);
00056
00060   void set_bool(const std::string &name, bool b);
00061
00065   void set_double(const std::string &name, double d);
00066
00070   void set_string(const std::string &name, std::string str);
00071
00074
00079   int int_or(const std::string &name, int otherwise);
00080
00085   bool bool_or(const std::string &name, bool otherwise);
00086
00091   double double_or(const std::string &name, double otherwise);
00092
00097   std::string string_or(const std::string &name, std::string otherwise);
00098 };
```

6.37 state_machine.h

```
00001 #pragma once
00002 #include <string>
00003 #include <type_traits>
00004 #include <utility>
00005
00034 template <typename System, typename IDType, typename Message, int32_t delay_ms, bool do_log = false>
00035 class StateMachine {
00036   static_assert(std::is_enum<Message>::value, "Message should be an enum (it's easier that way)");
00037   static_assert(std::is_enum<IDType>::value, "IDType should be an enum (it's easier that way)");
00038
00039 public:
00046   class MaybeMessage {
00047   public:
00051     MaybeMessage() : exists(false) {}
00056     MaybeMessage(Message msg) : exists(true), thing(msg) {}
00061     bool has_message() { return exists; }
00067     Message message() { return thing; }
00068
00069   private:
00070     bool exists;
00071     Message thing;
00072   };
00078   struct State {
00079     // run once when we enter the state
00080     virtual void entry(System &) {}
00081     // run continously while in the state
00082     virtual MaybeMessage work(System &) { return {}; }
00083     // run once when we exit the state
00084     virtual void exit(System &) {}
00085     // respond to a message when one comes in
00086     virtual State *respond(System &s, Message m) = 0;
00087     // Identify
00088     virtual IDType id() const = 0;
00089
00090     // virtual destructor cuz c++
00091     virtual ~State() {}
00092   };
00093
00094   // Data that gets passed to the runner thread. Don't worry too much about
00095   // this
00096   using thread_data = std::pair<State *, StateMachine *>;
00097
00102   StateMachine(State *initial) : runner(thread_runner, new thread_data{initial, this}) {}
00103
00109   IDType current_state() const {
00110     mut.lock();
00111     auto t = cur_type;
00112     mut.unlock();
00113     return t;
00114   }
00120   void send_message(Message msg) {
00121     mut.lock();
00122     incoming_msg = msg;
00123     mut.unlock();
00124   }
00125
00126 private:
00127   vex::task runner;
00128   mutable vex::mutex mut;
00129   MaybeMessage incoming_msg;
00130   IDType cur_type;
00131
00138   static int thread_runner(void *vptr) {
00139     thread_data *ptr = static_cast<thread_data *>(vptr);
00140     State *cur_state = ptr->first;
00141
00142     StateMachine &sys = *ptr->second;
00143     System &derived = *static_cast<System *>(&sys);
00144
00145     cur_state->entry(derived);
00146
00147     sys.cur_type = cur_state->id();
00148
00149     auto respond_to_message = [&](Message msg) {
00150       if (do_log) {
00151         printf("responding to msg: %s\n", to_string(msg).c_str());
00152         fflush(stdout);
00153       }
00154
00155       State *next_state = cur_state->respond(derived, msg);
00156
00157       if (cur_state != next_state) {
