Vex Team A

1.0.2

Generated by Doxygen 1.8.13

Contents

1 InTheZoneA **1**

2 Namespace Index **3**

2.1 Namespace List 3

3 Data Structure Index **5**

3.1 Data Structures 5

4 File Index **7**

4.1 File List 7

5 Namespace Documentation **9**

5.1 testMath Namespace Reference 9

5.1.1 Function Documentation 9

5.1.1.1 test() 9

6 Data Structure Documentation **11**

6.1 _matrix Struct Reference 11

6.1.1 Detailed Description 11

6.1.2 Field Documentation 11

6.1.2.1 data 11

6.1.2.2 height 12

6.1.2.3 width 12

6.2 accelerometer_odometry Struct Reference 12

6.2.1 Detailed Description 12

6.2.2 Field Documentation 12

6.2.2.1 x 12
6.2.2.2 y 13
6.3 cord Struct Reference 13
6.3.1 Detailed Description 13
6.3.2 Field Documentation 13
6.3.2.1 x 13
6.3.2.2 y 14
6.4 encoder_odemtry Struct Reference 14
6.4.1 Detailed Description 14
6.4.2 Field Documentation 14
6.4.2.1 theta 14
6.4.2.2 x 14
6.4.2.3 y 15
6.5 lcd_buttons Struct Reference 15
6.5.1 Detailed Description 15
6.5.2 Field Documentation 15
6.5.2.1 left 15
6.5.2.2 middle 16
6.5.2.3 right 16
6.6 location Struct Reference 16
6.6.1 Detailed Description 16
6.6.2 Field Documentation 16
6.6.2.1 theta 17
6.6.2.2 x 17
6.6.2.3 y 17
6.7 menu_t Struct Reference 17
6.7.1 Detailed Description 18
6.7.2 Field Documentation 18
6.7.2.1 current 18
6.7.2.2 length 19

6.7.2.3 max 19
6.7.2.4 max_f 19
6.7.2.5 min 20
6.7.2.6 min_f 20
6.7.2.7 options 20
6.7.2.8 prompt 21
6.7.2.9 step 21
6.7.2.10 step_f 21
6.7.2.11 type 22
6.8 polar_cord Struct Reference 22
6.8.1 Detailed Description 22
6.8.2 Field Documentation 23
6.8.2.1 angle 23
6.8.2.2 magnitue 23

7 File Documentation 25

7.1 include/auto.h File Reference 25
7.1.1 Detailed Description 26
7.1.2 Macro Definition Documentation 26
7.1.2.1 FRONT_LEFT_IME 26
7.1.2.2 GOAL_HEIGHT 26
7.1.2.3 MID_LEFT_DRIVE 26
7.1.2.4 MID_RIGHT_DRIVE 27
7.1.2.5 STOP_ONE 27
7.2 auto.h 27
7.3 include/battery.h File Reference 27
7.3.1 Detailed Description 28
7.3.2 Macro Definition Documentation 28
7.3.2.1 MIN_BACKUP_VOLTAGE 28
7.3.2.2 MIN_MAIN_VOLTAGE 29
7.3.3 Function Documentation 29

7.3.3.1 backup_battery_voltage() 29

7.3.3.2 battery_level_acceptable() 29

7.3.3.3 main_battery_voltage() 30

7.4 battery.h 30

7.5 include/claw.h File Reference 30

7.5.1 Detailed Description 31

7.5.2 Macro Definition Documentation 32

7.5.2.1 CLAW_CLOSE 32

7.5.2.2 CLAW_CLOSE_VAL 32

7.5.2.3 CLAW_OPEN 32

7.5.2.4 CLAW_OPEN_VAL 33

7.5.2.5 MAX_CLAW_SPEED 33

7.5.2.6 MIN_CLAW_SPEED 33

7.5.3 Enumeration Type Documentation 33

7.5.3.1 claw_state 33

7.5.4 Function Documentation 34

7.5.4.1 close_claw() 34

7.5.4.2 getClawTicks() 34

7.5.4.3 open_claw() 35

7.5.4.4 set_claw_motor() 35

7.5.4.5 update_claw() 36

7.6 claw.h 36

7.7 include/controller.h File Reference 37

7.7.1 Detailed Description 38

7.7.2 Macro Definition Documentation 38

7.7.2.1 LEFT_BUMPERS 38

7.7.2.2 LEFT_BUTTONS 38

7.7.2.3 LEFT_JOY_X 38

7.7.2.4 LEFT_JOY_Y 39

7.7.2.5 MASTER 39

7.7.2.6 PARTNER 39
7.7.2.7 RIGHT_BUMPERS 40
7.7.2.8 RIGHT_BUTTONS 40
7.7.2.9 RIGHT_JOY_X 40
7.7.2.10 RIGHT_JOY_Y 40
7.7.3 Enumeration Type Documentation 41
7.7.3.1 joystick 41
7.7.4 Function Documentation 41
7.7.4.1 get_joystick_cord() 41
7.8 controller.h 42
7.9 include/drive.h File Reference 42
7.9.1 Detailed Description 43
7.9.2 Macro Definition Documentation 44
7.9.2.1 THRESHOLD 44
7.9.3 Typedef Documentation 44
7.9.3.1 side_t 44
7.9.4 Enumeration Type Documentation 44
7.9.4.1 side 44
7.9.5 Function Documentation 45
7.9.5.1 set_side_speed() 45
7.9.5.2 setThresh() 46
7.9.5.3 update_drive_motors() 46
7.10 drive.h 47
7.11 include/encoders.h File Reference 47
7.11.1 Detailed Description 48
7.11.2 Macro Definition Documentation 48
7.11.2.1 IME_NUMBER 48
7.11.3 Function Documentation 48
7.11.3.1 get_encoder_ticks() 49
7.11.3.2 get_encoder_velocity() 49

7.11.3.3 init_encoders() 50
7.12 encoders.h 50
7.13 include/gyro.h File Reference 50
7.13.1 Macro Definition Documentation 51
7.13.1.1 GYRO_MULTIPLIER 51
7.13.1.2 GYRO_PORT 51
7.13.2 Function Documentation 51
7.13.2.1 get_main_gyro_angluar_velocity() 51
7.13.2.2 init_main_gyro() 52
7.14 gyro.h 52
7.15 include/lcd.h File Reference 52
7.15.1 Detailed Description 53
7.15.2 Macro Definition Documentation 53
7.15.2.1 BOTTOM_ROW 54
7.15.2.2 TOP_ROW 54
7.15.3 Enumeration Type Documentation 54
7.15.3.1 button_state 54
7.15.4 Function Documentation 55
7.15.4.1 init_main_lcd() 55
7.15.4.2 lcd_clear() 56
7.15.4.3 lcd_get_pressed_buttons() 56
7.15.4.4 lcd_print() 57
7.15.4.5 lcd_printf() 57
7.15.4.6 lcd_set_backlight() 58
7.15.4.7 promt_confirmation() 58
7.16 lcd.h 59
7.17 include/lifter.h File Reference 60
7.17.1 Detailed Description 61
7.17.2 Macro Definition Documentation 61
7.17.2.1 HEIGHT 61

7.17.2.2 INIT_ROTATION 62
7.17.2.3 LIFTER_D 62
7.17.2.4 LIFTER_DOWN 62
7.17.2.5 LIFTER_DOWN_PARTNER 62
7.17.2.6 LIFTER_DRIVER_LOAD 62
7.17.2.7 LIFTER_I 63
7.17.2.8 LIFTER_P 63
7.17.2.9 LIFTER_UP 63
7.17.2.10 LIFTER_UP_PARTNER 63
7.17.2.11 MAIN_LIFTER_MIN_HEIGHT 63
7.17.2.12 MAIN_LIFTER_POT 64
7.17.2.13 SECONDARY_LIFTER_MAX_HEIGHT 64
7.17.2.14 SECONDARY_LIFTER_MIN_HEIGHT 64
7.17.2.15 SECONDARY_LIFTER_POT_PORT 64
7.17.2.16 THRESHOLD 64
7.17.3 Function Documentation 64
7.17.3.1 getLifterHeight() 65
7.17.3.2 getLifterTicks() 65
7.17.3.3 lifterPotentiometerToDegree() 65
7.17.3.4 lower_main_lifter() 66
7.17.3.5 lower_secondary_lifter() 67
7.17.3.6 raise_main_lifter() 67
7.17.3.7 raise_secondary_lifter() 68
7.17.3.8 set_lifter_pos() 68
7.17.3.9 set_main_lifter_motors() 69
7.17.3.10 set_secondary_lifter_motors() 69
7.17.3.11 update_lifter() 70
7.18 lifter.h 71
7.19 include/localization.h File Reference 71
7.19.1 Detailed Description 72

7.19.2 Macro Definition Documentation 72
7.19.2.1 LOCALIZATION_UPDATE_FREQUENCY 73
7.19.3 Function Documentation 73
7.19.3.1 get_position() 73
7.19.3.2 init_localization() 73
7.19.3.3 update_position() 74
7.20 localization.h 75
7.21 include/log.h File Reference 75
7.21.1 Detailed Description 76
7.21.2 Macro Definition Documentation 76
7.21.2.1 DEBUG 76
7.21.2.2 ERROR 77
7.21.2.3 INFO 77
7.21.2.4 NONE 77
7.21.2.5 WARNING 78
7.21.3 Function Documentation 78
7.21.3.1 debug() 78
7.21.3.2 error() 79
7.21.3.3 info() 79
7.21.3.4 init_error() 80
7.21.3.5 warning() 81
7.22 log.h 81
7.23 include/main.h File Reference 82
7.23.1 Detailed Description 82
7.23.2 Function Documentation 83
7.23.2.1 autonomous() 83
7.23.2.2 initialize() 84
7.23.2.3 initializeIO() 84
7.23.2.4 operatorControl() 85
7.24 main.h 86

7.25 include/matrix.h File Reference 86
7.25.1 Detailed Description 87
7.25.2 Typedef Documentation 87
7.25.2.1 matrix 88
7.25.3 Function Documentation 88
7.25.3.1 assert() 88
7.25.3.2 copyMatrix() 88
7.25.3.3 covarianceMatrix() 89
7.25.3.4 dotDiagonalMatrix() 90
7.25.3.5 dotProductMatrix() 91
7.25.3.6 freeMatrix() 92
7.25.3.7 identityMatrix() 92
7.25.3.8 makeMatrix() 93
7.25.3.9 meanMatrix() 94
7.25.3.10 multiplyMatrix() 94
7.25.3.11 printMatrix() 95
7.25.3.12 rowSwap() 96
7.25.3.13 scaleMatrix() 96
7.25.3.14 traceMatrix() 97
7.25.3.15 transposeMatrix() 97
7.26 matrix.h 98
7.27 include/menu.h File Reference 99
7.27.1 Detailed Description 100
7.27.2 Typedef Documentation 100
7.27.2.1 menu_t 100
7.27.3 Enumeration Type Documentation 100
7.27.3.1 menu_type 101
7.27.4 Function Documentation 101
7.27.4.1 calculate_current_display() 101
7.27.4.2 create_menu() 102

7.27.4.3 denint_menu() 102
7.27.4.4 display_menu() 103
7.27.4.5 init_menu_float() 104
7.27.4.6 init_menu_int() 105
7.27.4.7 init_menu_var() 106
7.28 menu.h 107
7.29 include/mobile_goal_intake.h File Reference 107
7.29.1 Function Documentation 108
7.29.1.1 update_intake() 108
7.30 mobile_goal_intake.h 108
7.31 include/motor_ports.h File Reference 109
7.31.1 Detailed Description 109
7.31.2 Macro Definition Documentation 109
7.31.2.1 _MOTOR_PORTS_H_ 110
7.31.2.2 CLAW_MOTOR 110
7.31.2.3 INTAKE_MOTOR 110
7.31.2.4 MAX_SPEED 110
7.31.2.5 MIN_SPEED 110
7.31.2.6 MOTOR_BACK_LEFT 111
7.31.2.7 MOTOR_BACK_RIGHT 111
7.31.2.8 MOTOR_FRONT_LEFT 111
7.31.2.9 MOTOR_FRONT_RIGHT 112
7.31.2.10 MOTOR_LIFT 112
7.31.2.11 MOTOR_MIDDLE_LEFT 112
7.31.2.12 MOTOR_MIDDLE_RIGHT 113
7.31.2.13 MOTOR_SECONDARY_LIFTER 113
7.32 motor_ports.h 113
7.33 include/partner.h File Reference 114
7.33.1 Enumeration Type Documentation 114
7.33.1.1 CONTROLL_MODE 114

7.33.2 Function Documentation 114
7.33.2.1 get_mode() 115
7.33.2.2 update_control() 115
7.34 partner.h 115
7.35 include/potentiometer.h File Reference 116
7.35.1 Macro Definition Documentation 116
7.35.1.1 DEG_MAX 116
7.35.1.2 TICK_MAX 116
7.36 potentiometer.h 116
7.37 include/sensor_ports.h File Reference 116
7.37.1 Macro Definition Documentation 117
7.37.1.1 CLAW_POT 117
7.37.1.2 IME_FRONT_RIGHT 117
7.37.1.3 LIFTER 117
7.38 sensor_ports.h 118
7.39 include/slew.h File Reference 118
7.39.1 Detailed Description 119
7.39.2 Macro Definition Documentation 119
7.39.2.1 MOTOR_PORTS 119
7.39.2.2 RAMP_PROPORTION 119
7.39.2.3 UPDATE_PERIOD_MS 120
7.39.3 Function Documentation 120
7.39.3.1 deinitslew() 120
7.39.3.2 init_slew() 121
7.39.3.3 set_motor_immediate() 121
7.39.3.4 set_motor_slew() 122
7.39.3.5 updateMotors() 123
7.40 slew.h 123
7.41 include/toggle.h File Reference 124
7.41.1 Enumeration Type Documentation 124

7.41.1.1 button_t 124
7.41.2 Function Documentation 125
7.41.2.1 buttonGetState() 126
7.41.2.2 buttonInit() 127
7.41.2.3 buttonIsNewPress() 127
7.42 toggle.h 128
7.43 include/vlib.h File Reference 129
7.43.1 Detailed Description 129
7.43.2 Function Documentation 130
7.43.2.1 calloc_real() 130
7.43.2.2 ftoaa() 130
7.43.2.3 itoaa() 131
7.43.2.4 reverse() 132
7.44 vlib.h 132
7.45 include/vmath.h File Reference 133
7.45.1 Detailed Description 133
7.45.2 Macro Definition Documentation 134
7.45.2.1 M_PI 134
7.45.3 Function Documentation 134
7.45.3.1 cartesian_cord_to_polar() 134
7.45.3.2 cartesian_to_polar() 135
7.45.3.3 max() 136
7.45.3.4 min() 136
7.45.3.5 sind() 137
7.46 vmath.h 137
7.47 README.md File Reference 138
7.48 README.md 138
7.49 src/auto.c File Reference 138
7.49.1 Detailed Description 138
7.49.2 Function Documentation 138

7.49.2.1 autonomous() . . . 139
7.50 auto.c . . . 140
7.51 src/battery.c File Reference . . . 140
7.51.1 Function Documentation . . . 141
7.51.1.1 backup_battery_voltage() . . . 141
7.51.1.2 battery_level_acceptable() . . . 141
7.51.1.3 main_battery_voltage() . . . 142
7.52 battery.c . . . 142
7.53 src/claw.c File Reference . . . 142
7.53.1 Function Documentation . . . 143
7.53.1.1 close_claw() . . . 143
7.53.1.2 open_claw() . . . 143
7.53.1.3 set_claw_motor() . . . 144
7.53.1.4 update_claw() . . . 144
7.53.2 Variable Documentation . . . 145
7.53.2.1 state . . . 145
7.54 claw.c . . . 145
7.55 src/controller.c File Reference . . . 145
7.55.1 Function Documentation . . . 146
7.55.1.1 get_joystick_cord() . . . 146
7.56 controller.c . . . 146
7.57 src/drive.c File Reference . . . 147
7.57.1 Function Documentation . . . 147
7.57.1.1 getThresh() . . . 147
7.57.1.2 joystickExp() . . . 148
7.57.1.3 set_side_speed() . . . 148
7.57.1.4 setThresh() . . . 149
7.57.1.5 update_drive_motors() . . . 150
7.57.2 Variable Documentation . . . 150
7.57.2.1 thresh . . . 150

7.58 drive.c 151
7.59 src/encoders.c File Reference 152
7.59.1 Function Documentation 152
7.59.1.1 get_encoder_ticks() 152
7.59.1.2 get_encoder_velocity() 153
7.59.1.3 init_encoders() 153
7.60 encoders.c 154
7.61 src/gyro.c File Reference 154
7.61.1 Function Documentation 154
7.61.1.1 get_main_gyro_angluar_velocity() 154
7.61.1.2 init_main_gyro() 155
7.61.2 Variable Documentation 155
7.61.2.1 main_gyro 155
7.62 gyro.c 155
7.63 src/init.c File Reference 155
7.63.1 Detailed Description 156
7.63.2 Function Documentation 156
7.63.2.1 initialize() 156
7.63.2.2 initializeIO() 157
7.64 init.c 157
7.65 src/lcd.c File Reference 157
7.65.1 Function Documentation 158
7.65.1.1 init_main_lcd() 158
7.65.1.2 lcd_assert() 159
7.65.1.3 lcd_clear() 160
7.65.1.4 lcd_get_pressed_buttons() 160
7.65.1.5 lcd_print() 161
7.65.1.6 lcd_printf() 161
7.65.1.7 lcd_set_backlight() 162
7.65.1.8 promt_confirmation() 162

7.65.2 Variable Documentation 163
7.65.2.1 lcd_port 163
7.66 lcd.c 164
7.67 src/lifter.c File Reference 164
7.67.1 Function Documentation 165
7.67.1.1 getLifterHeight() 165
7.67.1.2 getLifterTicks() 166
7.67.1.3 lifterPotentiometerToDegree() 166
7.67.1.4 lower_main_lifter() 167
7.67.1.5 lower_secondary_lifter() 167
7.67.1.6 raise_main_lifter() 168
7.67.1.7 raise_secondary_lifter() 168
7.67.1.8 set_lifter_pos() 168
7.67.1.9 set_main_lifter_motors() 169
7.67.1.10 set_secondary_lifter_motors() 170
7.67.1.11 update_lifter() 170
7.68 lifter.c 171
7.69 src/localization.c File Reference 172
7.69.1 Function Documentation 173
7.69.1.1 calculate_accelerometer_odemetry() 173
7.69.1.2 calculate_angle() 173
7.69.1.3 calculate_gryo_anglular_velocity() 174
7.69.1.4 get_position() 174
7.69.1.5 init_localization() 174
7.69.1.6 integrate_gyro_w() 175
7.69.1.7 update_position() 175
7.69.2 Variable Documentation 176
7.69.2.1 g1 176
7.69.2.2 last_call 176
7.69.2.3 localization_task 177

7.69.2.4 state_matrix 177
7.70 localization.c 177
7.71 src/log.c File Reference 178
7.71.1 Function Documentation 179
7.71.1.1 debug() 179
7.71.1.2 error() 179
7.71.1.3 info() 180
7.71.1.4 init_error() 181
7.71.1.5 log_info() 181
7.71.1.6 warning() 182
7.71.2 Variable Documentation 182
7.71.2.1 log_lcd 182
7.71.2.2 log_level 183
7.72 log.c 183
7.73 src/matrix.c File Reference 183
7.73.1 Function Documentation 184
7.73.1.1 assert() 184
7.73.1.2 copyMatrix() 185
7.73.1.3 covarianceMatrix() 185
7.73.1.4 dotDiagonalMatrix() 186
7.73.1.5 dotProductMatrix() 187
7.73.1.6 freeMatrix() 188
7.73.1.7 identityMatrix() 189
7.73.1.8 makeMatrix() 189
7.73.1.9 meanMatrix() 190
7.73.1.10 multiplyMatrix() 190
7.73.1.11 printMatrix() 191
7.73.1.12 rowSwap() 192
7.73.1.13 scaleMatrix() 193
7.73.1.14 traceMatrix() 194

7.73.1.15 transposeMatrix() 195

7.74 matrix.c 195

7.75 src/menu.c File Reference 199

7.75.1 Function Documentation 199

7.75.1.1 calculate_current_display() 200

7.75.1.2 create_menu() 200

7.75.1.3 denint_menu() 201

7.75.1.4 display_menu() 202

7.75.1.5 init_menu_float() 203

7.75.1.6 init_menu_int() 203

7.75.1.7 init_menu_var() 204

7.76 menu.c 205

7.77 src/mobile_goal_intake.c File Reference 207

7.77.1 Function Documentation 207

7.77.1.1 lower_intake() 207

7.77.1.2 raise_intake() 208

7.77.1.3 set_intake_motor() 208

7.77.1.4 update_intake() 208

7.78 mobile_goal_intake.c 209

7.79 src/opcontrol.c File Reference 209

7.79.1 Detailed Description 209

7.79.2 Function Documentation 210

7.79.2.1 operatorControl() 210

7.80 opcontrol.c 210

7.81 src/partner.c File Reference 211

7.81.1 Function Documentation 211

7.81.1.1 get_mode() 211

7.81.1.2 update_control() 211

7.81.2 Variable Documentation 212

7.81.2.1 mode 212

7.82 partner.c 212
7.83 src/slew.c File Reference 212
7.83.1 Function Documentation 213
7.83.1.1 deinitslew() 213
7.83.1.2 init_slew() 214
7.83.1.3 set_motor_immediate() 214
7.83.1.4 set_motor_slew() 215
7.83.1.5 updateMotors() 216
7.83.2 Variable Documentation 216
7.83.2.1 initialized 216
7.83.2.2 motors_curr_speeds 217
7.83.2.3 motors_set_speeds 217
7.83.2.4 slew 217
7.83.2.5 speeds_mutex 217
7.84 slew.c 218
7.85 src/toggle.c File Reference 218
7.85.1 Function Documentation 219
7.85.1.1 buttonGetState() 219
7.85.1.2 buttonInit() 220
7.85.1.3 buttonIsNewPress() 221
7.85.2 Variable Documentation 222
7.85.2.1 buttonPressed 222
7.86 toggle.c 222
7.87 src/vlib.c File Reference 223
7.87.1 Function Documentation 224
7.87.1.1 ftoaa() 224
7.87.1.2 itoaa() 225
7.87.1.3 reverse() 226
7.88 vlib.c 226
7.89 src/vmath.c File Reference 227
7.89.1 Function Documentation 227
7.89.1.1 cartesian_cord_to_polar() 228
7.89.1.2 cartesian_to_polar() 228
7.89.1.3 max() 229
7.89.1.4 min() 230
7.89.1.5 sind() 231
7.90 vmath.c 231
7.91 test_code/testMath.py File Reference 231
7.92 testMath.py 232
7.93 testMath.py File Reference 232
7.94 testMath.py 232

Chapter 1

InTheZoneA

Team A code for In The Zone

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

testMath 9

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

_matrix 11
accelerometer_odometry 12
cord
A struct that contains cartesian coordinates 13
encoder_odemtry 14
lcd_buttons
State of the lcd buttons 15
location 16
menu_t
Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu 17
polar_cord
A struct that contains polar coordinates 22

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

testMath.py 232
include/ **auto.h**
Autonomous declarations and macros 25
include/ **battery.h**
Battery management related functions 27
include/ **claw.h**
Code for controlling the claw that grabs the cones 30
include/ **controller.h**
Controller definitions, macros and functions to assist with usig the vex controllers 37
include/ **drive.h**
Drive base definitions and enumerations 42
include/ **encoders.h**
Wrapper around encoder functions 47
include/ **gyro.h** 50
include/ **lcd.h**
LCD wrapper functions and macros 52
include/ **lifter.h**
Declarations and macros for controlling and manipulating the lifter 60
include/ **localization.h**
Declarations and macros for determining the location of the robot. [WIP] 71
include/ **log.h**
Contains logging functions 75
include/ **main.h**
Header file for global functions 82
None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevelant, be sure to use the function to reclaim some of that memory 86
include/ **menu.h**
Contains menu functionality and abstraction 99
include/ **mobile_goal_intake.h** 107
Macros for the different motors ports 109
include/ **partner.h** 114
include/ **potentiometer.h** 116
include/ **sensor_ports.h** 116
include/ **slew.h**
Contains the slew rate controller wrapper for the motors 118

include/ **toggle.h** 124
include/ **vlib.h**
Contains misc helpful functions 129
include/ **vmath.h**
Vex Specific Math Functions, includes: Cartesian to polar cordinates 133
src/ **auto.c**
File for autonomous code 138
src/ **battery.c** 140
src/ **claw.c** 142
src/ **controller.c** 145
src/ **drive.c** 147
src/ **encoders.c** 152
src/ **gyro.c** 154
src/ **init.c**
File for initialization code 155
src/ **lcd.c** 157
src/ **lifter.c** 164
src/ **localization.c** 172
src/ **log.c** 178
src/ **matrix.c** 183
src/ **menu.c** 199
src/ **mobile_goal_intake.c** 207
src/ **opcontrol.c**
File for operator control code 209
src/ **partner.c** 211
src/ **slew.c** 212
src/ **toggle.c** 218
src/ **vlib.c** 223
src/ **vmath.c** 227
test_code/ **testMath.py** 231

Chapter 5

Namespace Documentation

5.1 testMath Namespace Reference

Functions

- def **test** (l1, l2)

5.1.1 Function Documentation

5.1.1.1 test()

```
def testMath.test (
              l1,
              l2 )
```

Definition at line **3** of file **testMath.py**.

```
00003 def test(l1, l2):
00004     print(l1, l2)
00005     print("\n")
00006     theta = l1-l2
00007     x = ((l2)/(theta) + .5) - ((l2)/(theta) + .5) * cos(theta)
00008     y = ((l2)/(theta) + .5) * sin(theta)
00009     print(x)
00010     print(y)
00011     print(degrees(theta))
00012     print("\n")
00013     print("\n")
00014
00015 test(1.0, .5)
00016 test(.5, 1.0)
00017 test(2.0, .5)
00018 test(.5, 2.0)
00019 test(1.0, 3.5)
00020 test(3.5, 1.0)
00021 test(5, .3)
00022 test(.3, 5)
00023 test(1.0, 0)
00024
```

Chapter 6

Data Structure Documentation

6.1 _matrix Struct Reference

```
#include <matrix.h>
```

Data Fields

- double ∗ **data**
- int **height**
- int **width**

6.1.1 Detailed Description

A struct representing a matrix

Definition at line **14** of file **matrix.h**.