00158         // switched states
00159         sys.mut.lock();
00160
```

```
00161         cur_state->exit(derived);
00162         next_state->entry(derived);
00163
00164         delete cur_state;
00165
00166         cur_state = next_state;
00167         sys.cur_type = cur_state->id();
00168
00169         sys.mut.unlock();
00170       }
00171     };
00172
00173     while (true) {
00174       if (do_log) {
00175         std::string str = to_string(cur_state->id());
00176         std::string str2 = to_string(sys.cur_type);
00177
00178         printf("state: %s %s\n", str.c_str(), str2.c_str());
00179       }
00180
00181       // Internal Message passed
00182       MaybeMessage internal_msg = cur_state->work(derived);
00183
00184       if (internal_msg.has_message()) {
00185         respond_to_message(internal_msg.message());
00186       }
00187
00188       // External Message passed
00189       sys.mut.lock();
00190       MaybeMessage incoming = sys.incoming_msg;
00191       sys.incoming_msg = {};
00192       sys.mut.unlock();
00193
00194       if (incoming.has_message()) {
00195         respond_to_message(incoming.message());
00196       }
00197
00198       vexDelay(delay_ms);
00199     }
00200     return 0;
00201   }
00202 };
```

6.38 vector2d.h

```
00001 #pragma once
00002
00003 #include "../core/include/utils/geometry.h"
00004 #include <cmath>
00005
00006 #ifndef PI
00007 #define PI 3.141592654
00008 #endif
00014 class Vector2D {
00015 public:
00022   Vector2D(double dir, double mag);
00023
00029   Vector2D(point_t p);
00030
00038   double get_dir() const;
00039
00043   double get_mag() const;
00044
00048   double get_x() const;
00049
00053   double get_y() const;
00054
00059   Vector2D normalize();
00060
00065   point_t point();
00066
00072   Vector2D operator*(const double &x);
00079   Vector2D operator+(const Vector2D &other);
00086   Vector2D operator-(const Vector2D &other);
00087
00088 private:
00089   double dir, mag;
00090 };
00091
00097 double deg2rad(double deg);
00098
00105 double rad2deg(double r);
```

Index

accel
  OdometryBase, 101
add
  CommandController, 31, 32
  GenericAuto, 66
add_async
  GenericAuto, 66
add_cancel_func
  CommandController, 32
add_delay
  CommandController, 33
  GenericAuto, 66
add_entry
  ExponentialMovingAverage, 46
  Filter, 53
  MovingAverage, 91
add_samples
  GraphDrawer, 68
AndCondition, 13
  test, 13
ang_accel_deg
  OdometryBase, 101
ang_speed_deg
  OdometryBase, 101
Async, 14
  run, 15
auto_drive
  MecanumDrive, 83
auto_turn
  MecanumDrive, 83
AutoChooser, 15
  AutoChooser, 16
  choice, 17
  draw, 16
  get_choice, 16
  list, 17
  update, 17
AutoChooser::entry_t, 44
  name, 44
AutoCommand, 18
  on_timeout, 19
  run, 19
  timeout_seconds, 19

background_task
  OdometryBase, 98
BangBang, 20
  get, 20
  init, 20
  is_on_target, 20
  set_limits, 21
  update, 21
BasicSolenoidSet, 22
  BasicSolenoidSet, 22
  run, 23
BasicSpinCommand, 23
  BasicSpinCommand, 24
  run, 25
BasicStopCommand, 25
  BasicStopCommand, 26
  run, 26
bool_or
  Serializer, 137
BrakeType
  TankDrive, 152
Branch, 27
  on_timeout, 28
  run, 28
ButtonWidget
  screen::ButtonWidget, 29

calculate
  FeedForward, 50
  TrapezoidProfile, 165
choice
  AutoChooser, 17
CommandController, 30
  add, 31, 32
  add_cancel_func, 32
  add_delay, 33
  CommandController, 31
  last_command_timed_out, 33
  run, 33
Condition, 33
config
  PID, 120
control_continuous
  Lift< T >, 73
control_manual
  Lift< T >, 73
control_setpoints
  Lift< T >, 74
Core, 1
current_pos
  OdometryBase, 101
current_state
  StateMachine< System, IDType, Message, delay_ms, do_log >, 147
CustomEncoder, 34
  CustomEncoder, 34

position, 35
rotation, 35
setPosition, 35
setRotation, 36
velocity, 36

DelayCommand, 36
DelayCommand, 37
run, 37
dist
point_t, 127
double_or
Serializer, 137
draw
AutoChooser, 16
FlywheelPage, 57
GraphDrawer, 68
screen::FunctionPage, 65
screen::OdometryPage, 103
screen::Page, 110
screen::PIDPage, 125
screen::StatsPage, 148
screen::WidgetPage, 178
VideoPlayer, 174
drive
MecanumDrive, 84
drive_arcade
TankDrive, 153
drive_correction_cutoff
robot_specs_t, 134
drive_forward
TankDrive, 154
drive_raw
MecanumDrive, 84
drive_tank
TankDrive, 155
drive_tank_raw
TankDrive, 155
drive_to_point
TankDrive, 156
DriveForwardCommand, 38
DriveForwardCommand, 39
on_timeout, 39
run, 40
DriveStopCommand, 40
DriveStopCommand, 41
on_timeout, 41
run, 41
DriveToPointCommand, 42
DriveToPointCommand, 43
run, 44

end_async
OdometryBase, 98
error_method
PID::pid_config_t, 120
ERROR_TYPE
PID, 115
ExponentialMovingAverage, 45
add_entry, 46
ExponentialMovingAverage, 45
get_size, 46
get_value, 46

Feedback, 47
get, 48
init, 48
is_on_target, 48
set_limits, 48
update, 49
FeedForward, 49
calculate, 50
FeedForward, 50
FeedForward::ff_config_t, 51
kA, 51
kG, 51
kS, 52
kV, 52
Filter, 52
add_entry, 53
get_value, 53
Flywheel, 53
Flywheel, 54
get_motors, 54
get_target, 54
getRPM, 54
is_on_target, 55
Page, 55
spin_manual, 55
spin_rpm, 56
SpinRpmCmd, 56
spinRPMTask, 57
stop, 56
WaitUntilUpToSpeedCmd, 56
FlywheelPage, 57
draw, 57
update, 57
FlywheelStopCommand, 58
FlywheelStopCommand, 59
run, 59
FlywheelStopMotorsCommand, 59
FlywheelStopMotorsCommand, 60
run, 60
FlywheelStopNonTasksCommand, 61
FunctionCommand, 62
run, 63
FunctionCondition, 63
test, 64
FunctionPage
screen::FunctionPage, 64

GenericAuto, 65
add, 66
add_async, 66
add_delay, 66
run, 67
get
BangBang, 20

Feedback, 48
MotionController, 88
PID, 116
PIDFF, 121
TakeBackHalf, 150
get_accel
OdometryBase, 98
get_angular_accel_deg
OdometryBase, 98
get_angular_speed_deg
OdometryBase, 98
get_async
Lift< T >, 74
get_choice
AutoChooser, 16
get_dir
Vector2D, 171
get_error
PID, 116
get_mag
Vector2D, 171
get_motion
MotionController, 88
get_motors
Flywheel, 54
get_movement_time
TrapezoidProfile, 165
get_points
PurePursuit::Path, 114
get_position
OdometryBase, 99
get_radius
PurePursuit::Path, 114
get_sensor_val
PID, 116
get_setpoint
Lift< T >, 74
get_size
ExponentialMovingAverage, 46
MovingAverage, 91
get_speed
OdometryBase, 99
get_target
Flywheel, 54
PID, 116
get_value
ExponentialMovingAverage, 46
Filter, 53
MovingAverage, 92
get_x
Vector2D, 171
get_y
Vector2D, 171
getRPM
Flywheel, 54
GraphDrawer, 67
add_samples, 68
draw, 68
GraphDrawer, 67

handle
OdometryBase, 102
has_message
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 81
hold
Lift< T >, 74
home
Lift< T >, 74

IfTimePassed, 69
test, 70
include/robot_specs.h, 181
include/subsystems/custom_encoder.h, 181
include/subsystems/flywheel.h, 181
include/subsystems/fun/pl_mpeg.h, 182