6.1.2 Field Documentation

6.1.2.1 data

```
double* _matrix::data
```

Definition at line **17** of file **matrix.h**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **freeMatrix()**, **identityMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, **printMatrix()**, **rowSwap()**, **scaleMatrix()**, **traceMatrix()**, and **transposeMatrix()**.

6.1.2.2 height

```
int _matrix::height
```

Definition at line **15** of file **matrix.h**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, **printMatrix()**, **rowSwap()**, **scaleMatrix()**, **traceMatrix()**, and **transposeMatrix()**.

6.1.2.3 width

```
int _matrix::width
```

Definition at line **16** of file **matrix.h**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, **printMatrix()**, **rowSwap()**, **scaleMatrix()**, **traceMatrix()**, and **transposeMatrix()**.

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

- include/ **matrix.h**

6.2 accelerometer_odometry Struct Reference

Data Fields

- double **x**
- double **y**

6.2.1 Detailed Description

Definition at line **17** of file **localization.c**.

6.2.2 Field Documentation

6.2.2.1 x

```
double accelerometer_odometry::x
```

Definition at line **18** of file **localization.c**.

6.2.2.2 y

```
double accelerometer_odometry::y
```

Definition at line **19** of file **localization.c**.

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

- src/ **localization.c**

6.3 cord Struct Reference

A struct that contains cartesian coordinates.

```
#include <vmath.h>
```

Data Fields

- float **x**
- float **y**

6.3.1 Detailed Description

A struct that contains cartesian coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line **32** of file **vmath.h**.

6.3.2 Field Documentation

6.3.2.1 x

```
float cord::x
```

the x coordinate

Definition at line **34** of file **vmath.h**.

Referenced by **get_joystick_cord()**, and **update_drive_motors()**.

6.3.2.2 y

```
float cord::y
```

the y coordinate

Definition at line **36** of file **vmath.h**.

Referenced by **get_joystick_cord()**, and **update_drive_motors()**.

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

- include/ **vmath.h**

6.4 encoder_odemtry Struct Reference

Data Fields

- double **theta**
- double **x**
- double **y**

6.4.1 Detailed Description

Definition at line **11** of file **localization.c**.

6.4.2 Field Documentation

6.4.2.1 theta

```
double encoder_odemtry::theta
```

Definition at line **14** of file **localization.c**.

Referenced by **integrate_gyro_w()**.

6.4.2.2 x

```
double encoder_odemtry::x
```

Definition at line **12** of file **localization.c**.

6.4.2.3 y

```
double encoder_odemtry::y
```

Definition at line **13** of file **localization.c**.

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

- src/ **localization.c**

6.5 lcd_buttons Struct Reference

represents the state of the lcd buttons

```
#include <lcd.h>
```

Data Fields

- **button_state left**
- **button_state middle**
- **button_state right**

6.5.1 Detailed Description

represents the state of the lcd buttons

Author

Chris Jerrett

Date

9/9/2017

Definition at line **48** of file **lcd.h**.

6.5.2 Field Documentation

6.5.2.1 left

```
button_state lcd_buttons::left
```

Definition at line **49** of file **lcd.h**.

Referenced by **lcd_get_pressed_buttons()**.

6.5.2.2 middle

```
button_state lcd_buttons::middle
```

Definition at line **50** of file **lcd.h**.

Referenced by **lcd_get_pressed_buttons()**.

6.5.2.3 right

```
button_state lcd_buttons::right
```

Definition at line **51** of file **lcd.h**.

Referenced by **lcd_get_pressed_buttons()**.

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

- include/ **lcd.h**

6.6 location Struct Reference

```
#include <localization.h>
```

Data Fields

- int **theta**
- int **x**
- int **y**

6.6.1 Detailed Description

Vector storing the cartesian cords and an angle

Definition at line **23** of file **localization.h**.

6.6.2 Field Documentation

6.6.2.1 theta

`int location::theta`

Definition at line **26** of file **localization.h**.

6.6.2.2 x

`int location::x`

Definition at line **24** of file **localization.h**.

6.6.2.3 y

`int location::y`

Definition at line **25** of file **localization.h**.

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

- include/ **localization.h**

6.7 menu_t Struct Reference

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

`#include <menu.h>`

Data Fields

- int **current**
  contains the current index of menu.
- unsigned int **length**
  contains the length of options char∗∗.
- int **max**
  contains the maximum int value of menu. Defaults to minimum int value
- float **max_f**
  contains the maximum float value of menu. Defaults to minimum int value
- int **min**
  contains the minimum int value of menu. Defaults to minimum int value
- float **min_f**
  contains the minimum float value of menu. Defaults to minimum int value
- char ∗∗ **options**
  contains the array of string options.
- char ∗ **prompt**
  contains the prompt to display on the first line. Step is how much the int menu will increase of decrease with each press. Defaults to one
- int **step**
  contains the step int value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to one
- float **step_f**
  contains the step float value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to 1.0f
- enum **menu_type type**
  contains the type of menu.

6.7.1 Detailed Description

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

Author

Chris Jerrett

Date

9/8/17

See also

menu.h (p. 99)
menu_t (p. 17)
create_menu (p. 200)
init_menu
display_menu (p. 202)
menu_type (p. 100)
denint_menu (p. 201)

Definition at line **67** of file **menu.h**.

6.7.2 Field Documentation

6.7.2.1 current

```
int menu_t::current
```

contains the current index of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line **141** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, **display_menu()**, and **init_menu_int()**.

6.7.2.2 length

```
unsigned int menu_t::length
```

contains the length of options char∗∗.

Author

Chris Jerrett

Date

9/8/17

Definition at line **87** of file **menu.h**.

Referenced by **calculate_current_display()**, and **init_menu_var()**.

6.7.2.3 max

```
int menu_t::max
```

contains the maximum int value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line **103** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, and **init_menu_int()**.

6.7.2.4 max_f

```
float menu_t::max_f
```

contains the maximum float value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line **127** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, and **init_menu_float()**.

6.7.2.5 min

```
int menu_t::min
```

contains the minimum int value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line **95** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, and **init_menu_int()**.

6.7.2.6 min_f

```
float menu_t::min_f
```

contains the minimum float value of menu. Defaults to minimum int value

Author

Chris Jerrett

Date

9/8/17

Definition at line **119** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, and **init_menu_float()**.

6.7.2.7 options

```
char** menu_t::options
```

contains the array of string options.

Author

Chris Jerrett

Date

9/8/17

Definition at line **80** of file **menu.h**.

Referenced by **calculate_current_display()**, **denint_menu()**, and **init_menu_var()**.

6.7.2.8 prompt

```
char* menu_t::prompt
```

contains the prompt to display on the first line. Step is how much the int menu will increase of decrease with each press. Defaults to one

Author

Chris Jerrett

Date

9/8/17

Definition at line **148** of file **menu.h**.

Referenced by **create_menu()**, **denint_menu()**, and **display_menu()**.

6.7.2.9 step

```
int menu_t::step
```

contains the step int value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to one

Author

Chris Jerrett

Date

9/8/17

Definition at line **111** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, and **init_menu_int()**.

6.7.2.10 step_f

```
float menu_t::step_f
```

contains the step float value of menu. Step is how much the int menu will increase of decrease with each press. Defaults to 1.0f

Author

Chris Jerrett

Date

9/8/17

Definition at line **135** of file **menu.h**.

Referenced by **calculate_current_display()**, **create_menu()**, and **init_menu_float()**.

6.7.2.11 type

```
enum  menu_type menu_t::type
```

contains the type of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line **73** of file **menu.h**.

Referenced by **calculate_current_display()**, and **create_menu()**.

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

- include/ **menu.h**

6.8 polar_cord Struct Reference

A struct that contains polar coordinates.

```
#include <vmath.h>
```

Data Fields

- float **angle**
- float **magnitue**

6.8.1 Detailed Description

A struct that contains polar coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line **20** of file **vmath.h**.

6.8.2 Field Documentation

6.8.2.1 angle

```
float polar_cord::angle
```

the angle of the vector

Definition at line **22** of file **vmath.h**.

Referenced by **cartesian_to_polar()**.

6.8.2.2 magnitue

```
float polar_cord::magnitue
```

the magnitude of the vector

Definition at line **24** of file **vmath.h**.

Referenced by **cartesian_to_polar()**.

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

- include/ **vmath.h**

Chapter 7

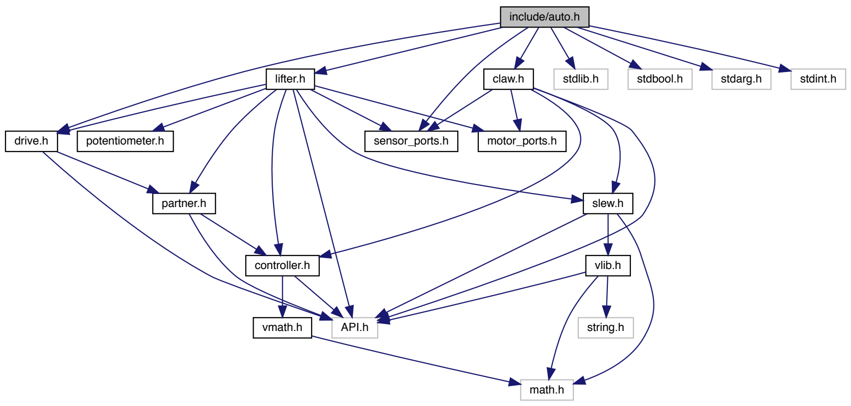
File Documentation

7.1 include/auto.h File Reference

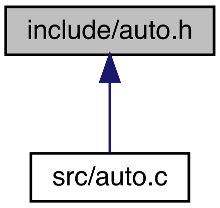
Autonomous declarations and macros.

```
#include "drive.h"
#include "sensor_ports.h"
#include "lifter.h"
#include "claw.h"
```

Include dependency graph for auto.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **FRONT_LEFT_IME** 0

    Front left motor integrated motor encoder.
- #define **GOAL_HEIGHT** 1325

    The height of the goal using potentiometer readings.
- #define **MID_LEFT_DRIVE** 1

    Middle left motor integrated motor encoder.
- #define **MID_RIGHT_DRIVE** 4

    Middle right motor integrated motor encoder.
- #define **STOP_ONE** 500

    First Stop position for stationary autonomous.

7.1.1 Detailed Description

Autonomous declarations and macros.

Author

Chris Jerrett

Date

9/18/2017

Definition in file **auto.h**.

7.1.2 Macro Definition Documentation

7.1.2.1 FRONT_LEFT_IME

```
#define FRONT_LEFT_IME 0
```

Front left motor integrated motor encoder.

Definition at line **18** of file **auto.h**.

7.1.2.2 GOAL_HEIGHT

```
#define GOAL_HEIGHT 1325
```

The height of the goal using potentiometer readings.

Definition at line **38** of file **auto.h**.

Referenced by **autonomous()**.

7.1.2.3 MID_LEFT_DRIVE

```
#define MID_LEFT_DRIVE 1
```

Middle left motor integrated motor encoder.

Definition at line **23** of file **auto.h**.

Referenced by **autonomous()**.

7.1.2.4 MID_RIGHT_DRIVE

```
#define MID_RIGHT_DRIVE 4
```

Middle right motor integrated motor encoder.

Definition at line **28** of file **auto.h**.

Referenced by **autonomous()**.

7.1.2.5 STOP_ONE

```
#define STOP_ONE 500
```

First Stop position for stationary autonomous.

Definition at line **33** of file **auto.h**.

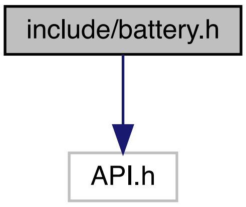
7.2 auto.h

```

#ifndef _AUTO_H_
#define _AUTO_H_

#include "drive.h"
#include "sensor_ports.h"
#include "lifter.h"
#include "claw.h"

#define FRONT_LEFT_IME 0

#define MID_LEFT_DRIVE 1

#define MID_RIGHT_DRIVE 4

#define STOP_ONE 500

#define GOAL_HEIGHT 1325


#endif
```

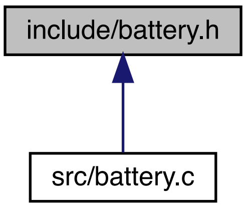
7.3 include/battery.h File Reference

Battery management related functions.

`#include <API.h>`
Include dependency graph for battery.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **MIN_BACKUP_VOLTAGE** 7.8

  The minimum acceptable backup battery voltage beofre a match.
- #define **MIN_MAIN_VOLTAGE** 7.8

  The minimum acceptable main battery voltage beofre a match.

Functions

- double **backup_battery_voltage** ()

  gets the backup battery voltage
- bool **battery_level_acceptable** ()

  returns if the batteries are acceptable
- double **main_battery_voltage** ()

  gets the main battery voltage

7.3.1 Detailed Description

Battery management related functions.

Author

Chris Jerrett

Date

9/18/2017

Definition in file **battery.h**.

7.3.2 Macro Definition Documentation

7.3.2.1 MIN_BACKUP_VOLTAGE

```
#define MIN_BACKUP_VOLTAGE 7.8
```

The minimum acceptable backup battery voltage beofre a match.

Definition at line **20** of file **battery.h**.

Referenced by **battery_level_acceptable()**.

7.3.2.2 MIN_MAIN_VOLTAGE

```
#define MIN_MAIN_VOLTAGE 7.8
```

The minimum acceptable main battery voltage beofre a match.

Definition at line **15** of file **battery.h**.

Referenced by **battery_level_acceptable()**.

7.3.3 Function Documentation

7.3.3.1 backup_battery_voltage()

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

Author

Chris Jerrett

Definition at line **17** of file **battery.c**.

Referenced by **battery_level_acceptable()**.

```
                                 {
  return powerLevelBackup() / 1000.0;
}
```

7.3.3.2 battery_level_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

See also

MIN_MAIN_VOLTAGE (p. 28)
MIN_BACKUP_VOLTAGE (p. 28)

Author

Chris Jerrett

Definition at line **28** of file **battery.c**.

References **backup_battery_voltage()**, **main_battery_voltage()**, **MIN_BACKUP_VOLTAGE**, and **MIN_MAI↩N_VOLTAGE**.

```
                                 {
  if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
  if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
  return true;
}
```

7.3.3.3 main_battery_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

Author

Chris Jerrett

Definition at line **9** of file **battery.c**.

Referenced by **battery_level_acceptable()**.

```
00009                              {
00010   return powerLevelMain() / 1000.0;
00011 }
```

7.4 battery.h

```

#ifndef _BATTERY_H_
#define _BATTERY_H_

#include <API.h>

#define MIN_MAIN_VOLTAGE 7.8

#define MIN_BACKUP_VOLTAGE 7.8

double main_battery_voltage();

double backup_battery_voltage();

bool battery_level_acceptable();

#endif
```

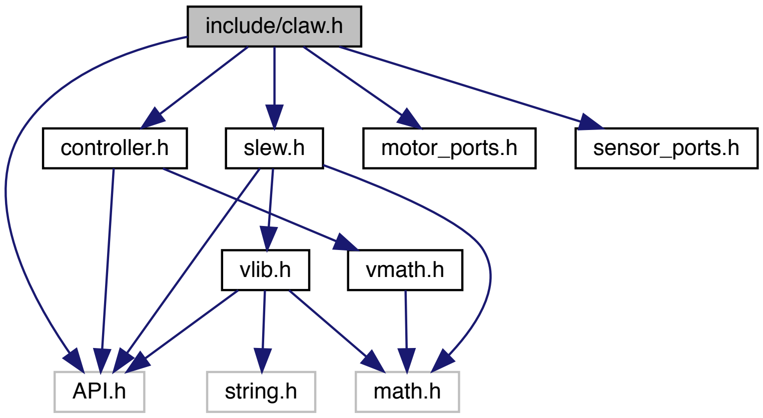
7.5 include/claw.h File Reference

Code for controlling the claw that grabs the cones.

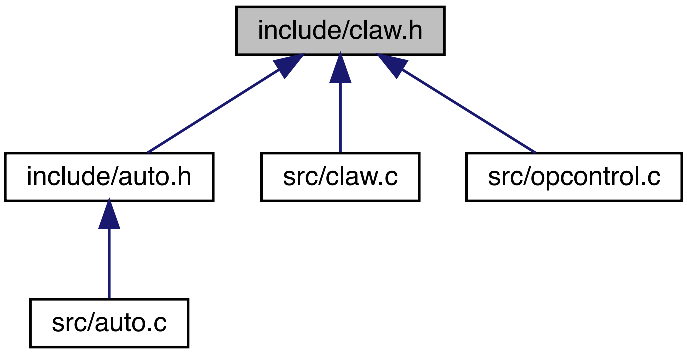
```
#include "slew.h"
#include <API.h>
#include "controller.h"
#include "motor_ports.h"
```

`#include "sensor_ports.h"`

Include dependency graph for claw.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **CLAW_CLOSE MASTER**, 6, JOY_UP

  The joystick parameters for closing the claw.
- #define **CLAW_CLOSE_VAL** 3000

  The potentiometer value for a closed claw.
- #define **CLAW_OPEN MASTER**, 6, JOY_DOWN

  The joystick parameters for opening the claw.
- #define **CLAW_OPEN_VAL** 1500

  The potentiometer value for a open claw.

- #define **MAX_CLAW_SPEED** 127

  The max motor vlaue of the claw.
- #define **MIN_CLAW_SPEED** -127

  The min motor vlaue of the claw.

Enumerations

- enum **claw_state** { **CLAW_OPEN_STATE**, **CLAW_CLOSE_STATE**, **CLAW_NEUTRAL_STATE** }

  The different states of the claw.

Functions

- void **close_claw** ()

  Drives the motors to close the claw.
- unsigned int **getClawTicks** ()

  Gets the claw position in potentiometer ticks.
- void **open_claw** ()

  Drives the motors to open the claw.
- void **set_claw_motor** (const int v)

  sets the claw motor speed
- void **update_claw** ()

  Updates the claw motor values.

7.5.1 Detailed Description

Code for controlling the claw that grabs the cones.

Author

Chris Jerrett, Christian Desimone

Date

8/30/2017

Definition in file **claw.h**.

7.5.2 Macro Definition Documentation

7.5.2.1 CLAW_CLOSE

```
#define CLAW_CLOSE  MASTER, 6, JOY_UP
```

The joystick parameters for closing the claw.

Author

Chris Jerrett

Definition at line **31** of file **claw.h**.

Referenced by **update_claw()**.

7.5.2.2 CLAW_CLOSE_VAL

```
#define CLAW_CLOSE_VAL 3000
```

The potentiometer value for a closed claw.

Author

Chris Jerrett

Definition at line **43** of file **claw.h**.

7.5.2.3 CLAW_OPEN

```
#define CLAW_OPEN  MASTER, 6, JOY_DOWN
```

The joystick parameters for opening the claw.

Author

Chris Jerrett

Definition at line **37** of file **claw.h**.

Referenced by **update_claw()**.

7.5.2.4 CLAW_OPEN_VAL

```
#define CLAW_OPEN_VAL 1500
```

The potentiometer value for a open claw.

Author

Chris Jerrett

Definition at line **49** of file **claw.h**.

7.5.2.5 MAX_CLAW_SPEED

```
#define MAX_CLAW_SPEED 127
```

The max motor vlaue of the claw.

Author

Chris Jerrett

Definition at line **20** of file **claw.h**.

Referenced by **open_claw()**, and **update_claw()**.

7.5.2.6 MIN_CLAW_SPEED

```
#define MIN_CLAW_SPEED -127
```

The min motor vlaue of the claw.

Author

Chris Jerrett

Definition at line **25** of file **claw.h**.

Referenced by **close_claw()**, and **update_claw()**.

7.5.3 Enumeration Type Documentation

7.5.3.1 claw_state

```
enum claw_state
```

The different states of the claw.

Author

Chris Jerrett

Enumerator

CLAW_OPEN_STATE	
CLAW_CLOSE_STATE	
CLAW_NEUTRAL_STATE	

Definition at line **85** of file **claw.h**.

```
                {
  CLAW_OPEN_STATE,
  CLAW_CLOSE_STATE,
  CLAW_NEUTRAL_STATE
};
```

7.5.4 Function Documentation

7.5.4.1 close_claw()

```
void close_claw ( )
```

Drives the motors to close the claw.

Author

Chris Jerrett

Definition at line **48** of file **claw.c**.