include/subsystems/fun/video.h, 230
include/subsystems/layout.h, 230
include/subsystems/lift.h, 230
include/subsystems/mecanum_drive.h, 233
include/subsystems/odometry/odometry_3wheel.h, 233
include/subsystems/odometry/odometry_base.h, 234
include/subsystems/odometry/odometry_tank.h, 235
include/subsystems/screen.h, 235
include/subsystems/tank_drive.h, 238
include/utils/auto_chooser.h, 239
include/utils/command_structure/auto_command.h, 239
include/utils/command_structure/basic_command.h, 241
include/utils/command_structure/command_controller.h, 242
include/utils/command_structure/delay_command.h, 242
include/utils/command_structure/drive_commands.h, 243
include/utils/command_structure/flywheel_commands.h, 245
include/utils/controls/bang_bang.h, 245
include/utils/controls/feedback_base.h, 246
include/utils/controls/feedforward.h, 246
include/utils/controls/motion_controller.h, 246
include/utils/controls/pid.h, 247
include/utils/controls/pidff.h, 248
include/utils/controls/take_back_half.h, 248
include/utils/controls/trapezoid_profile.h, 249
include/utils/generic_auto.h, 249
include/utils/geometry.h, 250
include/utils/graph_drawer.h, 251
include/utils/logger.h, 251
include/utils/math_util.h, 251
include/utils/moving_average.h, 252
include/utils/pure_pursuit.h, 253
include/utils/serializer.h, 254
include/utils/state_machine.h, 255
include/utils/vector2d.h, 256
init
BangBang, 20

Feedback, 48
MotionController, 88
PID, 117
PIDFF, 121
TakeBackHalf, 150
InOrder, 70
on_timeout, 71
run, 71
int_or
Serializer, 137
is_on_target
BangBang, 20
Feedback, 48
Flywheel, 55
MotionController, 88
PID, 118
PIDFF, 122
TakeBackHalf, 150
is_valid
PurePursuit::Path, 114

kA
FeedForward::ff_config_t, 51
kG
FeedForward::ff_config_t, 51
kS
FeedForward::ff_config_t, 52
kV
FeedForward::ff_config_t, 52

last_command_timed_out
CommandController, 33
Lift
Lift< T >, 73
Lift< T >, 72
control_continuous, 73
control_manual, 73
control_setpoints, 74
get_async, 74
get_setpoint, 74
hold, 74
home, 74
Lift, 73
set_async, 75
set_position, 75
set_sensor_function, 75
set_sensor_reset, 76
set_setpoint, 76
Lift< T >::lift_cfg_t, 76
list
AutoChooser, 17
Log
Logger, 78
Logf
Logger, 78
Logger, 77
Log, 78
Logf, 78
Logger, 77
Logln, 79
Logln
Logger, 79

Mat2, 80
MaybeMessage
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 81
MecanumDrive, 82
auto_drive, 83
auto_turn, 83
drive, 84
drive_raw, 84
MecanumDrive, 82
MecanumDrive::mecanumdrive_config_t, 85
message
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 81
modify_inputs
TankDrive, 157
motion_t, 85
MotionController, 86
get, 88
get_motion, 88
init, 88
is_on_target, 88
MotionController, 87
set_limits, 88
tune_feedforward, 89
update, 89
MotionController::m_profile_cfg_t, 79
MovingAverage, 90
add_entry, 91
get_size, 91
get_value, 92
MovingAverage, 91
mut
OdometryBase, 102

name
AutoChooser::entry_t, 44
None
TankDrive, 153
normalize
Vector2D, 172

Odometry3Wheel, 92
Odometry3Wheel, 94
tune, 94
update, 95
Odometry3Wheel::odometry3wheel_cfg_t, 95
off_axis_center_dist, 96
wheel_diam, 96
wheelbase_dist, 96
OdometryBase, 96
accel, 101
ang_accel_deg, 101
ang_speed_deg, 101
background_task, 98

current_pos, 101
end_async, 98
get_accel, 98
get_angular_accel_deg, 98
get_angular_speed_deg, 98
get_position, 99
get_speed, 99
handle, 102
mut, 102
OdometryBase, 97
pos_diff, 99
rot_diff, 100
set_position, 100
smallest_angle, 100
speed, 102
update, 101
zero_pos, 102
OdometryPage
screen::OdometryPage, 103
OdometryTank, 104
OdometryTank, 105, 106
set_position, 107
update, 107
OdomSetPosition, 107
OdomSetPosition, 108
run, 109
off_axis_center_dist
Odometry3Wheel::odometry3wheel_cfg_t, 96
on_target_time
PID::pid_config_t, 120
on_timeout
AutoCommand, 19
Branch, 28
DriveForwardCommand, 39
DriveStopCommand, 41
InOrder, 71
Parallel, 112
PurePursuitCommand, 130
RepeatUntil, 133
TurnDegreesCommand, 167
TurnToHeadingCommand, 169
operator+
point_t, 127
Vector2D, 172
operator-
point_t, 128
Vector2D, 173
operator∗
Vector2D, 172
OrCondition, 109
test, 109

Page
Flywheel, 55
Parallel, 111
on_timeout, 112
run, 112
parallel_runner_info, 113
Path
PurePursuit::Path, 113
PID, 114
config, 120
ERROR_TYPE, 115
get, 116
get_error, 116
get_sensor_val, 116
get_target, 116
init, 117
is_on_target, 118
PID, 115
reset, 118
set_limits, 118
set_target, 118
update, 119
PID::pid_config_t, 120
error_method, 120
on_target_time, 120
PIDFF, 121
get, 121
init, 121
is_on_target, 122
set_limits, 122
set_target, 122
update, 123
PIDPage
screen::PIDPage, 124
plm_frame_t, 125
plm_packet_t, 126
plm_plane_t, 126
plm_samples_t, 126
point
Vector2D, 173
point_t, 126
dist, 127
operator+, 127
operator-, 128
pos_diff
OdometryBase, 99
pose_t, 128
position
CustomEncoder, 35
pure_pursuit
TankDrive, 158, 159
PurePursuit::hermite_point, 69
PurePursuit::Path, 113
get_points, 114
get_radius, 114
is_valid, 114
Path, 113
PurePursuit::spline, 144
PurePursuitCommand, 129
on_timeout, 130
PurePursuitCommand, 130
run, 130

Rect, 130
RepeatUntil, 131
on_timeout, 133

RepeatUntil, 132, 133
run, 133
reset
PID, 118
reset_auto
TankDrive, 159
robot_specs_t, 134
drive_correction_cutoff, 134
rot_diff
OdometryBase, 100
rotation
CustomEncoder, 35
run
Async, 15
AutoCommand, 19
BasicSolenoidSet, 23
BasicSpinCommand, 25
BasicStopCommand, 26
Branch, 28
CommandController, 33
DelayCommand, 37
DriveForwardCommand, 40
DriveStopCommand, 41
DriveToPointCommand, 44
FlywheelStopCommand, 59
FlywheelStopMotorsCommand, 60
FunctionCommand, 63
GenericAuto, 67
InOrder, 71
OdomSetPosition, 109
Parallel, 112
PurePursuitCommand, 130
RepeatUntil, 133
SpinRPMCommand, 144
TurnDegreesCommand, 167
TurnToHeadingCommand, 169
WaitUntilCondition, 176
WaitUntilUpToSpeedCommand, 177

save_to_disk
Serializer, 138
screen::ButtonConfig, 28
screen::ButtonWidget, 28
ButtonWidget, 29
update, 30
screen::CheckboxConfig, 30
screen::FunctionPage, 64
draw, 65
FunctionPage, 64
update, 65
screen::LabelConfig, 72
screen::OdometryPage, 102
draw, 103
OdometryPage, 103
update, 103
screen::Page, 110
draw, 110
update, 111
screen::PIDPage, 124
draw, 125
PIDPage, 124
update, 125
screen::ScreenData, 134
screen::ScreenRect, 135
screen::SizedWidget, 139
screen::SliderConfig, 140
screen::SliderWidget, 140
SliderWidget, 141
update, 142
screen::StatsPage, 147
draw, 148
StatsPage, 148
update, 148
screen::TextConfig, 163
screen::WidgetConfig, 177
screen::WidgetPage, 178
draw, 178
update, 179
send_message
StateMachine< System, IDType, Message, delay_ms, do_log >, 147
Serializer, 135
bool_or, 137
double_or, 137
int_or, 137
save_to_disk, 138