References **CLAW_MOTOR**, **MIN_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
                  {
  set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
}
```

7.5.4.2 getClawTicks()

```
unsigned int getClawTicks ( )
```

Gets the claw position in potentiometer ticks.

Author

Chris Jerrett

7.5.4.3 open_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

Author

Chris Jerrett

Definition at line **40** of file **claw.c**.

References **CLAW_MOTOR**, **MAX_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00040                  {
00041   set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);
00042 }
```

7.5.4.4 set_claw_motor()

```
void set_claw_motor (
              const int v )
```

sets the claw motor speed

Author

Chris Jerrett

Definition at line **31** of file **claw.c**.

References **CLAW_MOTOR**, and **set_motor_immediate()**.

Referenced by **update_claw()**.

```
00031                                  {
00032   set_motor_immediate(CLAW_MOTOR, v);
00033 }
```

7.5.4.5 update_claw()

```
void update_claw ( )
```

Updates the claw motor values.

Author

Chris Jerrett

Definition at line **9** of file **claw.c**.

References **CLAW_CLOSE**, **CLAW_CLOSE_STATE**, **CLAW_NEUTRAL_STATE**, **CLAW_OPEN**, **CLAW_O↩PEN_STATE**, **MAX_CLAW_SPEED**, **MIN_CLAW_SPEED**, **set_claw_motor()**, and **state**.

Referenced by **operatorControl()**.

```
00009                    {
00010   if(joystickGetDigital(CLAW_CLOSE)) {
00011     state = CLAW_CLOSE_STATE;
00012   } else if(joystickGetDigital(CLAW_OPEN)) {
00013     state = CLAW_OPEN_STATE;
00014   } else {
00015     state = CLAW_NEUTRAL_STATE;
00016   }
00017
00018   if(state == CLAW_CLOSE_STATE) {
00019     set_claw_motor(MAX_CLAW_SPEED);
00020   } else if(state == CLAW_OPEN_STATE) {
00021     set_claw_motor(MIN_CLAW_SPEED);
00022   } else {
00023     set_claw_motor(0);
00024   }
00025 }
```

7.6 claw.h

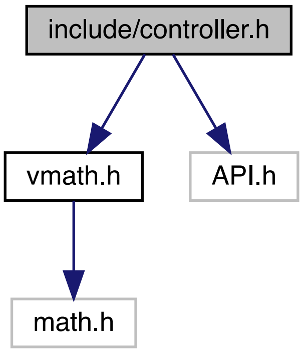
```
00001
00007 #ifndef _CLAW_H_
00008 #define _CLAW_H_
00009
00010 #include "slew.h"
00011 #include <API.h>
00012 #include "controller.h"
00013 #include "motor_ports.h"
00014 #include "sensor_ports.h"
00015
00020 #define MAX_CLAW_SPEED 127
00021
00025 #define MIN_CLAW_SPEED -127
00026
00031 #define CLAW_CLOSE MASTER, 6, JOY_UP
00032
00037 #define CLAW_OPEN MASTER, 6, JOY_DOWN
00038
00043 #define CLAW_CLOSE_VAL 3000
00044
00049 #define CLAW_OPEN_VAL 1500
00050
00055 void update_claw();
00056
00061 void set_claw_motor(const int v);
00062
00067 unsigned int getClawTicks();
00068
00073 void open_claw();
00074
00079 void close_claw();
00080
00085 enum claw_state {
00086   CLAW_OPEN_STATE,
00087   CLAW_CLOSE_STATE,
00088   CLAW_NEUTRAL_STATE
00089 };
00090
00091 #endif
```

7.7 include/controller.h File Reference

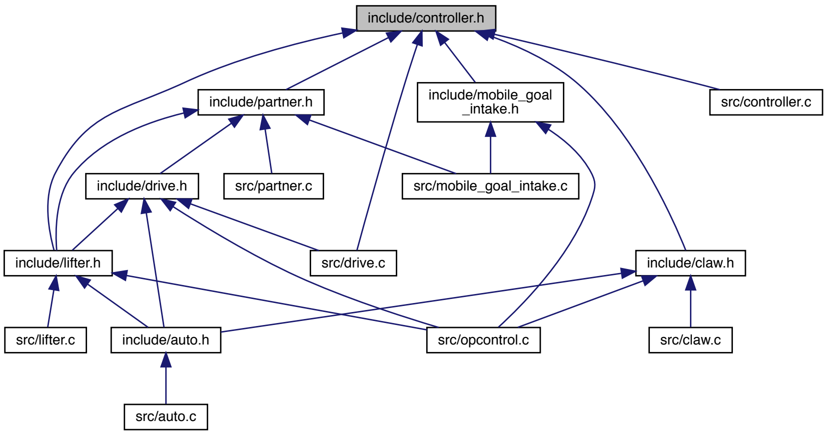
controller definitions, macros and functions to assist with usig the vex controllers.

```
#include "vmath.h"
#include <API.h>
```

Include dependency graph for controller.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **LEFT_BUMPERS** 6
- #define **LEFT_BUTTONS** 7
- #define **LEFT_JOY_X** 4

    the left x joystick on controller

- #define **LEFT_JOY_Y** 3
  the left y joystick on controller
- #define **MASTER** 1
  the master controller
- #define **PARTNER** 2
  the slave/partner controller
- #define **RIGHT_BUMPERS** 5
- #define **RIGHT_BUTTONS** 8
- #define **RIGHT_JOY_X** 1
  the right x joystick on controller
- #define **RIGHT_JOY_Y** 2
  the right y joystick on controller

Enumerations

- enum **joystick** { **RIGHT_JOY**, **LEFT_JOY** }
  Represents a joystick on the controller.

Functions

- struct **cord** **get_joystick_cord** (enum **joystick** **side**, int controller)
  Gets the location of a joystick on the controller.

7.7.1 Detailed Description

controller definitions, macros and functions to assist with usig the vex controllers.

Author

Chris Jerrett, Christian Desimone

Date

9/9/2017

Definition in file **controller.h**.

7.7.2 Macro Definition Documentation

7.7.2.1 LEFT_BUMPERS

```
#define LEFT_BUMPERS 6
```

Definition at line **18** of file **controller.h**.

7.7.2.2 LEFT_BUTTONS

```
#define LEFT_BUTTONS 7
```

Definition at line **16** of file **controller.h**.

7.7.2.3 LEFT_JOY_X

```
#define LEFT_JOY_X 4
```

the left x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **53** of file **controller.h**.

Referenced by **get_joystick_cord()**.

7.7.2.4 LEFT_JOY_Y

```
#define LEFT_JOY_Y 3
```

the left y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **60** of file **controller.h**.

Referenced by **get_joystick_cord()**.

7.7.2.5 MASTER

```
#define MASTER 1
```

the master controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **25** of file **controller.h**.

Referenced by **update_drive_motors()**, and **update_intake()**.

7.7.2.6 PARTNER

```
#define PARTNER 2
```

the slave/partner controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **32** of file **controller.h**.

Referenced by **update_control()**, and **update_drive_motors()**.

7.7.2.7 RIGHT_BUMPERS

```
#define RIGHT_BUMPERS 5
```

Definition at line **17** of file **controller.h**.

7.7.2.8 RIGHT_BUTTONS

```
#define RIGHT_BUTTONS 8
```

Definition at line **15** of file **controller.h**.

7.7.2.9 RIGHT_JOY_X

```
#define RIGHT_JOY_X 1
```

the right x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **39** of file **controller.h**.

Referenced by **get_joystick_cord()**.

7.7.2.10 RIGHT_JOY_Y

```
#define RIGHT_JOY_Y 2
```

the right y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **46** of file **controller.h**.

Referenced by **get_joystick_cord()**.

7.7.3 Enumeration Type Documentation

7.7.3.1 joystick

```
enum joystick
```

Represents a joystick on the controller.

Date

9/10/2017

Author

Chris Jerrett

Enumerator

RIGHT_JOY	The right joystick
LEFT_JOY	The left joystick

Definition at line **67** of file **controller.h**.

```
                {
  RIGHT_JOY,
  LEFT_JOY,
};
```

7.7.4 Function Documentation

7.7.4.1 get_joystick_cord()

```
struct cord get_joystick_cord (
            enum joystick side,
            int controller )
```

Gets the location of a joystick on the controller.

Author

Chris Jerrett

Definition at line **7** of file **controller.c**.

References **LEFT_JOY_X**, **LEFT_JOY_Y**, **RIGHT_JOY**, **RIGHT_JOY_X**, **RIGHT_JOY_Y**, **cord::x**, and **cord↩ ::y**.

```
                                                          {
  int x;
  int y;
  //Get the joystick value for either the right or left,
  //depending on the mode
  if(side == RIGHT_JOY) {
    y = joystickGetAnalog(controller, RIGHT_JOY_X);
    x = joystickGetAnalog(controller, RIGHT_JOY_Y);
  } else {
    y = joystickGetAnalog(controller, LEFT_JOY_X);
    x = joystickGetAnalog(controller, LEFT_JOY_Y);
  }
  //Define a coordinate for the joystick value
  struct cord c;
  c.x = x;
  c.y = y;
  return c;
}
```

7.8 controller.h

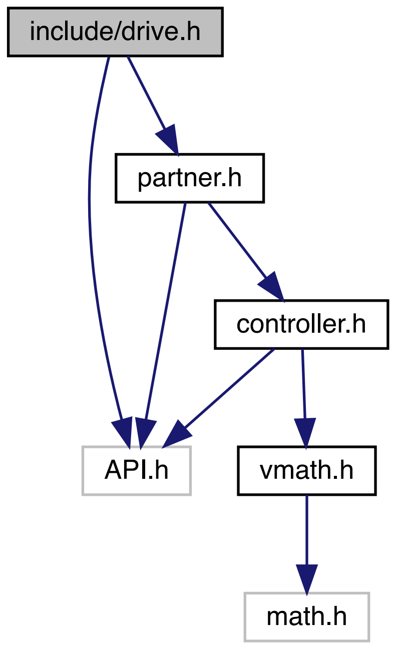
```

#ifndef _CONTROLLER_H_
#define _CONTROLLER_H_

#include "vmath.h"
#include <API.h>

#define RIGHT_BUTTONS 8
#define LEFT_BUTTONS 7
#define RIGHT_BUMPERS 5
#define LEFT_BUMPERS 6

#define MASTER 1

#define PARTNER 2

#define RIGHT_JOY_X 1

#define RIGHT_JOY_Y 2

#define LEFT_JOY_X 4

#define LEFT_JOY_Y 3

enum joystick {
  RIGHT_JOY,
  LEFT_JOY,
};

struct cord get_joystick_cord(enum joystick side, int controller);

#endif
```

7.9 include/drive.h File Reference

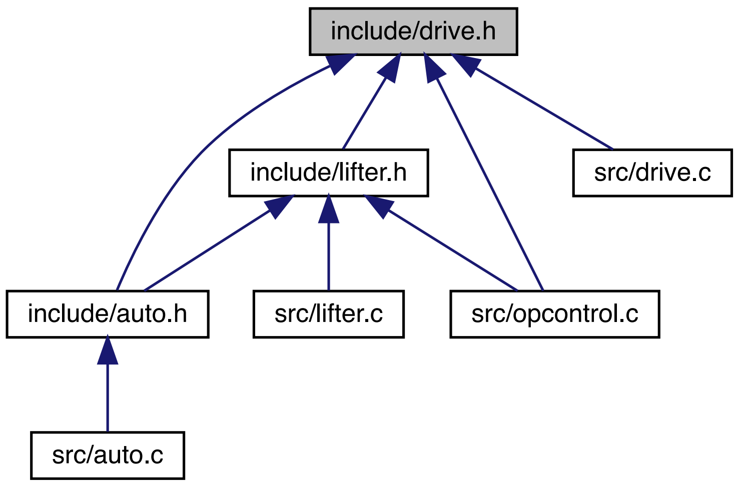
Drive base definitions and enumerations.

```
#include <API.h>
#include "partner.h"
```

Include dependency graph for drive.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **THRESHOLD** 10

  The dead spot on the controller to avoid running motors at low speeds.

Typedefs

- typedef enum **side** **side_t**

  enumeration indication side of the robot.

Enumerations

- enum **side** { **LEFT**, **BOTH**, **RIGHT** }

  enumeration indication side of the robot.

Functions

- void **set_side_speed** (**side_t** **side**, int speed)

  sets the speed of one side of the robot.
- void **setThresh** (int t)

  Sets the deadzone threshhold on the drive.
- void **update_drive_motors** ()

  Updates the drive motors during teleop.

7.9.1 Detailed Description

Drive base definitions and enumerations.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **drive.h**.

7.9.2 Macro Definition Documentation

7.9.2.1 THRESHOLD

```
#define THRESHOLD 10
```

The dead spot on the controller to avoid running motors at low speeds.

Definition at line **18** of file **drive.h**.

Referenced by **joystickExp()**.

7.9.3 Typedef Documentation

7.9.3.1 side_t

```
typedef enum side side_t
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

9/7/2017 Side can be right, both of left. Contained in side typedef, so enum is unnecessary.

7.9.4 Enumeration Type Documentation

7.9.4.1 side

```
enum side
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

9/7/2017 Side can be right, both of left. Contained in side typedef, so enum is unnecessary.

Enumerator

LEFT	
BOTH	
RIGHT	

Definition at line **26** of file **drive.h**.

```
                  {
  LEFT,
  BOTH,
  RIGHT
} side_t;
```

7.9.5 Function Documentation

7.9.5.1 set_side_speed()

```
void set_side_speed (
                side_t side,
               int speed )
```

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

side	a side enum which indicates the size.
speed	the speed of the side. Can range from -127 - 127 negative being back and positive forwards

Definition at line **68** of file **drive.c**.

References **BOTH**, **LEFT**, **MOTOR_BACK_LEFT**, **MOTOR_BACK_RIGHT**, **MOTOR_FRONT_LEFT**, **MOTOR_FRONT_RIGHT**, **MOTOR_MIDDLE_LEFT**, **MOTOR_MIDDLE_RIGHT**, **RIGHT**, and **set_motor_slew()**.

Referenced by **autonomous()**, and **update_drive_motors()**.

```
                                               {
  if(side == RIGHT || side == BOTH){
    set_motor_slew(MOTOR_BACK_RIGHT , -speed);
    set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
    set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
  }
  if(side == LEFT || side == BOTH){
    set_motor_slew(MOTOR_BACK_LEFT, speed);
    set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
    set_motor_slew(MOTOR_FRONT_LEFT, speed);
  }
}
```

7.9.5.2 setThresh()

```
void setThresh (
             int t )
```

Sets the deadzone threshhold on the drive.

Author

Chris Jerrett
Christian Desimone

Definition at line **25** of file **drive.c**.

References **thresh**.

```
                   {
  thresh = t;
}
```

7.9.5.3 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

Definition at line **34** of file **drive.c**.

References **get_mode()**, **LEFT**, **MASTER**, **PARTNER**, **PARTNER_CONTROLLER_MODE**, **RIGHT**, **set_side_speed()**, **thresh**, **cord::x**, and **cord::y**.

Referenced by **operatorControl()**.

```
                         {
  //Get the joystick values from the controller
  int x = 0;
  int y = 0;
  if(get_mode() == PARTNER_CONTROLLER_MODE) {
    x = (joystickGetAnalog(PARTNER, 3));
    y = (joystickGetAnalog(PARTNER, 1));
  } else {
    x = -(joystickGetAnalog(MASTER, 3));
    y = (joystickGetAnalog(MASTER, 1));
  }
  //Make sure the joystick values are significant enough to change the motors
  if(x < thresh && x > -thresh){
    x = 0;
  }
  if(y < thresh && y > -thresh){
    y = 0;
  }
  //Create motor values for the left and right from the x and y of the joystick
  int r = (x + y);
  int l = -(x - y);

  //Set the drive motors
  set_side_speed(LEFT, l);
  set_side_speed(RIGHT, -r);

}
```

7.10 drive.h

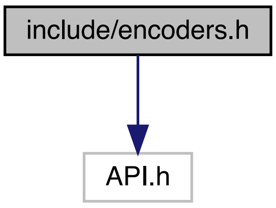
```

#ifndef _DRIVE_H_
#define _DRIVE_H_

#include <API.h>
#include "partner.h"

#define THRESHOLD 10

typedef enum side{
  LEFT,
  BOTH,
  RIGHT
} side_t;

void set_side_speed(side_t side, int speed);

void setThresh(int t);

void update_drive_motors();

#endif
```

7.11 include/encoders.h File Reference

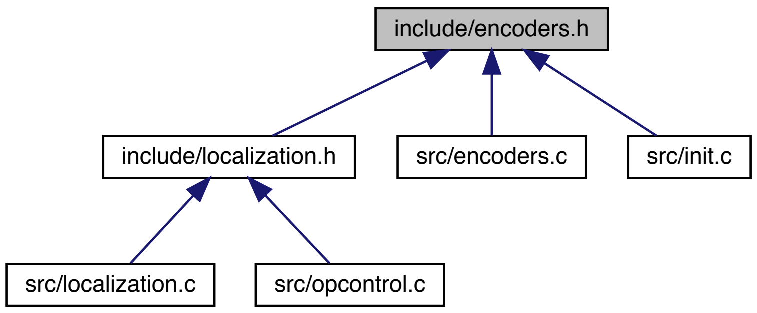
wrapper around encoder functions

`#include <API.h>`

Include dependency graph for encoders.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **IME_NUMBER** 6

  The number of IMEs. This number is compared against the number detect in init_encoders.

Functions

- int **get_encoder_ticks** (unsigned char address)

  Gets the encoder ticks since last reset.
- int **get_encoder_velocity** (unsigned char address)

  Gets the encoder reads.
- bool **init_encoders** ()

  Initializes all motor encoders.

7.11.1 Detailed Description

wrapper around encoder functions

Author

Chris Jerrett, Christian Desimone

Date

9/9/2017

Definition in file **encoders.h**.

7.11.2 Macro Definition Documentation

7.11.2.1 IME_NUMBER

```
#define IME_NUMBER 6
```

The number of IMEs. This number is compared against the number detect in init_encoders.

See also

init_encoders() (p. 49)

Author

Chris Jerrett

Date

9/9/2017

See also

IME_NUMBER (p. 48)

Definition at line **20** of file **encoders.h**.

Referenced by **init_encoders()**.

7.11.3 Function Documentation

7.11.3.1 get_encoder_ticks()

```
int get_encoder_ticks (
            unsigned char address )
```

Gets the encoder ticks since last reset.

Author

Chris Jerrett

Date

9/15/2017

Definition at line **23** of file **encoders.c**.

```
00023                                             {
00024   int i = 0;
00025   imeGet(address, &i);
00026   return i;
00027 }
```

7.11.3.2 get_encoder_velocity()

```
int get_encoder_velocity (
            unsigned char address )
```

Gets the encoder reads.

Author

Chris Jerrett

Date

9/15/2017

Definition at line **34** of file **encoders.c**.

```
                                                {
  int i = 0;
  imeGetVelocity(address, &i);
  return i;
}
```

7.11.3.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

IME_NUMBER (p. 48)

Definition at line **10** of file **encoders.c**.

References **IME_NUMBER**.

```
                      {
  #ifdef IME_NUMBER
  return imeInitializeAll() == IME_NUMBER;
  #else
  return imeInitializeAll();
  #endif
}
```

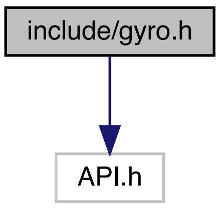
7.12 encoders.h

```

#ifndef _ENCODERS_H_
#define _ENCODERS_H_

#include <API.h>

#define IME_NUMBER 6

bool init_encoders();

int get_encoder_ticks(unsigned char address);

int get_encoder_velocity(unsigned char address);

#endif
```

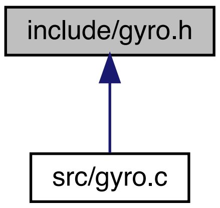
7.13 include/gyro.h File Reference

`#include "API.h"`

Include dependency graph for gyro.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **GYRO_MULTIPLIER** 0
- #define **GYRO_PORT** 1

Functions

- float **get_main_gyro_angluar_velocity** ()
- bool **init_main_gyro** ()

7.13.1 Macro Definition Documentation

7.13.1.1 GYRO_MULTIPLIER

```
#define GYRO_MULTIPLIER 0
```

Definition at line **7** of file **gyro.h**.

Referenced by **init_main_gyro()**.

7.13.1.2 GYRO_PORT

```
#define GYRO_PORT 1
```

Definition at line **6** of file **gyro.h**.

Referenced by **get_main_gyro_angluar_velocity()**, and **init_main_gyro()**.

7.13.2 Function Documentation

7.13.2.1 get_main_gyro_angluar_velocity()

```
float get_main_gyro_angluar_velocity ( )
```

Definition at line **11** of file **gyro.c**.

References **GYRO_PORT**.

```
00011                                        {
00012   uint32_t port = GYRO_PORT;
00013   int32_t reading = (int32_t)analogReadCalibratedHR(port + 1);
00014     return 0;
00015 }
```

7.13.2.2 init_main_gyro()

```
bool init_main_gyro ( )
```

Definition at line **5** of file **gyro.c**.

References **GYRO_MULTIPLIER**, **GYRO_PORT**, and **main_gyro**.

```
00005                        {
00006   main_gyro = gyroInit(GYRO_PORT, GYRO_MULTIPLIER);
00007   return main_gyro != NULL;
00008 }
```

7.14 gyro.h

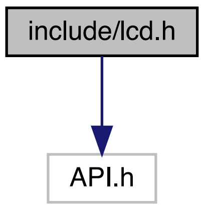
```
#ifndef _GYRO_H_
#define _GYRO_H_

#include "API.h"

#define GYRO_PORT 1
#define GYRO_MULTIPLIER 0

bool init_main_gyro();
float get_main_gyro_angluar_velocity();

#endif
```

7.15 include/lcd.h File Reference

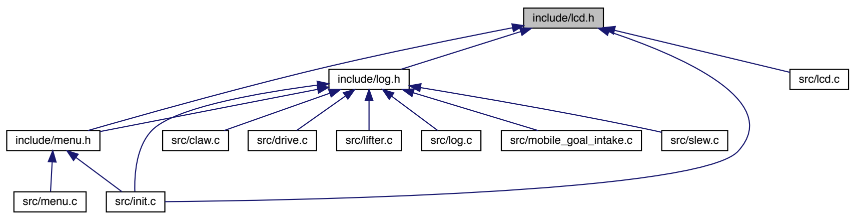
LCD wrapper functions and macros.

```
#include <API.h>
```

Include dependency graph for lcd.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **lcd_buttons**

  represents the state of the lcd buttons

Macros

- #define **BOTTOM_ROW** 2

  The bottom row on the lcd screen.
- #define **TOP_ROW** 1

  The top row on the lcd screen.

Enumerations

- enum **button_state** { **RELEASED** = false, **PRESSED** = true }

  Represents the state of a button.

Functions

- void **init_main_lcd** (FILE ∗lcd)

  Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- void **lcd_clear** ()

  Clears the lcd.
- **lcd_buttons** **lcd_get_pressed_buttons** ()

  Returns the pressed buttons.
- void **lcd_print** (unsigned int line, const char ∗str)

  prints a string to a line on the lcd
- void **lcd_printf** (unsigned int line, const char ∗format_str,...)

  prints a formated string to a line on the lcd. Smilar to printf
- void **lcd_set_backlight** (bool **state**)

  sets the backlight of the lcd
- void **promt_confirmation** (const char ∗confirm_text)

  Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

7.15.1 Detailed Description

LCD wrapper functions and macros.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **lcd.h**.