Serializer, 136
set_bool, 138
set_double, 138
set_int, 138
set_string, 139
string_or, 139
set_accel
TrapezoidProfile, 165
set_async
Lift< T >, 75
set_bool
Serializer, 138
set_double
Serializer, 138
set_endpts
TrapezoidProfile, 165
set_int
Serializer, 138
set_limits
BangBang, 21
Feedback, 48
MotionController, 88
PID, 118
PIDFF, 122
TakeBackHalf, 150
set_max_v
TrapezoidProfile, 166
set_position
Lift< T >, 75
OdometryBase, 100
OdometryTank, 107

set_sensor_function
Lift< T >, 75
set_sensor_reset
Lift< T >, 76
set_setpoint
Lift< T >, 76
set_string
Serializer, 139
set_target
PID, 118
PIDFF, 122
setPosition
CustomEncoder, 35
setRotation
CustomEncoder, 36
SliderCfg, 140
SliderWidget
screen::SliderWidget, 141
smallest_angle
OdometryBase, 100
Smart
TankDrive, 153
speed
OdometryBase, 102
spin_manual
Flywheel, 55
spin_rpm
Flywheel, 56
SpinRpmCmd
Flywheel, 56
SpinRPMCommand, 142
run, 144
SpinRPMCommand, 143
spinRPMTask
Flywheel, 57
StateMachine
StateMachine< System, IDType, Message, delay_ms, do_log >, 146
StateMachine< System, IDType, Message, delay_ms, do_log >, 145
current_state, 147
send_message, 147
StateMachine, 146
StateMachine< System, IDType, Message, delay_ms, do_log >::MaybeMessage, 80
has_message, 81
MaybeMessage, 81
message, 81
StateMachine< System, IDType, Message, delay_ms, do_log >::State, 145
StatsPage
screen::StatsPage, 148
stop
Flywheel, 56
TankDrive, 160
string_or
Serializer, 139

TakeBackHalf, 149
get, 150
init, 150
is_on_target, 150
set_limits, 150
update, 151
TankDrive, 151
BrakeType, 152
drive_arcade, 153
drive_forward, 154
drive_tank, 155
drive_tank_raw, 155
drive_to_point, 156
modify_inputs, 157
None, 153
pure_pursuit, 158, 159
reset_auto, 159
Smart, 153
stop, 160
TankDrive, 153
turn_degrees, 160
turn_to_heading, 161, 162
ZeroVelocity, 153
test
AndCondition, 13
FunctionCondition, 64
IfTimePassed, 70
OrCondition, 109
TimesTestedCondition, 163
timeout_seconds
AutoCommand, 19
TimesTestedCondition, 163
test, 163
TrapezoidProfile, 163
calculate, 165
get_movement_time, 165
set_accel, 165
set_endpts, 165
set_max_v, 166
TrapezoidProfile, 164
tune
Odometry3Wheel, 94
tune_feedforward
MotionController, 89
turn_degrees
TankDrive, 160
turn_to_heading
TankDrive, 161, 162
TurnDegreesCommand, 166
on_timeout, 167
run, 167
TurnDegreesCommand, 167
TurnToHeadingCommand, 168
on_timeout, 169
run, 169
TurnToHeadingCommand, 169

update
AutoChooser, 17
BangBang, 21

Feedback, 49
FlywheelPage, 57
MotionController, 89
Odometry3Wheel, 95
OdometryBase, 101
OdometryTank, 107
PID, 119
PIDFF, 123
screen::ButtonWidget, 30
screen::FunctionPage, 65
screen::OdometryPage, 103
screen::Page, 111
screen::PIDPage, 125
screen::SliderWidget, 142
screen::StatsPage, 148
screen::WidgetPage, 179
TakeBackHalf, 151
VideoPlayer, 174

Vector2D, 170
get_dir, 171
get_mag, 171
get_x, 171
get_y, 171
normalize, 172
operator+, 172
operator-, 173
operator∗, 172
point, 173
Vector2D, 170, 171
velocity
CustomEncoder, 36
VideoPlayer, 173
draw, 174
update, 174

WaitUntilCondition, 175
run, 176
WaitUntilUpToSpeedCmd
Flywheel, 56
WaitUntilUpToSpeedCommand, 176
run, 177
WaitUntilUpToSpeedCommand, 177
wheel_diam
Odometry3Wheel::odometry3wheel_cfg_t, 96
wheelbase_dist
Odometry3Wheel::odometry3wheel_cfg_t, 96

zero_pos
OdometryBase, 102
ZeroVelocity
TankDrive, 153