7.15.2 Macro Definition Documentation

7.15.2.1 BOTTOM_ROW

```
#define BOTTOM_ROW 2
```

The bottom row on the lcd screen.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **25** of file **lcd.h**.

Referenced by **log_info()**.

7.15.2.2 TOP_ROW

```
#define TOP_ROW 1
```

The top row on the lcd screen.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **18** of file **lcd.h**.

Referenced by **display_menu()**, and **log_info()**.

7.15.3 Enumeration Type Documentation

7.15.3.1 button_state

```
enum button_state
```

Represents the state of a button.

A button can be pressed of RELEASED. Release is false which is also 0. PRESSED is true or 1.

Author

Chris Jerrett

Date

9/9/2017

Enumerator

RELEASED	A released button
PRESSED	A pressed button

Definition at line **36** of file **lcd.h**.

```
             {
  RELEASED = false,
  PRESSED = true,
} button_state;
```

7.15.4 Function Documentation

7.15.4.1 init_main_lcd()

```
void init_main_lcd (
            FILE * lcd )
```

Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.

Parameters

lcd	the urart port of the lcd screen

See also

uart1
uart2

Author

Chris Jerrett

Date

9/9/2017

Definition at line **61** of file **lcd.c**.

References **lcd_clear()**, and **lcd_port**.

Referenced by **initialize()**.

```
                               {
  lcd_port = lcd;
  lcdInit(lcd);
  lcd_clear();
}
```

7.15.4.2 lcd_clear()

```
void lcd_clear ( )
```

Clears the lcd.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **47** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

Referenced by **display_menu()**, and **init_main_lcd()**.

```
                  {
  lcd_assert();
  lcdClear(lcd_port);
}
```

7.15.4.3 lcd_get_pressed_buttons()

```
lcd_buttons lcd_get_pressed_buttons ( )
```

Returns the pressed buttons.

Returns

a struct containing the states of all three buttons.

Author

Chris Jerrett

Date

9/9/2017

See also

lcd_buttons (p. 15)

Definition at line **28** of file **lcd.c**.

References **lcd_assert()**, **lcd_port**, **lcd_buttons::left**, **lcd_buttons::middle**, **PRESSED**, **RELEASED**, and **lcd_buttons::right**.

Referenced by **display_menu()**, and **promt_confirmation()**.

```
                                      {
  lcd_assert();
  unsigned int btn_binary = lcdReadButtons(lcd_port);
  bool left = btn_binary & 0x1;//0001
  bool middle = btn_binary & 0x2;//0010
  bool right = btn_binary & 0x4;//0100
  lcd_buttons btns;
  btns.left = left ? PRESSED : RELEASED;
  btns.middle = middle ? PRESSED : RELEASED;
  btns.right = right ? PRESSED : RELEASED;

  return btns;
}
```

7.15.4.4 lcd_print()

```
void lcd_print (
            unsigned int line,
            const char * str )
```

prints a string to a line on the lcd

Parameters

line	the line to print on
str	string to print

Author

Chris Jerrett

Date

9/9/2017

Definition at line **74** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

Referenced by **display_menu()**, and **promt_confirmation()**.

```
00074                                              {
00075   lcd_assert();
00076   lcdSetText(lcd_port, line, str);
00077 }
```

7.15.4.5 lcd_printf()

```
void lcd_printf (
            unsigned int line,
            const char * format_str,
             ... )
```

prints a formated string to a line on the lcd. Smilar to printf

Parameters

line	the line to print on
format_str	format string string to print

Author

Chris Jerrett

Date

9/9/2017

Definition at line **86** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

```
00086                                                                    {
00087   lcd_assert();
00088   lcdPrint(lcd_port, line, format_str);
00089 }
```

7.15.4.6 lcd_set_backlight()

```
void lcd_set_backlight (
            bool state )
```

sets the backlight of the lcd

Parameters

state	a boolean representing the state of the backlight. true = on, false = off.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **97** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

```
00097                                      {
00098   lcd_assert();
00099   lcdSetBacklight(lcd_port, state);
00100 }
```

7.15.4.7 promt_confirmation()

```
void promt_confirmation (
            const char * confirm_text )
```

Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Parameters

confirm_text	the text for the user to confirm.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **111** of file **lcd.c**.

References **lcd_assert()**, **lcd_get_pressed_buttons()**, **lcd_print()**, and **PRESSED**.

```
00111                                                    {
00112   lcd_assert();
00113   lcd_print(1, confirm_text);
00114   while(lcd_get_pressed_buttons().middle != PRESSED){
00115     delay(200);
00116   }
00117 }
```

7.16 lcd.h

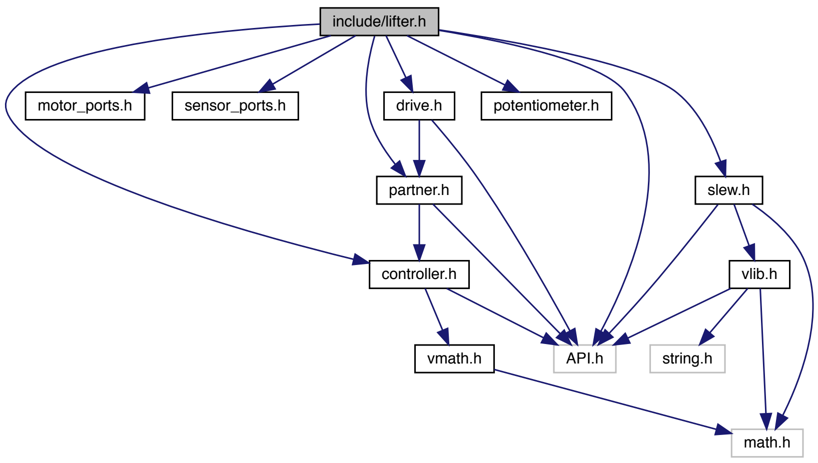
```

#ifndef _LCD_H_
#define _LCD_H_

#include <API.h>

#define TOP_ROW 1

#define BOTTOM_ROW 2

typedef enum {
  RELEASED = false,
  PRESSED = true,
} button_state;

typedef struct {
  button_state left;
  button_state middle;
  button_state right;
} lcd_buttons;


lcd_buttons lcd_get_pressed_buttons();

void lcd_clear();

void init_main_lcd(FILE *lcd);

void lcd_print(unsigned int line, const char *str);

void lcd_printf(unsigned int line, const char *format_str, ...);

void lcd_set_backlight(bool state);

void promt_confirmation(const char *confirm_text);

#endif
```

7.17 include/lifter.h File Reference

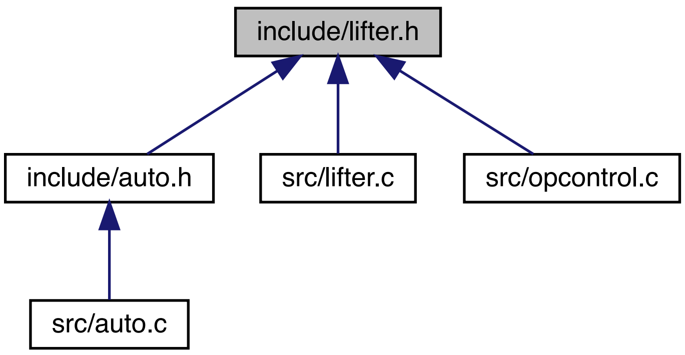
Declarations and macros for controlling and manipulating the lifter.

```
#include <API.h>
#include "motor_ports.h"
#include "sensor_ports.h"
#include "slew.h"
#include "controller.h"
#include "potentiometer.h"
#include "partner.h"
#include "drive.h"
```

Include dependency graph for lifter.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **HEIGHT** 19.1 - 3.8

  The integral constant for the lifter PID.
- #define **INIT_ROTATION** 680

  The initial rotation of the lifter potentiometer at height zero.
- #define **LIFTER_D** 0

  The derivative constant for the lifter PID.
- #define **LIFTER_DOWN** **MASTER**, 5, JOY_DOWN

  The lifter down controller params.
- #define **LIFTER_DOWN_PARTNER** **PARTNER**, 5, JOY_DOWN

  The lifter down controller params for the partner.
- #define **LIFTER_DRIVER_LOAD** **MASTER**, **RIGHT_BUTTONS**, JOY_RIGHT

  Height to raise lifter to driver preload height.
- #define **LIFTER_I** 0

  The integral constant for the lifter PID.
- #define **LIFTER_P** .15

  The proportional constant for the lifter PID.
- #define **LIFTER_UP** **MASTER**, 5, JOY_UP

  The lifter up controller params.
- #define **LIFTER_UP_PARTNER** **PARTNER**, 5, JOY_UP

  The lifter up controller params for the partner.
- #define **MAIN_LIFTER_MIN_HEIGHT** 1700
- #define **MAIN_LIFTER_POT** 1
- #define **SECONDARY_LIFTER_MAX_HEIGHT** 2500
- #define **SECONDARY_LIFTER_MIN_HEIGHT** 1300
- #define **SECONDARY_LIFTER_POT_PORT** 2
- #define **THRESHOLD** 10

  The threshold of a signficant speed for the lifter.

Functions

- double **getLifterHeight** ()

  Gets the height of the lifter in inches.
- int **getLifterTicks** ()

  Gets the value of the lifter pot.
- float **lifterPotentiometerToDegree** (int x)

  height of the lifter in degrees from 0 height
- void **lower_main_lifter** ()

  Lowers the main lifter.
- void **lower_secondary_lifter** ()

  Lowers the secondary lifter.
- void **raise_main_lifter** ()

  Raises the main lifter.
- void **raise_secondary_lifter** ()

  Raises the main lifter.
- void **set_lifter_pos** (int pos)

  Sets the lifter positions to the given value.
- void **set_main_lifter_motors** (const int v)

  Sets the main lifter motors to the given value.
- void **set_secondary_lifter_motors** (const int v)

  Sets the secondary lifter motors to the given value.
- void **update_lifter** ()

  Updates the lifter in teleop.

7.17.1 Detailed Description

Declarations and macros for controlling and manipulating the lifter.

Author

Chris Jerrett, Christian Desimone

Date

8/27/2017

Definition in file **lifter.h**.

7.17.2 Macro Definition Documentation

7.17.2.1 HEIGHT

```
#define HEIGHT 19.1 - 3.8
```

The integral constant for the lifter PID.

Definition at line **48** of file **lifter.h**.

7.17.2.2 INIT_ROTATION

```
#define INIT_ROTATION 680
```

The initial rotation of the lifter potentiometer at height zero.

Definition at line **22** of file **lifter.h**.

Referenced by **lifterPotentiometerToDegree()**.

7.17.2.3 LIFTER_D

```
#define LIFTER_D 0
```

The derivative constant for the lifter PID.

Definition at line **32** of file **lifter.h**.

7.17.2.4 LIFTER_DOWN

```
#define LIFTER_DOWN  MASTER, 5, JOY_DOWN
```

The lifter down controller params.

Definition at line **58** of file **lifter.h**.

Referenced by **update_lifter()**.

7.17.2.5 LIFTER_DOWN_PARTNER

```
#define LIFTER_DOWN_PARTNER  PARTNER, 5, JOY_DOWN
```

The lifter down controller params for the partner.

Definition at line **73** of file **lifter.h**.

7.17.2.6 LIFTER_DRIVER_LOAD

```
#define LIFTER_DRIVER_LOAD  MASTER,  RIGHT_BUTTONS, JOY_RIGHT
```

Height to raise lifter to driver preload height.

Definition at line **63** of file **lifter.h**.

7.17.2.7 LIFTER_I

```
#define LIFTER_I 0
```

The integral constant for the lifter PID.

Definition at line **42** of file **lifter.h**.

7.17.2.8 LIFTER_P

```
#define LIFTER_P .15
```

The proportional constant for the lifter PID.

Definition at line **27** of file **lifter.h**.

7.17.2.9 LIFTER_UP

```
#define LIFTER_UP  MASTER, 5, JOY_UP
```

The lifter up controller params.

Definition at line **53** of file **lifter.h**.

Referenced by **update_lifter()**.

7.17.2.10 LIFTER_UP_PARTNER

```
#define LIFTER_UP_PARTNER  PARTNER, 5, JOY_UP
```

The lifter up controller params for the partner.

Definition at line **68** of file **lifter.h**.

7.17.2.11 MAIN_LIFTER_MIN_HEIGHT

```
#define MAIN_LIFTER_MIN_HEIGHT 1700
```

Definition at line **83** of file **lifter.h**.

Referenced by **update_lifter()**.

7.17.2.12 MAIN_LIFTER_POT

```
#define MAIN_LIFTER_POT 1
```

Definition at line **81** of file **lifter.h**.

Referenced by **update_lifter()**.

7.17.2.13 SECONDARY_LIFTER_MAX_HEIGHT

```
#define SECONDARY_LIFTER_MAX_HEIGHT 2500
```

Definition at line **77** of file **lifter.h**.

Referenced by **update_lifter()**.

7.17.2.14 SECONDARY_LIFTER_MIN_HEIGHT

```
#define SECONDARY_LIFTER_MIN_HEIGHT 1300
```

Definition at line **79** of file **lifter.h**.

7.17.2.15 SECONDARY_LIFTER_POT_PORT

```
#define SECONDARY_LIFTER_POT_PORT 2
```

Definition at line **75** of file **lifter.h**.

Referenced by **update_lifter()**.

7.17.2.16 THRESHOLD

```
#define THRESHOLD 10
```

The threshold of a signficant speed for the lifter.

Definition at line **37** of file **lifter.h**.

7.17.3 Function Documentation

7.17.3.1 getLifterHeight()

```
double getLifterHeight ( )
```

Gets the height of the lifter in inches.

Returns

the height of the lifter.

Author

Chris Jerrett

Date

9/17/2017

Definition at line **133** of file **lifter.c**.

References **getLifterTicks()**.

```
00133                         {
00134   unsigned int ticks = getLifterTicks();
00135   return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) + 0.0198 * ticks + 2.3033);
00136 }
```

7.17.3.2 getLifterTicks()

```
int getLifterTicks ( )
```

Gets the value of the lifter pot.

Returns

the value of the pot.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **122** of file **lifter.c**.

References **LIFTER**.

Referenced by **getLifterHeight()**.

```
00122                        {
00123   return analogRead(LIFTER);
00124 }
```

7.17.3.3 lifterPotentiometerToDegree()

```
float lifterPotentiometerToDegree (
            int x )
```

height of the lifter in degrees from 0 height

Parameters

x	the pot value

Returns

the positions in degrees

Author

Chris Jerrett

Date

10/13/2017

Definition at line **111** of file **lifter.c**.

References **DEG_MAX**, **INIT_ROTATION**, and **TICK_MAX**.

```
00111                                    {
00112   return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
00113 }
```

7.17.3.4 lower_main_lifter()

```
void lower_main_lifter ( )
```

Lowers the main lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **53** of file **lifter.c**.

References **MIN_SPEED**, and **set_main_lifter_motors()**.

```
00053                         {
00054   set_main_lifter_motors(MIN_SPEED);
00055 }
```

7.17.3.5 lower_secondary_lifter()

```
void lower_secondary_lifter ( )
```

Lowers the secondary lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **73** of file **lifter.c**.

References **MIN_SPEED**, and **set_secondary_lifter_motors()**.

```
00073                              {
00074   set_secondary_lifter_motors(MIN_SPEED);
00075 }
```

7.17.3.6 raise_main_lifter()

```
void raise_main_lifter ( )
```

Raises the main lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **43** of file **lifter.c**.

References **MAX_SPEED**, and **set_main_lifter_motors()**.

```
00043                         {
00044   set_main_lifter_motors(MAX_SPEED);
00045 }
```

7.17.3.7 raise_secondary_lifter()

```
void raise_secondary_lifter ( )
```

Raises the main lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **63** of file **lifter.c**.

References **MAX_SPEED**, and **set_secondary_lifter_motors()**.

```
00063                              {
00064   set_secondary_lifter_motors(MAX_SPEED);
00065 }
```

7.17.3.8 set_lifter_pos()

```
void set_lifter_pos (
            int pos )
```

Sets the lifter positions to the given value.

Parameters

pos	The height in inches

Author

Chris Jerrett

Date

9/12/2017

Definition at line **33** of file **lifter.c**.

```
                          {
 
}
```

7.17.3.9 set_main_lifter_motors()

```
void set_main_lifter_motors (
            const int v )
```

Sets the main lifter motors to the given value.

Parameters

v	value for the lifter motor. Between -128 - 127, any values outside are clamped.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **22** of file **lifter.c**.

References **MOTOR_LIFT**, and **set_motor_immediate()**.

Referenced by **lower_main_lifter()**, **raise_main_lifter()**, and **update_lifter()**.

```
                                     {
  set_motor_immediate(MOTOR_LIFT, v);
}
```

7.17.3.10 set_secondary_lifter_motors()

```
void set_secondary_lifter_motors (
            const int v )
```

Sets the secondary lifter motors to the given value.

Parameters

v	value for the lifter motor. Between -128 - 127, any values outside are clamped.

Author

Chris Jerrett

Date

1/6/2018

Definition at line **11** of file **lifter.c**.

References **MOTOR_SECONDARY_LIFTER**, and **set_motor_immediate()**.

Referenced by **lower_secondary_lifter()**, **raise_secondary_lifter()**, and **update_lifter()**.

```
00011                                          {
00012   set_motor_immediate(MOTOR_SECONDARY_LIFTER, v);
00013 }
```

7.17.3.11 update_lifter()

```
void update_lifter ( )
```

Updates the lifter in teleop.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **83** of file **lifter.c**.

References **LIFTER_DOWN**, **LIFTER_UP**, **MAIN_LIFTER_MIN_HEIGHT**, **MAIN_LIFTER_POT**, **MAX_SPEED**, **MIN_SPEED**, **SECONDARY_LIFTER_MAX_HEIGHT**, **SECONDARY_LIFTER_POT_PORT**, **set_main_lifter_motors()**, and **set_secondary_lifter_motors()**.

Referenced by **operatorControl()**.

```
                       {
  printf("%d\n", analogRead(1));
  if(joystickGetDigital(LIFTER_DOWN) && analogRead(MAIN_LIFTER_POT) <
 MAIN_LIFTER_MIN_HEIGHT) {
    set_secondary_lifter_motors(MAX_SPEED);
    set_main_lifter_motors(MIN_SPEED);
  } else if(joystickGetDigital(LIFTER_DOWN) && analogRead(MAIN_LIFTER_POT) >=
 MAIN_LIFTER_MIN_HEIGHT) {
    set_secondary_lifter_motors(MAX_SPEED);
    set_main_lifter_motors(0);
  } else if(joystickGetDigital(LIFTER_UP) && analogRead(
 SECONDARY_LIFTER_POT_PORT) < SECONDARY_LIFTER_MAX_HEIGHT) {
    set_secondary_lifter_motors(MIN_SPEED);
    set_main_lifter_motors(0);
  } else if(joystickGetDigital(LIFTER_UP) && analogRead(
 SECONDARY_LIFTER_POT_PORT) >= SECONDARY_LIFTER_MAX_HEIGHT) {
    set_main_lifter_motors(MAX_SPEED);
    set_secondary_lifter_motors(MIN_SPEED);
  } else {
    set_secondary_lifter_motors(0);
    set_main_lifter_motors(0);
  }
}
```

7.18 lifter.h

```

#ifndef _LIFTER_H_
#define _LIFTER_H_

#include <API.h>
#include "motor_ports.h"
#include "sensor_ports.h"
#include "slew.h"
#include "controller.h"
#include "potentiometer.h"
#include "partner.h"
#include "drive.h"

#define INIT_ROTATION 680

#define LIFTER_P .15

#define LIFTER_D 0

#define THRESHOLD 10

#define LIFTER_I 0


#define HEIGHT 19.1 - 3.8

#define LIFTER_UP MASTER, 5, JOY_UP

#define LIFTER_DOWN MASTER, 5, JOY_DOWN

#define LIFTER_DRIVER_LOAD MASTER, RIGHT_BUTTONS, JOY_RIGHT

#define LIFTER_UP_PARTNER PARTNER, 5, JOY_UP

#define LIFTER_DOWN_PARTNER PARTNER, 5, JOY_DOWN

#define SECONDARY_LIFTER_POT_PORT 2
```

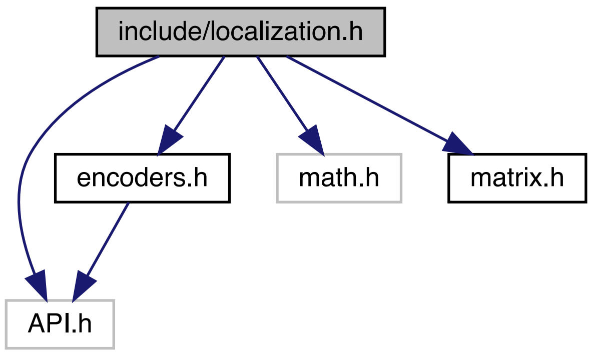
```

#define SECONDARY_LIFTER_MAX_HEIGHT 2500

#define SECONDARY_LIFTER_MIN_HEIGHT 1300

#define MAIN_LIFTER_POT 1

#define MAIN_LIFTER_MIN_HEIGHT 1700

void set_secondary_lifter_motors(const int v);

void set_main_lifter_motors(const int v);

void set_lifter_pos(int pos);

void raise_main_lifter();

void lower_main_lifter();

void raise_secondary_lifter();

void lower_secondary_lifter();

void update_lifter();

float lifterPotentiometerToDegree(int x);

int getLifterTicks();

double getLifterHeight();


#endif
```

7.19 include/localization.h File Reference

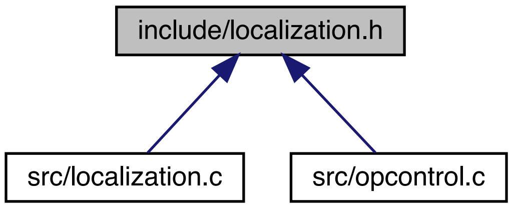
Declarations and macros for determining the location of the robot. [WIP].

```
#include <API.h>
#include "encoders.h"
#include <math.h>
#include "matrix.h"
```

Include dependency graph for localization.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **location**

Macros

- #define **LOCALIZATION_UPDATE_FREQUENCY** 0.500

Functions

- struct **location** **get_position** ()

  Gets the current posituion of the robot.
- bool **init_localization** (const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int start←
  _theta)

  Starts the localization process.
- void **update_position** ()

  Updates the position from the localization.

7.19.1 Detailed Description

Declarations and macros for determining the location of the robot. [WIP].

Author

Chris Jerrett, Christian Desimone

Date

9/27/2017

Definition in file **localization.h**.

7.19.2 Macro Definition Documentation

7.19.2.1 LOCALIZATION_UPDATE_FREQUENCY

```
#define LOCALIZATION_UPDATE_FREQUENCY 0.500
```

How often the localization code updates the position.

Definition at line **18** of file **localization.h**.

Referenced by **calculate_gryo_anglular_velocity()**, **init_localization()**, and **integrate_gyro_w()**.

7.19.3 Function Documentation

7.19.3.1 get_position()

```
struct location get_position ( )
```

Gets the current posituion of the robot.

Parameters

gyro1	The first gyro

Returns

the loacation of the robot as a struct.

Definition at line **31** of file **localization.c**.

```
00031                             {
00032 
00033 }
```

7.19.3.2 init_localization()

```
bool init_localization (
            const unsigned char gyro1,
            unsigned short multiplier,
            int start_x,
            int start_y,
            int start_theta )
```

Starts the localization process.

Author

Chris Jerrett

Parameters

gyro1	The first gyro The multiplier parameter can tune the gyro to adapt to specific sensors. The default value at this time is 196; higher values will increase the number of degrees reported for a fixed actual rotation, while lower values will decrease the number of degrees reported. If your robot is consistently turning too far, increase the multiplier, and if it is not turning far enough, decrease the multiplier.

Definition at line **100** of file **localization.c**.

References **g1**, **last_call**, **localization_task**, **LOCALIZATION_UPDATE_FREQUENCY**, **makeMatrix()**, and **update_position()**.

```
                {
  g1 = gyroInit(gyro1, multiplier);
  //init state matrix

  //one dimensional vector with x, y, theta, acceleration in x and y
  state_matrix = makeMatrix(1, 5);
  localization_task = taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
  last_call = millis();
  return true;
}
```

7.19.3.3 update_position()

```
void update_position ( )
```

Updates the position from the localization.

Author

Chris Jerrett

Definition at line **40** of file **localization.c**.

References **calculate_accelerometer_odemetry()**, and **last_call**.

Referenced by **init_localization()**.

```
                       {
  //int curr_theta = calculate_angle();

  struct accelerometer_odometry oddem = calculate_accelerometer_odemetry();
  //printf("x: %d y: %d T: %d\n", a.x, a.y, 0);

  /*int l = 1;
  int vr = get_encoder_velocity(1);
  int vl = get_encoder_velocity(2);
  int theta_dot = (vr - vl) / l;
  int curr_theta = theta + theta_dot;
  double dt = LOCALIZATION_UPDATE_FREQUENCY;
  double v_tot = (vr+vl)/2.0;
  int x_curr = x - v_tot*dt*sin(curr_theta);
  int y_curr = y + v_tot*dt*cos(curr_theta);
  x = x_curr;
  y = y_curr;*/
  last_call = millis();
}
```

7.20 localization.h

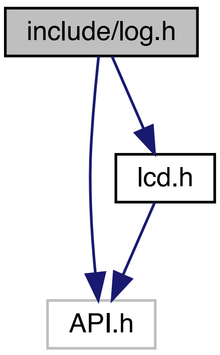
```
#ifndef _LOCALIZATION_H_
#define _LOCALIZATION_H_

#include <API.h>
#include "encoders.h"
#include <math.h>
#include "matrix.h"

#define LOCALIZATION_UPDATE_FREQUENCY 0.500

struct location {
  int x;
  int y;
  int theta;
};

bool init_localization(const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int
      start_theta);

struct location get_position();

void update_position();

#endif
```

7.21 include/log.h File Reference

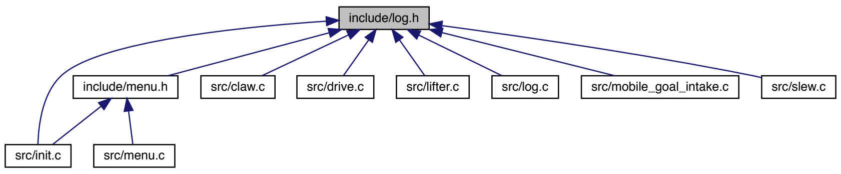
Contains logging functions.

```
#include <API.h>
#include "lcd.h"
```

Include dependency graph for log.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **DEBUG** 4

  logging only info debug. most verbose level
- #define **ERROR** 1

  logging only errors. Also displays error to lcd
- #define **INFO** 3

  logging only info messages and higher.
- #define **NONE** 0

  No logging. Should be used in competition to reduce serial communication.
- #define **WARNING** 2

  logs errors and warnings. Also displays error to lcd

Functions

- void **debug** (const char ∗debug_message)

  prints a info message
- void **error** (const char ∗error_message)

  prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE
- void **info** (const char ∗info_message)

  prints a info message
- void **init_error** (bool use_lcd, FILE ∗lcd)

  Initializes the error lcd system Only required if using lcd.
- void **warning** (const char ∗warning_message)

  prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

7.21.1 Detailed Description

Contains logging functions.

Author

Chris Jerrett

Date

9/16/2017

Definition in file **log.h**.

7.21.2 Macro Definition Documentation

7.21.2.1 DEBUG

```
#define DEBUG 4
```

logging only info debug. most verbose level

Author

Chris Jerrett

Date

9/10/17

Definition at line **50** of file **log.h**.

7.21.2.2 ERROR

```
#define ERROR 1
```

logging only errors. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line **27** of file **log.h**.

Referenced by **debug()**, and **info()**.

7.21.2.3 INFO

```
#define INFO 3
```

logging only info messages and higher.

Author

Chris Jerrett

Date

9/10/17

Definition at line **42** of file **log.h**.

7.21.2.4 NONE

```
#define NONE 0
```

No logging. Should be used in competition to reduce serial communication.

Author

Chris Jerrett

Date

9/10/17

Definition at line **19** of file **log.h**.

Referenced by **error()**.

7.21.2.5 WARNING

```
#define WARNING 2
```

logs errors and warnings. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line **35** of file **log.h**.

Referenced by **warning()**.

7.21.3 Function Documentation

7.21.3.1 debug()

```
void debug (
            const char * debug_message )
```

prints a info message

Only will print and display if log_level is greater than info

See also

log_level (p. 182)

Parameters

debug_message	the message

Definition at line **77** of file **log.c**.

References **ERROR**, and **log_level**.

Referenced by **set_motor_immediate()**, and **set_motor_slew()**.

```
00077                                       {
00078   if(log_level>ERROR) {
00079     printf("[INFO]: %s\n", debug_message);
00080   }
00081 }
```

7.21.3.2 error()

```
void error (
            const char * error_message )
```

prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

log_level (p. 182)

Author

Chris Jerrett

Date

9/10/17

Parameters

error_message	the message

Definition at line **39** of file **log.c**.

References **log_info()**, **log_level**, and **NONE**.

Referenced by **create_menu()**.

```
                                          {
   if(log_level>NONE)
     log_info("ERROR", error_message);
 }
```

7.21.3.3 info()

```
void info (
              const char * info_message )
```

prints a info message

Only will print and display if log_level is greater than ERROR

See also

log_level (p. 182)

Parameters

info_message	the message

Definition at line **64** of file **log.c**.

References **ERROR**, **log_info()**, and **log_level**.

Referenced by **init_slew()**, and **initialize()**.

```
                                        {
   if(log_level>ERROR) {
     log_info("INFO", info_message);
   }
 }
```

7.21.3.4 init_error()

```
void init_error (
            bool use_lcd,
            FILE * lcd )
```

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

use_lcd	whether to use the lcd
lcd	the lcd

Definition at line **14** of file **log.c**.

References **log_lcd**.

Referenced by **initialize()**.

```
                                    {
  if(use_lcd) {
    lcdInit(lcd);
    log_lcd = lcd;
    lcdClear(log_lcd);
    printf("LCD Init\n");
  }
}
```

7.21.3.5 warning()

```
void warning (
            const char * warning_message )
```

prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

log_level (p. 182)

Author

Chris Jerrett

Date

9/10/17

Parameters

warning_message	the message

Definition at line **52** of file **log.c**.

References **log_info()**, **log_level**, and **WARNING**.

Referenced by **init_slew()**.

```
                                           {
  if(log_level>WARNING)
    log_info("WARNING", warning_message);
}
```

7.22 log.h

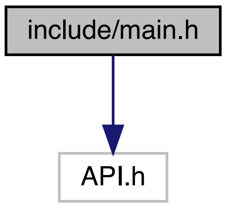
```

#ifndef _LOG_H_
#define _LOG_H_

#include <API.h>
#include "lcd.h"

#define NONE 0

#define ERROR 1

#define WARNING 2

#define INFO 3

#define DEBUG 4

void init_error(bool use_lcd, FILE *lcd);

void error(const char *error_message);

void warning(const char *warning_message);

void info(const char *info_message);

void debug(const char *debug_message);


#endif
```

7.23 include/main.h File Reference

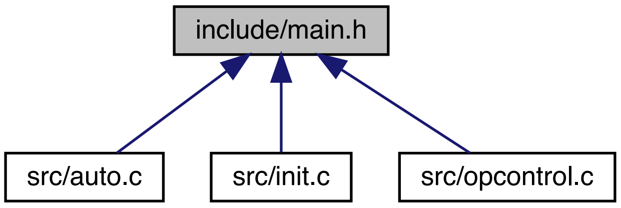
Header file for global functions.

```
#include <API.h>
```

Include dependency graph for main.h:



This graph shows which files directly or indirectly include this file:



Functions

- void **autonomous** ()
- void **initialize** ()
- void **initializeIO** ()
- void **operatorControl** ()

7.23.1 Detailed Description

Header file for global functions.

Any experienced C or C++ programmer knows the importance of header files. For those who do not, a header file allows multiple files to reference functions in other files without necessarily having to see the code (and therefore causing a multiple definition). To make a function in "opcontrol.c", "auto.c", "main.c", or any other C file visible to the core implementation files, prototype it here.

This file is included by default in the predefined stubs in each VEX Cortex PROS Project.

Purdue Robotics OS contains FreeRTOS (`http://www.freertos.org`) whose source code may be obtained from `http://sourceforge.net/projects/freertos/files/` or on request.

Definition in file **main.h**.

7.23.2 Function Documentation

7.23.2.1 autonomous()

```
void autonomous ( )
```

Runs the user autonomous code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is lost, the autonomous task will be stopped by the kernel. Re-enabling the robot will restart the task, not re-start it from where it left off.

Code running in the autonomous task cannot access information from the VEX Joystick. However, the autonomous function can be invoked from another task if a VEX Competition Switch is not available, and it can access joystick information if called in this way.

The autonomous task may exit, unlike **operatorControl()** (p. 84) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

Definition at line **30** of file **auto.c**.

References **BOTH**, **close_claw()**, **deinitslew()**, **GOAL_HEIGHT**, **init_slew()**, **LIFTER**, **MID_LEFT_DRIVE**, **MID_RIGHT_DRIVE**, **open_claw()**, and **set_side_speed()**.

```
                  {
  init_slew();

  delay(10);
  printf("auto\n");
  //How far the left wheels have gone
  int counts_drive_left;
  //How far the right wheels have gone
  int counts_drive_right;
  //The average distance traveled forward
  int counts_drive;

  //Reset the integrated motor controllers
  imeReset(MID_LEFT_DRIVE);
  imeReset(MID_RIGHT_DRIVE);
  //Set initial values for how far the wheels have gone
  imeGet(MID_LEFT_DRIVE, &counts_drive_left);
  imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
  counts_drive = counts_drive_left + counts_drive_right;
  counts_drive /= 2;

  //Grab pre-load cone
  close_claw();
  delay(300);

  //Raise the lifter
  while(analogRead(LIFTER) < GOAL_HEIGHT){
    //set_lifter_motors(-127);
  }
  //set_lifter_motors(0);
  //Drive towards the goal
  while(counts_drive < 530){
    set_side_speed(BOTH, 127);
    //Restablish the distance traveled
    imeGet(MID_LEFT_DRIVE, &counts_drive_left);
    imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
    counts_drive = counts_drive_left + counts_drive_right;
    counts_drive /= 2;
  }
  //Stop moving
  set_side_speed(BOTH, 0);
  delay(1000);

  //Drop the cone on the goal
  open_claw();
  delay(1000);
  deinitslew();
}
```

7.23.2.2 initialize()

```
void initialize ( )
```

Runs user initialization code. This function will be started in its own task with the default priority and stack size once when the robot is starting up. It is possible that the VEXnet communication link may not be fully established at this time, so reading from the VEX Joystick may fail.

This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global variables, and IMEs.

This function must exit relatively promptly, or the **operatorControl()** (p. 84) and **autonomous()** (p. 83) tasks will not start. An autonomous mode selection menu like the pre_auton() in other environments can be implemented in this task if desired.

Definition at line **47** of file **init.c**.

References **display_menu()**, **info()**, **init_error()**, **init_main_lcd()**, **init_menu_var()**, and **STRING_TYPE**.

```
                    {
  init_main_lcd(uart1);
  info("LCD Init");
  menu_t *t = init_menu_var(STRING_TYPE, "TEST Menu", 5, "1","2","3","4","5");
  init_error(true, uart2);
  display_menu(t);
  setTeamName("9228A");
}
```

7.23.2.3 initializeIO()

```
void initializeIO ( )
```

Runs pre-initialization code. This function will be started in kernel mode one time while the VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.

The purpose of this function is solely to set the default pin modes (pinMode()) and port states (digitalWrite()) of limit switches, push buttons, and solenoids. It can also safely configure a UART port (usartOpen()) but cannot set up an LCD (lcdInit()).

Definition at line **30** of file **init.c**.

```
                      {
    watchdogInit();
}
```

7.23.2.4 operatorControl()

```
void operatorControl ( )
```

Runs the user operator control code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the operator control mode. If the robot is disabled or communications is lost, the operator control task will be stopped by the kernel. Re-enabling the robot will restart the task, not resume it from where it left off.

If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will run the operator control task. Be warned that this will also occur if the VEX Cortex is tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.

Code running in this task can take almost any action, as the VEX Joystick is available and the scheduler is operational. However, proper use of delay() or taskDelayUntil() is highly recommended to give other tasks (including system tasks such as updating LCDs) time to run.

This task should never exit; it should end with some kind of infinite loop, even if empty.

Runs the user operator control code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the operator control mode. If the robot is disabled or communications is lost, the operator control task will be stopped by the kernel. Re-enabling the robot will restart the task, not resume it from where it left off.

If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will run the operator control task. Be warned that this will also occur if the VEX Cortex is tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.

Code running in this task can take almost any action, as the VEX Joystick is available and the scheduler is operational. However, proper use of delay() or taskDelayUntil() is highly recommended to give other tasks (including system tasks such as updating LCDs) time to run.

This task should never exit; its should end with some kind of infinite loop, even if empty.

Definition at line **42** of file **opcontrol.c**.

References **init_slew()**, **update_claw()**, **update_drive_motors()**, **update_intake()**, and **update_lifter()**.

```
00042                        {
00043 
00044     init_slew();
00045     delay(10);
00046     while (1) {
00047         update_claw();
00048         delay(5);
00049         update_intake();
00050         delay(5);
00051         update_lifter();
00052         delay(5);
00053         update_drive_motors();
00054         delay(25);
00055 
00056     }
00057 }
```

7.24 main.h

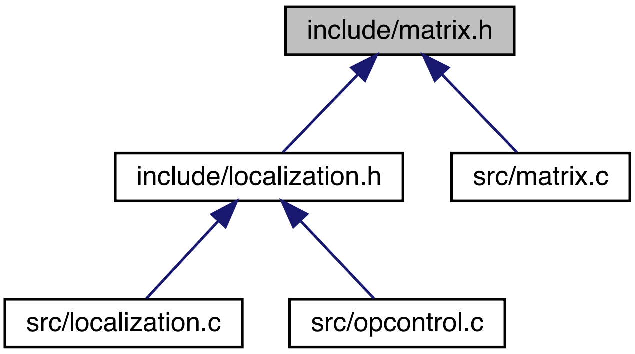
```

#ifndef MAIN_H_

// This prevents multiple inclusion, which isn't bad for this file but is good practice
#define MAIN_H_

#include <API.h>

// Allow usage of this file in C++ programs
#ifdef __cplusplus
extern "C" {
#endif

//#define AUTO_DEBUG

// A function prototype looks exactly like its declaration, but with a semicolon instead of
// actual code. If a function does not match a prototype, compile errors will occur.

// Prototypes for initialization, operator control and autonomous

void autonomous();
void initializeIO();
void initialize();
void operatorControl();

// End C++ export structure
#ifdef __cplusplus
}
#endif

#endif
```

7.25 include/matrix.h File Reference

Various Matrix operations.

None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevelant, be sure to use the function to reclaim some of that memory.

This graph shows which files directly or indirectly include this file:



Data Structures

- struct **_matrix**

Typedefs

- typedef struct **_matrix** **matrix**

Functions

- void **assert** (int assertion, const char ∗message)

  Asserts a condition is true.
- **matrix** ∗ **copyMatrix** (**matrix** ∗m)

  Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.
- **matrix** ∗ **covarianceMatrix** (**matrix** ∗m)

  returns the covariance of the matrix
- **matrix** ∗ **dotDiagonalMatrix** (**matrix** ∗a, **matrix** ∗b)

  performs a diagonial matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a 1 by a->height matrix of the cross product of each matrix along the diagonal.
- **matrix** ∗ **dotProductMatrix** (**matrix** ∗a, **matrix** ∗b)

  returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a a->height by b->height matrix of the cross product of each matrix.
- void **freeMatrix** (**matrix** ∗m)

  Frees the resources of a matrix.
- **matrix** ∗ **identityMatrix** (int n)

  Returns an identity matrix of size n by n.
- **matrix** ∗ **makeMatrix** (int width, int height)

  Makes a matrix with a width and height parameters.
- **matrix** ∗ **meanMatrix** (**matrix** ∗m)

  Given an "m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.
- **matrix** ∗ **multiplyMatrix** (**matrix** ∗a, **matrix** ∗b)

  Given a two matrices, returns the multiplication of the two.
- void **printMatrix** (**matrix** ∗m)

  Prints a matrix.
- void **rowSwap** (**matrix** ∗a, int p, int q)

  swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q, provided that p and q are less than or equal to the height of matrix A and p and q are different values.
- **matrix** ∗ **scaleMatrix** (**matrix** ∗m, double value)

  scales a matrix.
- double **traceMatrix** (**matrix** ∗m)

  Given an "m rows by n columns" matrix.
- **matrix** ∗ **transposeMatrix** (**matrix** ∗m)

  returns the transpose matrix.

7.25.1 Detailed Description

Various Matrix operations.

None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevelant, be sure to use the function to reclaim some of that memory.

Definition in file **matrix.h**.

7.25.2 Typedef Documentation

7.25.2.1 matrix

```
typedef struct _matrix matrix
```

A struct representing a matrix

7.25.3 Function Documentation

7.25.3.1 assert()

```
void assert (
            int assertion,
            const char * message )
```

Asserts a condition is true.

If the assertion is non-zero (i.e. true), then it returns. If the assertion is zero (i.e. false), then it display the string and aborts the program. This is ment to act like Python's assert keyword.

Definition at line **14** of file **matrix.c**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, and **rowSwap()**.

```
                                                  {
    if (assertion == 0) {
        fprintf(stderr, "%s\n", message);
        exit(1);
    }
}
```

7.25.3.2 copyMatrix()

```
matrix* copyMatrix (
            matrix * m )
```

Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.

Parameters

m	a pointer to the matrix

Returns

a copied matrix

Definition at line **52** of file **matrix.c**.

References **scaleMatrix()**.

```
                         {
    return scaleMatrix(m, 1);
}
```

7.25.3.3 covarianceMatrix()

```
matrix* covarianceMatrix (
              matrix * m )
```

returns the covariance of the matrix

Parameters

the	matrix

Returns

a matrix with n row and n columns, where each element represents covariance of 2 columns.

Definition at line **168** of file **matrix.c**.

References **assert()**, **_matrix::data**, **freeMatrix()**, **_matrix::height**, **makeMatrix()**, **meanMatrix()**, and **_↩ matrix::width**.

```
                                  {
    int i, j, k = 0;
    matrix* out;
    matrix* mean;
    double* ptrA;
    double* ptrB;
    double* ptrOut;

    assert(m->height > 1, "Height of matrix cannot be zero or one.");

    mean = meanMatrix(m);
    out = makeMatrix(m->width, m->width);
    ptrOut = out->data;

    for (i = 0; i < m->width; i++) {
        for (j = 0; j < m->width; j++) {
            ptrA = &m->data[i];
            ptrB = &m->data[j];
            *ptrOut = 0.0;
            for (k = 0; k < m->height; k++) {
                *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
                ptrA += m->width;
                ptrB += m->width;
            }
            *ptrOut /= m->height - 1;
            ptrOut++;
        }
    }

    freeMatrix(mean);
    return out;
}
```

7.25.3.4 dotDiagonalMatrix()

```
matrix* dotDiagonalMatrix (
            matrix * a,
            matrix * b )
```

performs a diagonial matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a 1 by a->height matrix of the cross product of each matrix along the diagonal.

Dot product is essentially the sum-of-squares of two vectors.

If the second paramter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

a	the first matrix
b	the second matrix

Returns

the matrix result

Definition at line **385** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                                {
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;
    int i, j;

    if (b != NULL) {
        assert(a->width == b->width && a->height == b->height, "Matrices must be of the same
  dimensionality.");
    }

    // Are we computing the sum of squares of the same matrix?
    if (a == b || b == NULL) {
        b = a; // May not appear safe, but we can do this without risk of losing b.
    }

    out = makeMatrix(1, a->height);
    ptrOut = out->data;
    ptrA = a->data;
    ptrB = b->data;

    for (i = 0; i < a->height; i++) {
        *ptrOut = 0;
        for (j = 0; j < a->width; j++) {
            *ptrOut += *ptrA * *ptrB;
            ptrA++;
            ptrB++;
        }
        ptrOut++;
    }

    return out;
}
```

7.25.3.5 dotProductMatrix()

```
matrix* dotProductMatrix (
            matrix * a,
            matrix * b )
```

returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a a->height by b->height matrix of the cross product of each matrix.

Dot product is essentially the sum-of-squares of two vectors.

Also, if the second paramter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

a	the first matrix
the	second matrix

Returns

the result of the dot product

Definition at line **333** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                          {
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;
    int i, j, k;

    if (b != NULL) {
        assert(a->width == b->width, "Matrices must be of the same dimensionality.");
    }

    // Are we computing the sum of squares of the same matrix?
    if (a == b || b == NULL) {
        b = a; // May not appear safe, but we can do this without risk of losing b.
    }

    out = makeMatrix(b->height, a->height);
    ptrOut = out->data;

    for (i = 0; i < a->height; i++) {
        ptrB = b->data;

        for (j = 0; j < b->height; j++) {
            ptrA = &a->data[ i * a->width ];

            *ptrOut = 0;
            for (k = 0; k < a->width; k++) {
                *ptrOut += *ptrA * *ptrB;
                ptrA++;
                ptrB++;
            }
            ptrOut++;
        }
    }

    return out;
}
```

7.25.3.6 freeMatrix()

```
void freeMatrix (
              matrix * m )
```

Frees the resources of a matrix.

Parameters

the	matrix to free

Definition at line **60** of file **matrix.c**.

References **_matrix::data**.

Referenced by **covarianceMatrix()**.

```
                             {
    if (m != NULL) {
        if (m->data != NULL) {
            free(m->data);
            m->data = NULL;
        }
        free(m);
    }
    return;
}
```

7.25.3.7 identityMatrix()

```
 matrix* identityMatrix (
             int n )
```

Returns an identity matrix of size n by n.

Parameters

n	the input matrix. parameter.
n	the input matrix.

Returns

the identity matrix parameter.

Definition at line **94** of file **matrix.c**.

References **assert()**, **_matrix::data**, and **makeMatrix()**.

```
                                {
    int i;
    matrix *out;
    double* ptr;

    assert(n > 0, "Identity matrix must have value greater than zero.");

    out = makeMatrix(n, n);
    ptr = out->data;
    for (i = 0; i < n; i++) {
        *ptr = 1.0;
        ptr += n + 1;
    }

    return out;
}
```

7.25.3.8 makeMatrix()

```
matrix* makeMatrix (
              int width,
              int height )
```

Makes a matrix with a width and height parameters.

Parameters

width	The width of the matrix
height	the height of the matrix

Returns

the new matrix

Definition at line **27** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, and **_matrix::width**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **init_↩ localization()**, **meanMatrix()**, **multiplyMatrix()**, **scaleMatrix()**, and **transposeMatrix()**.

```
                                           {
    matrix* out;
    assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
    out = (matrix*) malloc(sizeof(matrix));

    assert(out != NULL, "Out of memory.");

    out->width = width;
    out->height = height;
    out->data = (double*) malloc(sizeof(double) * width * height);

    assert(out->data != NULL, "Out of memory.");

    memset(out->data, 0.0, width * height * sizeof(double));

    return out;
}
```

7.25.3.9 meanMatrix()

```
matrix* meanMatrix (
            matrix * m )
```

Given an "m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.

Returns

matrix with 1 row and n columns each element represents the mean of that full column.

Definition at line **142** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **covarianceMatrix()**.

```
                                {
    int i, j;
    matrix* out;

    assert(m->height > 0, "Height of matrix cannot be zero.");

    out = makeMatrix(m->width, 1);

    for (i = 0; i < m->width; i++) {
        double* ptr;
        out->data[i] = 0.0;
        ptr = &m->data[i];
        for (j = 0; j < m->height; j++) {
            out->data[i] += *ptr;
            ptr += out->width;
        }
        out->data[i] /= (double) m->height;
    }
    return out;
}
```

7.25.3.10 multiplyMatrix()

```
matrix* multiplyMatrix (
            matrix * a,
            matrix * b )
```

Given a two matrices, returns the multiplication of the two.

Parameters

a	the first matrix
b	the seconf matrix return the result of the multiplication

Definition at line **230** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                               {
    int i, j, k;
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;

    assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");

    out = makeMatrix(b->width, a->height);
    ptrOut = out->data;

    for (i = 0; i < a->height; i++) {

        for (j = 0; j < b->width; j++) {
            ptrA = &a->data[ i * a->width ];
            ptrB = &b->data[ j ];

            *ptrOut = 0;
            for (k = 0; k < a->width; k++) {
                *ptrOut += *ptrA * *ptrB;
                ptrA++;
                ptrB += b->width;
            }
            ptrOut++;
        }
    }

    return out;
}
```

7.25.3.11 printMatrix()

```
void printMatrix (
               matrix * m )
```

Prints a matrix.

Parameters

the	matrix

Definition at line **75** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
                                {
    int i, j;
    double* ptr = m->data;
    printf("%d %d\n", m->width, m->height);
    for (i = 0; i < m->height; i++) {
        for (j = 0; j < m->width; j++) {
            printf(" %9.6f", *(ptr++));
        }
        printf("\n");
    }
    return;
}
```

7.25.3.12 rowSwap()

```
void rowSwap (
              matrix * a,
              int p,
              int q )
```

swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q, provided that p and q are less than or equal to the height of matrix A and p and q are different values.

Parameters

the	matrix to swap. This method changes the input matrix.
the	first row
the	second row

Definition at line **290** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
                                     {
    int i;
    double temp;
    double* pRow;
    double* qRow;

    assert(a->height > 2, "Matrix must have at least two rows to swap.");
    assert(p < a->height && q < a->height, "Values p and q must be less than the height of the matrix.");

    // If p and q are equal, do nothing.
    if (p == q) {
        return;
    }

    pRow = a->data + (p * a->width);
    qRow = a->data + (q * a->width);

    // Swap!
    for (i = 0; i < a->width; i++) {
        temp = *pRow;
        *pRow = *qRow;
        *qRow = temp;
        pRow++;
        qRow++;
    }

    return;
}
```

7.25.3.13 scaleMatrix()

```
matrix* scaleMatrix (
              matrix * m,
              double value )
```

scales a matrix.

Parameters

m	the matrix to scale
the	value to scale by

Returns

a new matrix where each element in the input matrix is multiplied by the scalar value

Definition at line **268** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **copyMatrix()**.

```
00268                                              {
00269     int i, elements = m->width * m->height;
00270     matrix* out = makeMatrix(m->width, m->height);
00271     double* ptrM = m->data;
00272     double* ptrOut = out->data;
00273
00274     for (i = 0; i < elements; i++) {
00275         *(ptrOut++) = *(ptrM++) * value;
00276     }
00277
00278     return out;
00279 }
```

7.25.3.14 traceMatrix()

```
double traceMatrix (
                matrix * m )
```

Given an "m rows by n columns" matrix.

Returns

the sum of the elements along the diagonal.

Given an "m rows by n columns" matrix.

Returns

the sum of the elements along the diagonal.

Definition at line **116** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
00116                               {
00117     int i;
00118     int size;
00119     double* ptr = m->data;
00120     double sum = 0.0;
00121
00122     if (m->height < m->width) {
00123         size = m->height;
00124     }
00125     else {
00126         size = m->width;
00127     }
00128
00129     for (i = 0; i < size; i++) {
00130         sum += *ptr;
00131         ptr += m->width + 1;
00132     }
00133
00134     return sum;
00135 }
```

7.25.3.15 transposeMatrix()

```
 matrix* transposeMatrix (
                matrix * m )
```

returns the transpose matrix.

Parameters

the	matrix to transpose.

Returns

the transposed matrix.

Definition at line **206** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                   {
    matrix* out = makeMatrix(m->height, m->width);
    double* ptrM = m->data;
    int i, j;

    for (i = 0; i < m->height; i++) {
        double* ptrOut;
        ptrOut = &out->data[i];
        for (j = 0; j < m->width; j++) {
            *ptrOut = *ptrM;
            ptrM++;
            ptrOut += out->width;
        }
    }

    return out;
}
```

7.26 matrix.h

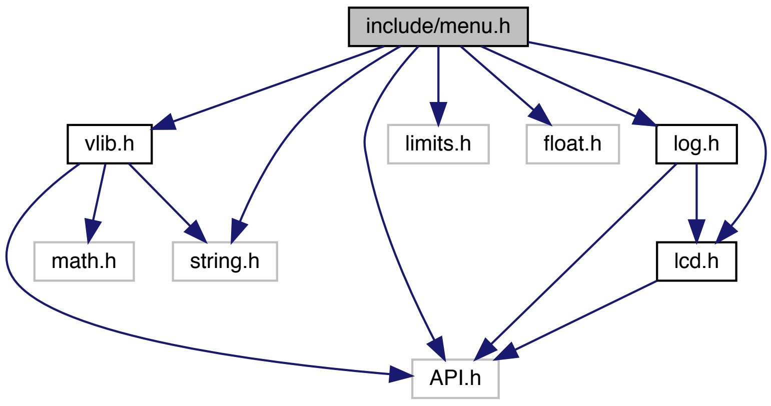
```

#ifndef _MATRIX_H_
#define _MATRIX_H_

typedef struct _matrix {
    int height;
    int width;
    double* data;
} matrix;

void assert(int assertion, const char* message);

matrix* makeMatrix(int width, int height);

matrix* copyMatrix(matrix* m);

void freeMatrix(matrix* m);

void printMatrix(matrix* m);

//============================
// Basic Matrix operations
//============================
matrix* identityMatrix(int n);

double traceMatrix(matrix* m);

matrix* transposeMatrix(matrix* m);

matrix* meanMatrix(matrix* m);

matrix* multiplyMatrix(matrix* a, matrix* b);

matrix* scaleMatrix(matrix* m, double value);

matrix* covarianceMatrix(matrix* m);

void rowSwap(matrix* a, int p, int q);
matrix* dotProductMatrix(matrix* a, matrix* b);

matrix* dotDiagonalMatrix(matrix* a, matrix* b);

#endif
```

7.27 include/menu.h File Reference

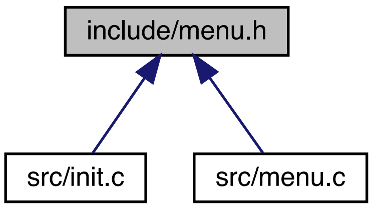
Contains menu functionality and abstraction.

```
#include "lcd.h"
#include "API.h"
#include <string.h>
#include <limits.h>
#include <float.h>
#include <vlib.h>
#include "log.h"
```

Include dependency graph for menu.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **menu_t**

  Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

Typedefs

- typedef struct **menu_t** **menu_t**

  Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

Enumerations

- enum **menu_type** { **INT_TYPE**, **FLOAT_TYPE**, **STRING_TYPE** }

  Represents the different types of menus.

Functions

- static void **calculate_current_display** (char ∗rtn, **menu_t** ∗menu)

  Static function that calculates the string from menu.
- static **menu_t** ∗ **create_menu** (enum **menu_type** type, const char ∗prompt)

  Static function that handles creation of menu. Menu must be freed or will cause memory leak
- void **denint_menu** (**menu_t** ∗menu)

  Destroys a menu Menu must be freed or will cause memory leak
- int **display_menu** (**menu_t** ∗menu)

  Displays a menu contex. Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.
- **menu_t** ∗ **init_menu_float** (enum **menu_type** type, float **min**, float **max**, float step, const char ∗prompt)

  Creates a menu context, but does not display. Menu must be freed or will cause memory leak!
- **menu_t** ∗ **init_menu_int** (enum **menu_type** type, int **min**, int **max**, int step, const char ∗prompt)

  Creates a menu context, but does not display. Menu must be freed or will cause memory leak
- **menu_t** ∗ **init_menu_var** (enum **menu_type** type, const char ∗prompt, int nums,...)

  Creates a menu context, but does not display. Menu must be freed or will cause memory leak

7.27.1 Detailed Description

Contains menu functionality and abstraction.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **menu.h**.

7.27.2 Typedef Documentation

7.27.2.1 menu_t

```
typedef struct menu_t menu_t
```

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint_menu.

Author

Chris Jerrett

Date

9/8/17

See also

menu.h (p. 99)
menu_t (p. 17)
create_menu (p. 200)
init_menu
display_menu (p. 202)
menu_type (p. 100)
denint_menu (p. 201)

7.27.3 Enumeration Type Documentation

7.27.3.1 menu_type

```
enum menu_type
```

Represents the different types of menus.

Author

Chris Jerrett

Date

9/8/17

See also

menu.h (p. 99)
menu_t (p. 17)
create_menu (p. 102)
init_menu
display_menu (p. 103)
menu_type (p. 100)

Enumerator

INT_TYPE	Menu type allowing user to select a integer. The integer type menu has a max, min and a step value. Each step is calculated. Will return the index of the selected value. Example: User goes forwards twice then it will return 2.
FLOAT_TYPE	Menu type allowing user to select a float The float type menu has a max, min and a step value. Each step is calculated. Will return the index of the selected value. Example: User goes forwards twice then it will return 2.
STRING_TYPE	Menu type allowing user to select a string from a array of strings. Will return the index of the selected value. Example: User goes forwards twice then it will return 2.

Definition at line **31** of file **menu.h**.

```
00031               {
00038   INT_TYPE,
00045   FLOAT_TYPE,
00051   STRING_TYPE
00052 };
```

7.27.4 Function Documentation

7.27.4.1 calculate_current_display()

```
static void calculate_current_display (
             char * rtn,
              menu_t * menu )  [static]
```

Static function that calculates the string from menu.

Parameters

rtn	the string to be written to
menu	the menu for prompt to be calculated from

Author

Chris Jerrett

Date

9/8/17

7.27.4.2 create_menu()

```
static  menu_t* create_menu (
             enum  menu_type type,
             const char * prompt )  [static]
```

Static function that handles creation of menu. *Menu must be freed or will cause memory leak*

Author

Chris Jerrett

Date

9/8/17

7.27.4.3 denint_menu()

```
void denint_menu (
              menu_t * menu )
```

Destroys a menu *Menu must be freed or will cause memory leak*

Parameters

menu	the menu to free

See also

menu

Author

Chris Jerrett

Date

9/8/17

Definition at line **186** of file **menu.c**.

References **menu_t::options**, and **menu_t::prompt**.

```
00186                              {
00187   free(menu->prompt);
00188   if(menu->options != NULL) free(menu->options);
00189   free(menu);
00190 }
```

7.27.4.4 display_menu()

```
int display_menu (
              menu_t * menu )
```

Displays a menu contex. *Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.*

Parameters

menu	the menu to display

See also

menu_type (p. 100)

Author

Chris Jerrett

Date

9/8/17

Definition at line **148** of file **menu.c**.

References **calculate_current_display()**, **menu_t::current**, **lcd_clear()**, **lcd_get_pressed_buttons()**, **lcd_↩ print()**, **PRESSED**, **menu_t::prompt**, **RELEASED**, and **TOP_ROW**.

Referenced by **initialize()**.

```
                              {
  lcd_print(TOP_ROW, menu->prompt);
  printf("printed prompt\n");
  //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
  char val[16];
  while(lcd_get_pressed_buttons().middle == RELEASED) {
    calculate_current_display(val, menu);

    if(lcd_get_pressed_buttons().right == PRESSED) {
      menu->current += 1;
    }
    if(lcd_get_pressed_buttons().left == PRESSED) {
      menu->current -= 1;
    }
    printf("%s\n", val);
    printf("%d\n",menu->current);
    lcd_print(2, val);
    delay(300);
  }
  printf("%d\n", menu->current);
  printf("return\n");
  lcd_clear();
  lcd_print(1, "Thk Cm Agn");
  lcd_print(2, val);
  delay(800);
  lcd_clear();
  return menu->current;
}
```

7.27.4.5 init_menu_float()

```
menu_t* init_menu_float (
            enum menu_type type,
            float min,
            float max,
            float step,
            const char * prompt )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak!*

Parameters

type	the type of menu

See also

menu_type (p. 100)

Parameters

min	the minimum value
max	the maximum value
step	the step value
prompt	the prompt to display to user

Author

Chris Jerrett

Date

9/8/17

Definition at line **95** of file **menu.c**.

References **create_menu()**, **max()**, **menu_t::max_f**, **min()**, **menu_t::min_f**, and **menu_t::step_f**.

```
{
  menu_t* menu = create_menu(type, prompt);
  menu->min_f = min;
  menu->max_f = max;
  menu->step_f = step;
  return menu;
}
```

7.27.4.6 init_menu_int()

```
menu_t* init_menu_int (
            enum  menu_type type,
            int min,
            int max,
            int step,
            const char * prompt )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

type	the type of menu

See also

menu_type (p. 100)

Parameters

min	the minimum value
max	the maximum value
step	the step value
prompt	the prompt to display to user

Author

Chris Jerrett

Date

9/8/17

Definition at line **73** of file **menu.c**.

References **create_menu()**, **menu_t::current**, **max()**, **menu_t::max**, **min()**, **menu_t::min**, and **menu_t::step**.

```
                                                                                                {
  menu_t* menu = create_menu(type, prompt);
  menu->min = min;
  menu->max = max;
  menu->step = step;
  menu->current = 0;
  return menu;
}
```

7.27.4.7 init_menu_var()

```
menu_t* init_menu_var (
            enum  menu_type type,
            const char * prompt,
            int nums,
             ...  )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

type	the type of menu

See also

menu_type (p. 100)

Parameters

nums	the number of elements passed to function
prompt	the prompt to display to user
options	the options to display for user

Author

Chris Jerrett

Date

9/8/17

Definition at line **45** of file **menu.c**.

References **create_menu()**, **menu_t::length**, and **menu_t::options**.

Referenced by **initialize()**.

```
00045                                                                       {
00046   menu_t* menu = create_menu(type, prompt);
00047   va_list ap;
00048   char **options_array = (char**)calloc(sizeof(char*), nums);
00049   va_start(ap, nums);
00050   for(int i = 0; i < nums; i++){
00051     options_array[i] = (char*) va_arg(ap, char*);
00052     printf("%s\n", options_array[i]);
00053   }
00054   va_end(ap);
00055   menu->options = options_array;
00056   menu->length = nums;
00057   return menu;
00058 }
```

7.28 menu.h

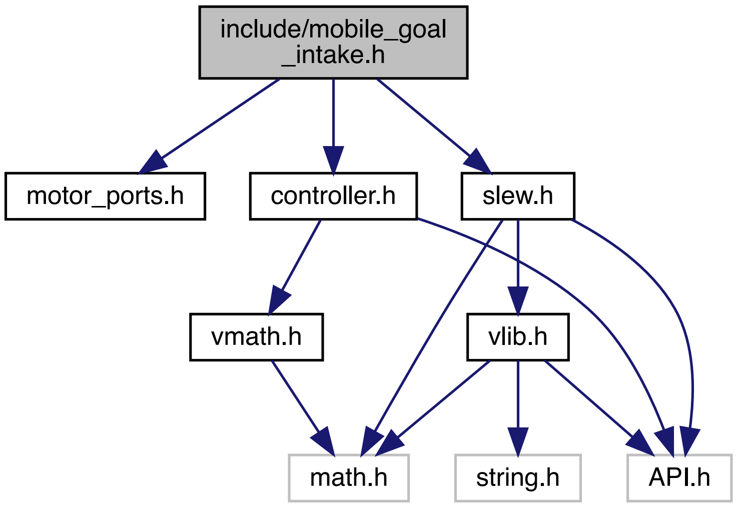
```
00001
00008 #ifndef _MENU_H_
00009 #define _MENU_H_
00010
00011 #include "lcd.h"
00012 #include "API.h"
00013 #include <string.h>
00014 #include <limits.h>
00015 #include <float.h>
00016 #include <vlib.h>
00017 //#include <stdlib.h>
00018 #include "log.h"
00019
00031 enum menu_type {
00038   INT_TYPE,
00045   FLOAT_TYPE,
00051   STRING_TYPE
00052 };
00053
00067 typedef struct menu_t{
00073   enum menu_type type;
00074
00080   char **options;
00081
00087   unsigned int length;
00088
00095   int min;
00096
00103   int max;
00104
```

```
  int step;

  float min_f;

  float max_f;

  float step_f;
  int current;
  char *prompt;
} menu_t;

static menu_t* create_menu(enum menu_type type, const char *prompt);

menu_t* init_menu_var(enum menu_type type, const char *prompt, int nums,...);

menu_t* init_menu_int(enum menu_type type, int min, int max, int step, const char*
prompt);

menu_t* init_menu_float(enum menu_type type, float min, float max, float step, const char*
prompt);

static void calculate_current_display(char* rtn, menu_t *menu);

int display_menu(menu_t *menu);

void denint_menu(menu_t *menu);

#endif
```

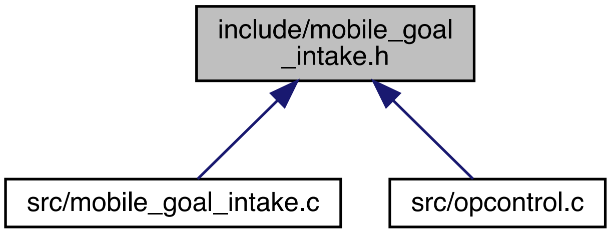
7.29 include/mobile_goal_intake.h File Reference

```
#include "motor_ports.h"
#include "controller.h"
#include "slew.h"
```

Include dependency graph for mobile_goal_intake.h:



This graph shows which files directly or indirectly include this file:



Functions

- void **update_intake** ()

  updates the mobile goal intake in teleop.

7.29.1 Function Documentation

7.29.1.1 update_intake()

```
void update_intake ( )
```

updates the mobile goal intake in teleop.

Author

Chris Jerrett

Definition at line **20** of file **mobile_goal_intake.c**.

References **lower_intake()**, **MASTER**, **raise_intake()**, and **set_intake_motor()**.

Referenced by **operatorControl()**.

```
                      {
  if(joystickGetDigital(MASTER, 7, JOY_UP)) {
    raise_intake();
  }
  else if(joystickGetDigital(MASTER, 7, JOY_DOWN)){
    lower_intake();
  }
  else set_intake_motor(0);
}
```

7.30 mobile_goal_intake.h

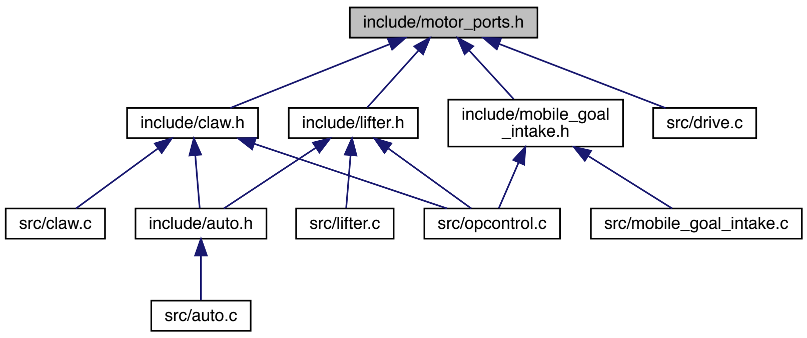
```
#ifndef _MOBLE_GOAL_INTAKE_
#define  _MOBLE_GOAL_INTAKE_

#include "motor_ports.h"
#include "controller.h"
#include "slew.h"

void update_intake();

#endif
```

7.31 include/motor_ports.h File Reference

The motor port definitions

Macros for the different motors ports.

This graph shows which files directly or indirectly include this file:



Macros

- #define **_MOTOR_PORTS_H_**
- #define **CLAW_MOTOR** 10
- #define **INTAKE_MOTOR** 8
- #define **MAX_SPEED** 127

  Max motor speed.
- #define **MIN_SPEED** -127
- #define **MOTOR_BACK_LEFT** 5

  Back left drive motor of robot base.
- #define **MOTOR_BACK_RIGHT** 4

  Back right drive motor of robot base.
- #define **MOTOR_FRONT_LEFT** 7

  Front left drive motor of robot base.
- #define **MOTOR_FRONT_RIGHT** 2

  Front right drive motor of robot base.
- #define **MOTOR_LIFT** 9
- #define **MOTOR_MIDDLE_LEFT** 6

  Middle left drive motor of robot base.
- #define **MOTOR_MIDDLE_RIGHT** 3

  Middle right drive motor of robot base.
- #define **MOTOR_SECONDARY_LIFTER** 1

7.31.1 Detailed Description

The motor port definitions

Macros for the different motors ports.

Definition in file **motor_ports.h**.

7.31.2 Macro Definition Documentation

7.31.2.1 _MOTOR_PORTS_H_

```
#define _MOTOR_PORTS_H_
```

Definition at line **7** of file **motor_ports.h**.

7.31.2.2 CLAW_MOTOR

```
#define CLAW_MOTOR 10
```

Definition at line **58** of file **motor_ports.h**.

Referenced by **close_claw()**, **open_claw()**, and **set_claw_motor()**.

7.31.2.3 INTAKE_MOTOR

```
#define INTAKE_MOTOR 8
```

Definition at line **60** of file **motor_ports.h**.

Referenced by **set_intake_motor()**.

7.31.2.4 MAX_SPEED

```
#define MAX_SPEED 127
```

Max motor speed.

Definition at line **12** of file **motor_ports.h**.

Referenced by **raise_main_lifter()**, **raise_secondary_lifter()**, and **update_lifter()**.

7.31.2.5 MIN_SPEED

```
#define MIN_SPEED -127
```

Definition at line **13** of file **motor_ports.h**.

Referenced by **lower_main_lifter()**, **lower_secondary_lifter()**, and **update_lifter()**.

7.31.2.6 MOTOR_BACK_LEFT

```
#define MOTOR_BACK_LEFT 5
```

Back left drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **54** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

7.31.2.7 MOTOR_BACK_RIGHT

```
#define MOTOR_BACK_RIGHT 4
```

Back right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **48** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

7.31.2.8 MOTOR_FRONT_LEFT

```
#define MOTOR_FRONT_LEFT 7
```

Front left drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **27** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

7.31.2.9 MOTOR_FRONT_RIGHT

```
#define MOTOR_FRONT_RIGHT 2
```

Front right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **20** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

7.31.2.10 MOTOR_LIFT

```
#define MOTOR_LIFT 9
```

Definition at line **56** of file **motor_ports.h**.

Referenced by **set_main_lifter_motors()**.

7.31.2.11 MOTOR_MIDDLE_LEFT

```
#define MOTOR_MIDDLE_LEFT 6
```

Middle left drive motor of robot base.

Date

9/7/2017

Author

Christian Desimone

Definition at line **41** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

7.31.2.12 MOTOR_MIDDLE_RIGHT

```
#define MOTOR_MIDDLE_RIGHT 3
```

Middle right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line **34** of file **motor_ports.h**.

Referenced by **set_side_speed()**.

7.31.2.13 MOTOR_SECONDARY_LIFTER

```
#define MOTOR_SECONDARY_LIFTER 1
```

Definition at line **59** of file **motor_ports.h**.

Referenced by **set_secondary_lifter_motors()**.

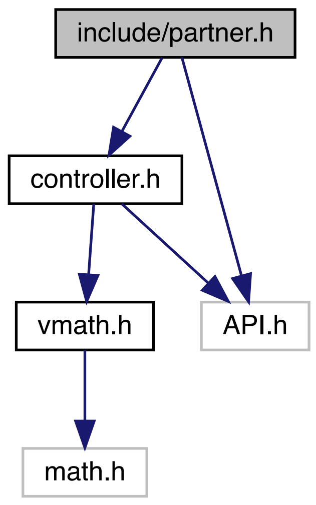
7.32 motor_ports.h

```

#ifndef _MOTOT_PORTS_H_
#define _MOTOR_PORTS_H_

#define MAX_SPEED 127
#define MIN_SPEED -127

#define MOTOR_FRONT_RIGHT 2

#define MOTOR_FRONT_LEFT 7

#define MOTOR_MIDDLE_RIGHT 3

#define MOTOR_MIDDLE_LEFT 6

#define MOTOR_BACK_RIGHT 4

#define MOTOR_BACK_LEFT 5

#define MOTOR_LIFT 9

#define CLAW_MOTOR 10
#define MOTOR_SECONDARY_LIFTER 1
#define INTAKE_MOTOR 8

#endif
```

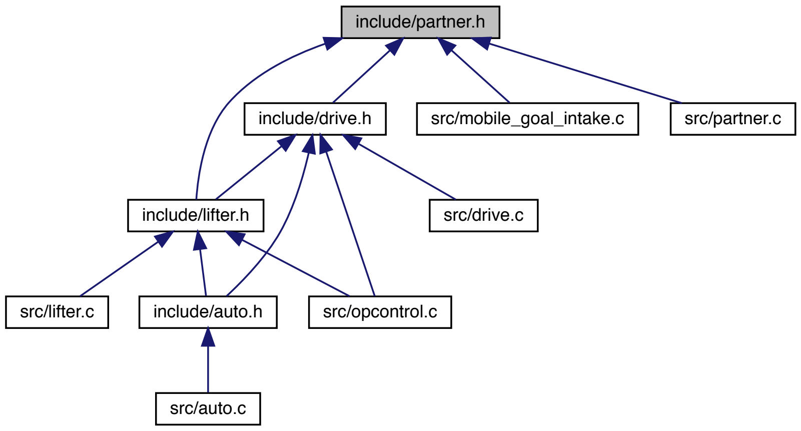
7.33 include/partner.h File Reference

```
#include "controller.h"
#include "API.h"
```

Include dependency graph for partner.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum **CONTROLL_MODE** { **MAIN_CONTROLLER_MODE**, **PARTNER_CONTROLLER_MODE** }

Functions

- enum **CONTROLL_MODE** **get_mode** ()
- void **update_control** ()

  Updates the controller mode between Driver and Partner modes.

7.33.1 Enumeration Type Documentation

7.33.1.1 CONTROLL_MODE

`enum` **`CONTROLL_MODE`**

Enumerator

MAIN_CONTROLLER_MODE	
PARTNER_CONTROLLER_MODE	

Definition at line **7** of file **partner.h**.

```
00007                        {
00008   MAIN_CONTROLLER_MODE,
00009   PARTNER_CONTROLLER_MODE
00010 };
```

7.33.2 Function Documentation

7.33.2.1 get_mode()

```
enum CONTROLL_MODE get_mode ( )
```

Definition at line **5** of file **partner.c**.

References **mode**.

Referenced by **update_drive_motors()**.

```
                                {
  return mode;
}
```

7.33.2.2 update_control()

```
void update_control ( )
```

Updates the controller mode between Driver and Partner modes.

Author

Chris Jerrett

Definition at line **9** of file **partner.c**.

References **MAIN_CONTROLLER_MODE**, **mode**, **PARTNER**, and **PARTNER_CONTROLLER_MODE**.

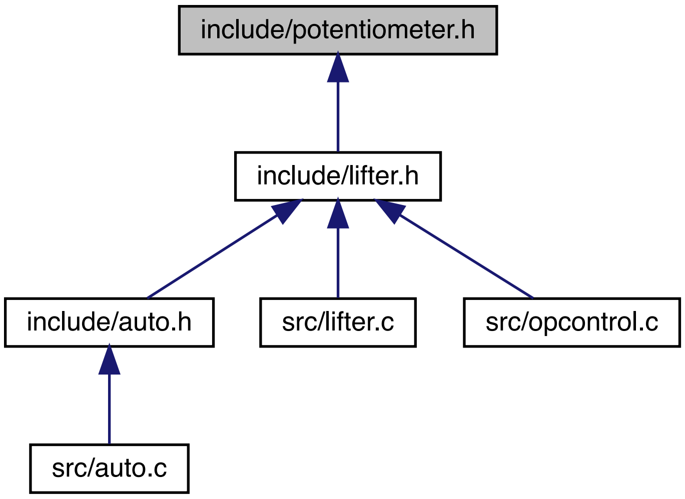
```
                      {
  if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
    mode = MAIN_CONTROLLER_MODE;
  } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
    mode = PARTNER_CONTROLLER_MODE;
  }
}
```

7.34 partner.h

```
#ifndef _PARTNER_H_
#define _PARTNER_H_

#include "controller.h"
#include "API.h"

enum CONTROLL_MODE {
  MAIN_CONTROLLER_MODE,
  PARTNER_CONTROLLER_MODE
};
void update_control();

enum CONTROLL_MODE get_mode();

#endif
```

7.35 include/potentiometer.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define **DEG_MAX** 250.0
- #define **TICK_MAX** 4095.0

7.35.1 Macro Definition Documentation

7.35.1.1 DEG_MAX

```
#define DEG_MAX 250.0
```

Definition at line **5** of file **potentiometer.h**.

Referenced by **lifterPotentiometerToDegree()**.

7.35.1.2 TICK_MAX

```
#define TICK_MAX 4095.0
```

Definition at line **4** of file **potentiometer.h**.

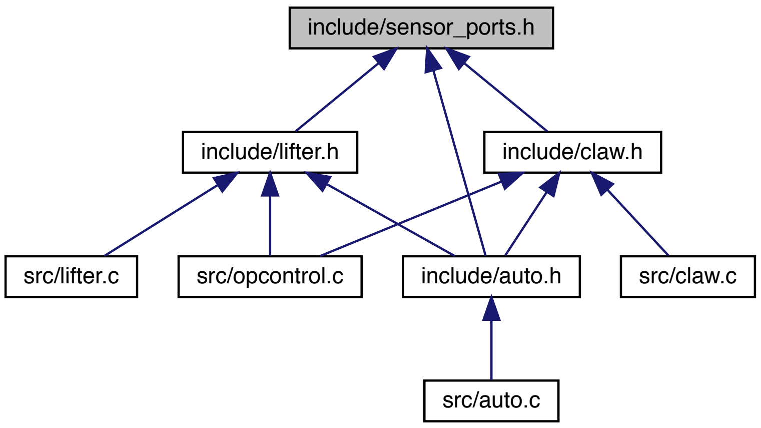
Referenced by **lifterPotentiometerToDegree()**.

7.36 potentiometer.h

```
#ifndef _POTENTIOMETER_H_
#define _POTENTIOMETER_H_

#define TICK_MAX 4095.0
#define DEG_MAX 250.0

#endif
```

7.37 include/sensor_ports.h File Reference

This graph shows which files directly or indirectly include this file:



Macros

- #define **CLAW_POT** 1
- #define **IME_FRONT_RIGHT** 0

  Number of integrated motor encoders Used when checking to see if all imes are plugged in.
- #define **LIFTER** 2

7.37.1 Macro Definition Documentation

7.37.1.1 CLAW_POT

```
#define CLAW_POT 1
```

Definition at line **21** of file **sensor_ports.h**.

7.37.1.2 IME_FRONT_RIGHT

```
#define IME_FRONT_RIGHT 0
```

Number of integrated motor encoders Used when checking to see if all imes are plugged in.

See also

init_encoders (p. 153)

Author

Christian Desimone

Date

9/7/2017

Definition at line **18** of file **sensor_ports.h**.

7.37.1.3 LIFTER

```
#define LIFTER 2
```

Definition at line **20** of file **sensor_ports.h**.

Referenced by **autonomous()**, and **getLifterTicks()**.

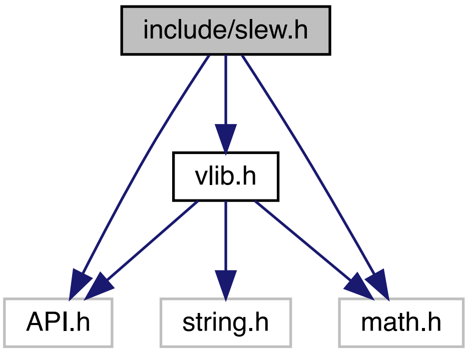
7.38 sensor_ports.h

```
00001
00008 #ifndef _PORTS_H_
00009 #define _PORTS_H_
00010
00018 #define IME_FRONT_RIGHT 0
00019 //#define POTENTIOMETER_PORT 2
00020 #define LIFTER 2
00021 #define CLAW_POT 1
00022
00023 #endif
```

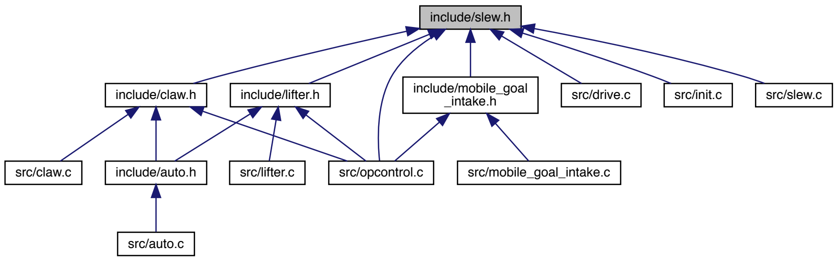
7.39 include/slew.h File Reference

Contains the slew rate controller wrapper for the motors.

```
#include <API.h>
#include <math.h>
#include <vlib.h>
```
Include dependency graph for slew.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define **MOTOR_PORTS** 12

  The number of motor ports on the robot.
- #define **RAMP_PROPORTION** 1

  proportion defining how quickly the motor should converge on the correct value. higher value leads to slower convergence
- #define **UPDATE_PERIOD_MS** 25

  How frequently to update the motors, in milliseconds.

Functions

- void **deinitslew** ()

  Deinitializes the slew rate controller and frees memory.
- void **init_slew** ()

  Initializes the slew rate controller.
- void **set_motor_immediate** (int motor, int speed)

  Sets the motor speed ignoring the slew controller.
- void **set_motor_slew** (int motor, int speed)

  Sets motor speed wrapped inside the slew rate controller.
- void **updateMotors** ()

  Closes the distance between the desired motor value and the current motor value by half for each motor.

7.39.1 Detailed Description

Contains the slew rate controller wrapper for the motors.

Author

Chris Jerrett

Date

9/14/17

Definition in file **slew.h**.

7.39.2 Macro Definition Documentation

7.39.2.1 MOTOR_PORTS

```
#define MOTOR_PORTS 12
```

The number of motor ports on the robot.

Author

Christian DeSimone

Date

9/14/17

Definition at line **27** of file **slew.h**.

7.39.2.2 RAMP_PROPORTION

```
#define RAMP_PROPORTION 1
```

proportion defining how quickly the motor should converge on the correct value. higher value leads to slower convergence

Author

Chris Jerrett

Date

9/14/17

Definition at line **34** of file **slew.h**.

7.39.2.3 UPDATE_PERIOD_MS

```
#define UPDATE_PERIOD_MS 25
```

How frequently to update the motors, in milliseconds.

Author

Chris Jerrett

Date

9/14/17

Definition at line **20** of file **slew.h**.

7.39.3 Function Documentation

7.39.3.1 deinitslew()

`void deinitslew ( )`

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line **58** of file **slew.c**.

References **initialized**, **motors_curr_speeds**, **motors_set_speeds**, and **slew**.

Referenced by **autonomous()**.

```
00058                   {
00059   taskDelete(slew);
00060   memset(motors_set_speeds, 0, sizeof(int) * 10);
00061   memset(motors_curr_speeds, 0, sizeof(int) * 10);
00062   initialized = false;
00063 }
```

7.39.3.2 init_slew()

`void init_slew ( )`

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

Definition at line **40** of file **slew.c**.

References **info()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **slew**, **speeds_mutex**, **update↩ Motors()**, and **warning()**.

Referenced by **autonomous()**, **operatorControl()**, **set_motor_immediate()**, and **set_motor_slew()**.

```
00040                  {
00041   if(initialized) {
00042     warning("Trying to init already init slew");
00043   }
00044   memset(motors_set_speeds, 0, sizeof(int) * 10);
00045   memset(motors_curr_speeds, 0, sizeof(int) * 10);
00046   motorStopAll();
00047   info("Did Init Slew");
00048   speeds_mutex = mutexCreate();
00049   slew = taskRunLoop(updateMotors, 100);
00050   initialized = true;
00051 }
```

7.39.3.3 set_motor_immediate()

```
void set_motor_immediate (
               int motor,
               int speed )
```

Sets the motor speed ignoring the slew controller.

Parameters

motor	the motor port to use
speed	the speed to use, between -127 and 127

Author

Chris Jerrett

Date

9/14/17

Definition at line **89** of file **slew.c**.

References **debug()**, **init_slew()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, and **speeds_↩ mutex**.

Referenced by **close_claw()**, **open_claw()**, **set_claw_motor()**, **set_intake_motor()**, **set_main_lifter_motors()**, and **set_secondary_lifter_motors()**.

```
                                          {
  if(!initialized) {
    debug("Slew Not Initialized! Initializing");
    init_slew();
  }
  motorSet(motor, speed);
  mutexTake(speeds_mutex, 10);
  motors_curr_speeds[motor-1] = speed;
  motors_set_speeds[motor-1] = speed;
  mutexGive(speeds_mutex);
}
```

7.39.3.4 set_motor_slew()

```
void set_motor_slew (
              int motor,
              int speed )
```

Sets motor speed wrapped inside the slew rate controller.

Parameters

motor	the motor port to use
speed	the speed to use, between -127 and 127

Author

Chris Jerrett

Date

9/14/17

Definition at line **72** of file **slew.c**.

References **debug()**, **init_slew()**, **initialized**, **motors_set_speeds**, and **speeds_mutex**.

Referenced by **set_side_speed()**.

```
                                          {
  if(!initialized) {
    debug("Slew Not Initialized! Initializing");
    init_slew();
  }
  mutexTake(speeds_mutex, 10);
  motors_set_speeds[motor-1] = speed;
  mutexGive(speeds_mutex);
}
```

7.39.3.5 updateMotors()

```
void updateMotors ( )
```

Closes the distance between the desired motor value and the current motor value by half for each motor.

Author

Chris Jerrett

Date

9/14/17

Definition at line **18** of file **slew.c**.

References **motors_curr_speeds**, **motors_set_speeds**, and **speeds_mutex**.

Referenced by **init_slew()**.

```
                    {
  //Take back half approach
  //Not linear but equal to setSpeed(1-(1/2)^x)
  for(unsigned int i = 0; i < 9; i++) {
    if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
    mutexTake(speeds_mutex, 10);
    int set_speed = (motors_set_speeds[i]);
    int curr_speed = motors_curr_speeds[i];
    mutexGive(speeds_mutex);
    int diff = set_speed - curr_speed;
    int offset = diff;
    int n = curr_speed + offset;
    motors_curr_speeds[i] = n;
    motorSet(i+1, n);
  }
}
```

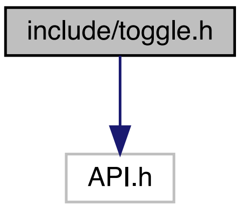
7.40 slew.h

```

#ifndef _SLEW_H_
#define _SLEW_H_

#include <API.h>
#include <math.h>
#include <vlib.h>

#define UPDATE_PERIOD_MS 25

#define MOTOR_PORTS 12

#define RAMP_PROPORTION 1

void updateMotors();

void deinitslew();

void init_slew();

void set_motor_slew(int motor, int speed);

void set_motor_immediate(int motor, int speed);

#endif
```

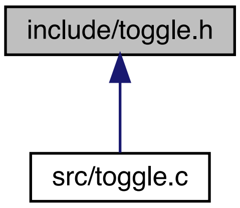
7.41 include/toggle.h File Reference

`#include <API.h>`

Include dependency graph for toggle.h:



This graph shows which files directly or indirectly include this file:



Enumerations

- enum **button_t** {
  JOY1_5D = 0, **JOY1_5U** = 1, **JOY1_6D** = 2, **JOY1_6U** = 3,
  JOY1_7U = 4, **JOY1_7L** = 5, **JOY1_7R** = 6, **JOY1_7D** = 7,
  JOY1_8U = 8, **JOY1_8L** = 9, **JOY1_8R** = 10, **JOY1_8D** = 11,
  JOY2_5D = 12, **JOY2_5U** = 13, **JOY2_6D** = 14, **JOY2_6U** = 15,
  JOY2_7U = 16, **JOY2_7L** = 17, **JOY2_7R** = 18, **JOY2_7D** = 19,
  JOY2_8U = 20, **JOY2_8L** = 21, **JOY2_8R** = 22, **JOY2_8D** = 23,
  LCD_LEFT = 24, **LCD_CENT** = 25, **LCD_RIGHT** = 26 }

Functions

- bool **buttonGetState** (**button_t**)

  Returns the current status of a button (pressed or not pressed)
- void **buttonInit** ()

  Initializes the buttons.
- bool **buttonIsNewPress** (**button_t**)

  Detects if button is a new press from most recent check by comparing previous value to current value.

7.41.1 Enumeration Type Documentation

7.41.1.1 button_t

`enum button_t`

Renames the input channels

Enumerator

JOY1_5D	
JOY1_5U	
JOY1_6D	
JOY1_6U	
JOY1_7U	
JOY1_7L	
JOY1_7R	
JOY1_7D	
JOY1_8U	
JOY1_8L	
JOY1_8R	
JOY1_8D	
JOY2_5D	
JOY2_5U	
JOY2_6D	
JOY2_6U	
JOY2_7U	
JOY2_7L	

Enumerator

JOY2_7R	
JOY2_7D	
JOY2_8U	
JOY2_8L	
JOY2_8R	
JOY2_8D	
LCD_LEFT	
LCD_CENT	
LCD_RIGHT	

Definition at line **20** of file **toggle.h**.

```
00020              {
00021     JOY1_5D = 0,
00022     JOY1_5U = 1,
00023     JOY1_6D = 2,
00024     JOY1_6U = 3,
00025     JOY1_7U = 4,
00026     JOY1_7L = 5,
00027     JOY1_7R = 6,
00028     JOY1_7D = 7,
00029     JOY1_8U = 8,
00030     JOY1_8L = 9,
00031     JOY1_8R = 10,
00032     JOY1_8D = 11,
00033
00034     JOY2_5D = 12,
00035     JOY2_5U = 13,
00036     JOY2_6D = 14,
00037     JOY2_6U = 15,
00038     JOY2_7U = 16,
00039     JOY2_7L = 17,
00040     JOY2_7R = 18,
00041     JOY2_7D = 19,
00042     JOY2_8U = 20,
00043     JOY2_8L = 21,
00044     JOY2_8R = 22,
00045     JOY2_8D = 23,
00046
00047     LCD_LEFT = 24,
00048     LCD_CENT = 25,
00049     LCD_RIGHT = 26
00050 } button_t;
```

7.41.2 Function Documentation

7.41.2.1 buttonGetState()

```
bool buttonGetState (
             button_t  )
```

Returns the current status of a button (pressed or not pressed)

Parameters

button	The button to detect from the Buttons enumeration.

Returns

true (pressed) or false (not pressed)

Definition at line **25** of file **toggle.c**.

References **LCD_CENT**, **LCD_LEFT**, and **LCD_RIGHT**.

Referenced by **buttonIsNewPress()**.

```
                                    {
    bool currentButton = false;

    // Determine how to get the current button value (from what function) and where it
    // is, then get it.
    if (button < LCD_LEFT) {
        // button is a joystick button
        unsigned char joystick;
        unsigned char buttonGroup;
        unsigned char buttonLocation;

        button_t newButton;
        if (button <= 11) {
            // button is on joystick 1
            joystick = 1;
            newButton = button;
        }
        else {
            // button is on joystick 2
            joystick = 2;
            // shift button down to joystick 1 buttons in order to
            // detect which button on joystick is queried
            newButton = (button_t)(button - 12);
        }

        switch (newButton) {
        case 0:
            buttonGroup = 5;
            buttonLocation = JOY_DOWN;
            break;
        case 1:
            buttonGroup = 5;
            buttonLocation = JOY_UP;
            break;
        case 2:
            buttonGroup = 6;
            buttonLocation = JOY_DOWN;
            break;
        case 3:
            buttonGroup = 6;
            buttonLocation = JOY_UP;
            break;
        case 4:
            buttonGroup = 7;
            buttonLocation = JOY_UP;
            break;
        case 5:
            buttonGroup = 7;
            buttonLocation = JOY_LEFT;
            break;
        case 6:
            buttonGroup = 7;
            buttonLocation = JOY_RIGHT;
            break;
        case 7:
            buttonGroup = 7;
            buttonLocation = JOY_DOWN;
            break;
        case 8:
            buttonGroup = 8;
            buttonLocation = JOY_UP;
            break;
        case 9:
            buttonGroup = 8;
            buttonLocation = JOY_LEFT;
            break;
        case 10:
            buttonGroup = 8;
            buttonLocation = JOY_RIGHT;
            break;
        case 11:
            buttonGroup = 8;
```

```
00097             buttonLocation = JOY_DOWN;
00098             break;
00099         default:
00100             break;
00101         }
00102         currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
00103     }
00104     else {
00105         // button is on LCD
00106         if (button == LCD_LEFT)
00107             currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);
00108 
00109         if (button == LCD_CENT)
00110             currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);
00111 
00112         if (button == LCD_RIGHT)
00113             currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
00114     }
00115     return currentButton;
00116 }
```

7.41.2.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons.

Initializes the buttons.

Definition at line **20** of file **toggle.c**.

References **buttonPressed**.

```
00020                   {
00021     for (int i = 0; i < 27; i++)
00022         buttonPressed[i] = false;
00023 }
```

7.41.2.3 buttonIsNewPress()

```
bool buttonIsNewPress (
                button_t button )
```

Detects if button is a new press from most recent check by comparing previous value to current value.

Parameters

button	The button to detect from the Buttons enumeration (see include/buttons.h).

Returns

true or false depending on if there was a change in button state.

Parameters

button	The button to detect from the Buttons enumeration (see include/buttons.h).

Returns

true or false depending on if there was a change in button state.

Example code:

```
...
if(buttonIsNewPress(JOY1_8D))
    digitalWrite(1, !digitalRead(1));
...
```

Definition at line **135** of file **toggle.c**.

References **buttonGetState()**, and **buttonPressed**.

```
00135                                       {
00136     bool currentButton = buttonGetState(button);
00137
00138     if (!currentButton) // buttons is not currently pressed
00139         buttonPressed[button] = false;
00140
00141     if (currentButton && !buttonPressed[button]) {
00142         // button is currently pressed and was not detected as being pressed during last check
00143         buttonPressed[button] = true;
00144         return true;
00145     }
00146     else return false; // button is not pressed or was already detected
00147 }
```

7.42 toggle.h

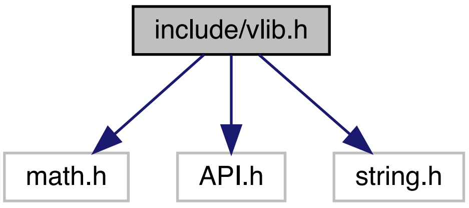
```
#ifndef BUTTONS_H_
#define BUTTONS_H_

#include <API.h>

typedef enum {
    JOY1_5D = 0,
    JOY1_5U = 1,
    JOY1_6D = 2,
    JOY1_6U = 3,
    JOY1_7U = 4,
    JOY1_7L = 5,
    JOY1_7R = 6,
    JOY1_7D = 7,
    JOY1_8U = 8,
    JOY1_8L = 9,
    JOY1_8R = 10,
    JOY1_8D = 11,

    JOY2_5D = 12,
    JOY2_5U = 13,
    JOY2_6D = 14,
    JOY2_6U = 15,
    JOY2_7U = 16,
    JOY2_7L = 17,
    JOY2_7R = 18,
    JOY2_7D = 19,
    JOY2_8U = 20,
    JOY2_8L = 21,
    JOY2_8R = 22,
    JOY2_8D = 23,

    LCD_LEFT = 24,
    LCD_CENT = 25,
    LCD_RIGHT = 26
} button_t;

void buttonInit();

bool buttonIsNewPress(button_t);

bool buttonGetState(button_t);

#endif
```

7.43 include/vlib.h File Reference

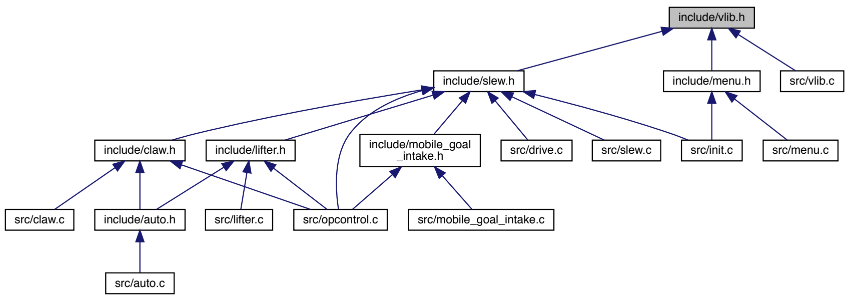
Contains misc helpful functions.

```
#include <math.h>
#include <API.h>
#include <string.h>
```

Include dependency graph for vlib.h:



This graph shows which files directly or indirectly include this file:



Functions

- void ∗ **calloc_real** (size_t elements, size_t size)
- void **ftoaa** (float a, char ∗buffer, int precision)

    converts a float to string.

- int **itoaa** (int a, char ∗buffer, int digits)
  converts a int to string.
- void **reverse** (char ∗str, int len)
  reverses a string 'str' of length 'len'

7.43.1 Detailed Description

Contains misc helpful functions.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **vlib.h**.

7.43.2 Function Documentation

7.43.2.1 calloc_real()

```
void* calloc_real (
             size_t elements,
             size_t size )
```

7.43.2.2 ftoaa()

```
void ftoaa (
             float a,
             char * buffer,
             int precision )
```

converts a float to string.

Parameters

a	the float
buffer	the string the float will be written to.
precision	digits after the decimal to write

Author

Christian DeSimone

Date

9/26/2017

Definition at line **55** of file **vlib.c**.

References **itoaa()**.

Referenced by **calculate_current_display()**.

```
                                                {

  // Extract integer part
  int ipart = (int)a;

  // Extract floating part
  float fpart = a - (float)ipart;

  // convert integer part to string
  int i = itoaa(ipart, buffer, 0);

  // check for display option after point
  if(precision != 0) {
    buffer[i] = '.';  // add dot

    // Get the value of fraction part up to given num.
    // of points after dot. The third parameter is needed
    // to handle cases like 233.007
    fpart = fpart * pow(10, precision);

    itoaa((int)fpart, buffer + i + 1, precision);
  }
}
```

7.43.2.3 itoaa()

```
int itoaa (
               int a,
               char * buffer,
               int digits )
```

converts a int to string.

Parameters

a	the integer
buffer	the string the int will be written to.
digits	the number of digits to be written

Returns

the digits

Author

Chris Jerrett, Christian DeSimone

Date

9/9/2017

Definition at line **30** of file **vlib.c**.

References **reverse()**.

Referenced by **ftoaa()**.

```
                                              {
    int i = 0;
     while (a) {
         buffer[i++] = (a%10) + '0';
         a = a/10;
     }

     // If number of digits required is more, then
     // add 0s at the beginning
     while (i < digits)
         buffer[i++] = '0';

     reverse(buffer, i);
     buffer[i] = '\0';
     return i;
 }
```

7.43.2.4 reverse()

```
void reverse (
              char * str,
              int len )
```

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

str	the string to reverse
len	the length

Definition at line **10** of file **vlib.c**.

Referenced by **itoaa()**.

```
                                {
    int i=0, j=len-1, temp;
    while (i<j) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++; j--;
    }
}
```

7.44 vlib.h

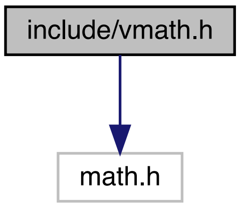
```

#ifndef _VLIB_H_
#define _VLIB_H_

#include <math.h>
#include <API.h>
#include <string.h>

void reverse(char *str, int len);

int itoaa(int a, char *buffer, int digits);


void ftoaa(float a, char *buffer, int precision);

void *calloc_real(size_t elements, size_t size);


#endif
```

7.45 include/vmath.h File Reference

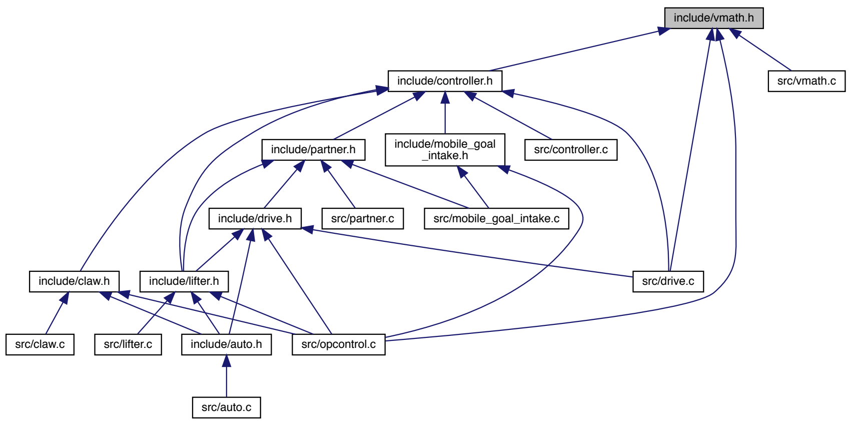
Vex Specific Math Functions, includes: Cartesian to polar cordinates.

`#include <math.h>`

Include dependency graph for vmath.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct **cord**

  A struct that contains cartesian coordinates.
- struct **polar_cord**

  A struct that contains polar coordinates.

Macros

- #define **M_PI** 3.14159265358979323846

Functions

- struct **polar_cord** **cartesian_cord_to_polar** (struct **cord** cords)

  Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.
- struct **polar_cord** **cartesian_to_polar** (float x, float y)

  Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- int **max** (int a, int b)

  the min of two values
- int **min** (int a, int b)

  the min of two values
- double **sind** (double angle)

  sine of a angle in degrees

7.45.1 Detailed Description

Vex Specific Math Functions, includes: Cartesian to polar cordinates.

Author

Christian Desimone
Chris Jerrett

Date

9/9/2017

Definition in file **vmath.h**.

7.45.2 Macro Definition Documentation

7.45.2.1 M_PI

```
#define M_PI 3.14159265358979323846
```

Definition at line **13** of file **vmath.h**.

Referenced by **sind()**.

7.45.3 Function Documentation

7.45.3.1 cartesian_cord_to_polar()

```
struct  polar_cord cartesian_cord_to_polar (
             struct  cord cords )
```

Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

cords	the cartesian cords

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 22)
cord (p. 13)

Definition at line **55** of file **vmath.c**.

References **cartesian_to_polar()**.

```
{
  return cartesian_to_polar(cords.x, cords.y);
}
```

7.45.3.2 cartesian_to_polar()

```
struct  polar_cord cartesian_to_polar (
            float x,
            float y )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

x	float value of the x cartesian coordinate.
y	float value of the y cartesian coordinate.

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 22)

Definition at line **14** of file **vmath.c**.

References **polar_cord::angle**, and **polar_cord::magnitue**.

Referenced by **cartesian_cord_to_polar()**.

```
{
  float degree = 0;
  double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));

  if(x < 0){
    degree += 180.0;
  }
  else if(x > 0 && y < 0){
    degree += 360.0;
  }

  if(x != 0 && y != 0){
    degree += atan((float)y / (float)x);
  }
  else if(x == 0 && y > 0){
```

```
00029     degree = 90.0;
00030   }
00031   else if(y == 0 && x < 0){
00032     degree = 180.0;
00033   }
00034   else if(x == 0 && y < 0){
00035     degree = 270.0;
00036   }
00037 
00038   struct polar_cord p;
00039   p.angle = degree;
00040   p.magnitue = magnitude;
00041   return p;
00042 }
```

7.45.3.3 max()

```
int max (
              int a,
              int b )
```

the min of two values

Parameters

a	the first
b	the second

Returns

the smaller of a and b

Definition at line **84** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
00084                         {
00085   if(a > b) return a;
00086   return b;
00087 }
```

7.45.3.4 min()

```
int min (
              int a,
              int b )
```

the min of two values

Parameters

a	the first
b	the second

Returns

the smaller of a and b

Definition at line **73** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
00073                         {
00074   if(a < b) return a;
00075   return b;
00076 }
```

7.45.3.5 sind()

```
double sind (
              double angle )
```

sine of a angle in degrees

Definition at line **62** of file **vmath.c**.

References **M_PI**.

```
00062                             {
00063     double angleradians = angle * M_PI / 180.0f;
00064     return sin(angleradians);
00065 }
```

7.46 vmath.h

```
00001
00009 #ifndef _VMATH_H_
00010 #define _VMATH_H_
00011
00012 #include <math.h>
00013 #define M_PI 3.14159265358979323846
00014
00020 struct polar_cord {
00022   float angle;
00024   float magnitue;
00025 };
00026
00032 struct cord {
00034   float x;
00036   float y;
00037 };
00038
00050 struct polar_cord cartesian_to_polar(float x, float y);
00051
00063 struct polar_cord cartesian_cord_to_polar(struct cord cords);
00064
00071 int min(int a, int b);
00072
00079 int max(int a, int b);
00080
00084 double sind(double angle);
00085 #endif
```

7.47 README.md File Reference

7.48 README.md

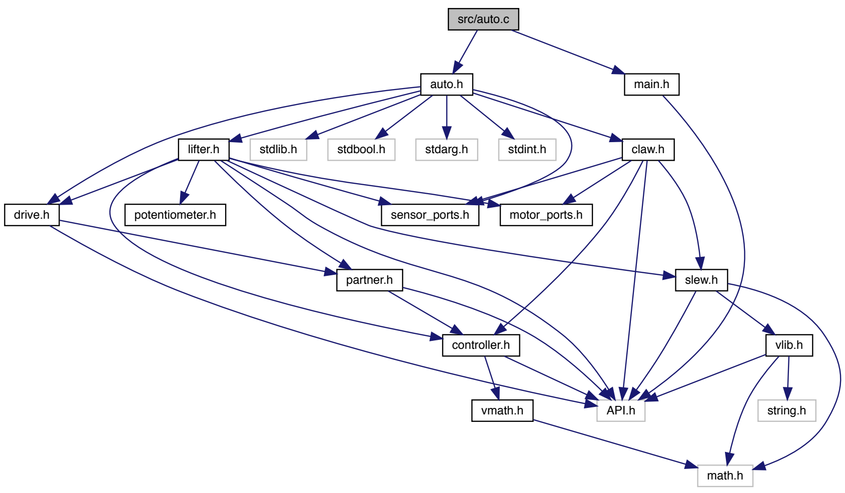
```
00001 # InTheZoneA
00002 Team A code for In The Zone
```

7.49 src/auto.c File Reference

File for autonomous code.

```
#include "main.h"
#include "auto.h"
```

Include dependency graph for auto.c:



Functions

- void **autonomous** ()

7.49.1 Detailed Description

File for autonomous code.

This file should contain the user **autonomous()** (p. 138) function and any functions related to it.

Any copyright is dedicated to the Public Domain. `http://creativecommons.org/publicdomain/zero/1.↩0/`

PROS contains FreeRTOS (`http://www.freertos.org`) whose source code may be obtained from `http↩://sourceforge.net/projects/freertos/files/` or on request.

Definition in file **auto.c**.

7.49.2 Function Documentation

7.49.2.1 autonomous()

```
void autonomous ( )
```

Runs the user autonomous code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is lost, the autonomous task will be stopped by the kernel. Re-enabling the robot will restart the task, not re-start it from where it left off.

Code running in the autonomous task cannot access information from the VEX Joystick. However, the autonomous function can be invoked from another task if a VEX Competition Switch is not available, and it can access joystick information if called in this way.

The autonomous task may exit, unlike **operatorControl()** (p. 84) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

Definition at line **30** of file **auto.c**.

References **BOTH**, **close_claw()**, **deinitslew()**, **GOAL_HEIGHT**, **init_slew()**, **LIFTER**, **MID_LEFT_DRIVE**, **MID_RIGHT_DRIVE**, **open_claw()**, and **set_side_speed()**.

```
                   {
  init_slew();

  delay(10);
  printf("auto\n");
  //How far the left wheels have gone
  int counts_drive_left;
  //How far the right wheels have gone
  int counts_drive_right;
  //The average distance traveled forward
  int counts_drive;

  //Reset the integrated motor controllers
  imeReset(MID_LEFT_DRIVE);
  imeReset(MID_RIGHT_DRIVE);
  //Set initial values for how far the wheels have gone
  imeGet(MID_LEFT_DRIVE, &counts_drive_left);
  imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
  counts_drive = counts_drive_left + counts_drive_right;
  counts_drive /= 2;

  //Grab pre-load cone
  close_claw();
  delay(300);

  //Raise the lifter
  while(analogRead(LIFTER) < GOAL_HEIGHT){
    //set_lifter_motors(-127);
  }
  //set_lifter_motors(0);
  //Drive towards the goal
  while(counts_drive < 530){
    set_side_speed(BOTH, 127);
    //Restablish the distance traveled
    imeGet(MID_LEFT_DRIVE, &counts_drive_left);
    imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
    counts_drive = counts_drive_left + counts_drive_right;
    counts_drive /= 2;
  }
  //Stop moving
  set_side_speed(BOTH, 0);
  delay(1000);

  //Drop the cone on the goal
  open_claw();
  delay(1000);
  deinitslew();
}
```

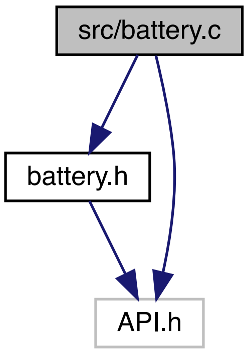
7.50 auto.c

```

#include "main.h"
#include "auto.h"

/*
 * Runs the user autonomous code. This function will be started in its own task with the default
 * priority and stack size whenever the robot is enabled via the Field Management System or the
 * VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is
 * lost,  the autonomous task will be stopped by the kernel. Re-enabling the robot will restart
 * the task, not re-start it from where it left off.
 *
 * Code running in the autonomous task cannot access information from the VEX Joystick. However,
 * the autonomous function can be invoked from another task if a VEX Competition Switch is not
 * available, and it can access joystick information if called in this way.
 *
 * The autonomous task may exit, unlike operatorControl() which should never exit. If it does
 * so, the robot will await a switch to another mode or disable/enable cycle.
 */
void autonomous() {
  init_slew();

  delay(10);
  printf("auto\n");
  //How far the left wheels have gone
  int counts_drive_left;
  //How far the right wheels have gone
  int counts_drive_right;
  //The average distance traveled forward
  int counts_drive;

  //Reset the integrated motor controllers
  imeReset(MID_LEFT_DRIVE);
  imeReset(MID_RIGHT_DRIVE);
  //Set initial values for how far the wheels have gone
  imeGet(MID_LEFT_DRIVE, &counts_drive_left);
  imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
  counts_drive = counts_drive_left + counts_drive_right;
  counts_drive /= 2;

  //Grab pre-load cone
  close_claw();
  delay(300);

  //Raise the lifter
  while(analogRead(LIFTER) < GOAL_HEIGHT){
    //set_lifter_motors(-127);
  }
  //set_lifter_motors(0);
  //Drive towards the goal
  while(counts_drive < 530){
    set_side_speed(BOTH, 127);
    //Restablish the distance traveled
    imeGet(MID_LEFT_DRIVE, &counts_drive_left);
    imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
    counts_drive = counts_drive_left + counts_drive_right;
    counts_drive /= 2;
  }
  //Stop moving
  set_side_speed(BOTH, 0);
  delay(1000);

  //Drop the cone on the goal
  open_claw();
  delay(1000);
  deinitslew();
}
```

7.51 src/battery.c File Reference

```
#include "battery.h"
#include <API.h>
```

Include dependency graph for battery.c:



Functions

- double **backup_battery_voltage** ()

  gets the backup battery voltage
- bool **battery_level_acceptable** ()

  returns if the batteries are acceptable
- double **main_battery_voltage** ()

  gets the main battery voltage

7.51.1 Function Documentation

7.51.1.1 backup_battery_voltage()

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

Author

Chris Jerrett

Definition at line **17** of file **battery.c**.

Referenced by **battery_level_acceptable()**.

```
00017                                {
00018   return powerLevelBackup() / 1000.0;
00019 }
```

7.51.1.2 battery_level_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

See also

MIN_MAIN_VOLTAGE (p. 28)
MIN_BACKUP_VOLTAGE (p. 28)

Author

Chris Jerrett

Definition at line **28** of file **battery.c**.

References **backup_battery_voltage()**, **main_battery_voltage()**, **MIN_BACKUP_VOLTAGE**, and **MIN_MAIN_VOLTAGE**.

```
00028                                {
00029   if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
00030   if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
00031   return true;
00032 }
```

7.51.1.3 main_battery_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

Author

Chris Jerrett

Definition at line **9** of file **battery.c**.

Referenced by **battery_level_acceptable()**.

```
00009                              {
00010   return powerLevelMain() / 1000.0;
00011 }
```

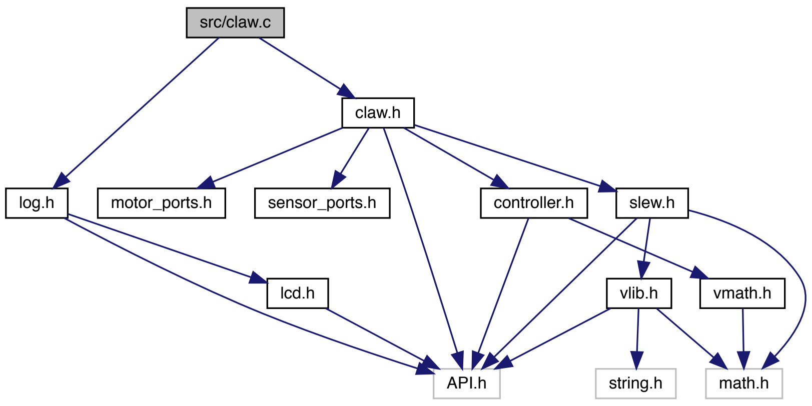
7.52 battery.c

```
#include "battery.h"
#include <API.h>


double main_battery_voltage() {
  return powerLevelMain() / 1000.0;
}

double backup_battery_voltage() {
  return powerLevelBackup() / 1000.0;
}

bool battery_level_acceptable() {
  if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
  if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
  return true;
}
```

7.53 src/claw.c File Reference

```
#include "claw.h"
#include "log.h"
```

Include dependency graph for claw.c:



Functions

- void **close_claw** ()

  Drives the motors to close the claw.
- void **open_claw** ()

  Drives the motors to open the claw.
- void **set_claw_motor** (const int v)

  sets the claw motor speed
- void **update_claw** ()

  Updates the claw motor values.

Variables

- static enum **claw_state** **state** = **CLAW_NEUTRAL_STATE**

7.53.1 Function Documentation

7.53.1.1 close_claw()

```
void close_claw ( )
```

Drives the motors to close the claw.

Author

Chris Jerrett

Definition at line **48** of file **claw.c**.

References **CLAW_MOTOR**, **MIN_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00048                   {
00049   set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
00050 }
```

7.53.1.2 open_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

Author

Chris Jerrett

Definition at line **40** of file **claw.c**.

References **CLAW_MOTOR**, **MAX_CLAW_SPEED**, and **set_motor_immediate()**.

Referenced by **autonomous()**.

```
00040                  {
00041   set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);
00042 }
```

7.53.1.3 set_claw_motor()

```
void set_claw_motor (
             const int v )
```

sets the claw motor speed

Author

Chris Jerrett

Definition at line **31** of file **claw.c**.

References **CLAW_MOTOR**, and **set_motor_immediate()**.

Referenced by **update_claw()**.

```
                              {
  set_motor_immediate(CLAW_MOTOR, v);
}
```

7.53.1.4 update_claw()

```
void update_claw ( )
```

Updates the claw motor values.

Author

Chris Jerrett

Definition at line **9** of file **claw.c**.

References **CLAW_CLOSE**, **CLAW_CLOSE_STATE**, **CLAW_NEUTRAL_STATE**, **CLAW_OPEN**, **CLAW_OPEN_STATE**, **MAX_CLAW_SPEED**, **MIN_CLAW_SPEED**, **set_claw_motor()**, and **state**.

Referenced by **operatorControl()**.

```
                    {
  if(joystickGetDigital(CLAW_CLOSE)) {
    state = CLAW_CLOSE_STATE;
  } else if(joystickGetDigital(CLAW_OPEN)) {
    state = CLAW_OPEN_STATE;
  } else {
    state = CLAW_NEUTRAL_STATE;
  }

  if(state == CLAW_CLOSE_STATE) {
    set_claw_motor(MAX_CLAW_SPEED);
  } else if(state == CLAW_OPEN_STATE) {
    set_claw_motor(MIN_CLAW_SPEED);
  } else {
    set_claw_motor(0);
  }
}
```

7.53.2 Variable Documentation

7.53.2.1 state

```
enum claw_state state = CLAW_NEUTRAL_STATE [static]
```

Definition at line **3** of file **claw.c**.

Referenced by **update_claw()**.

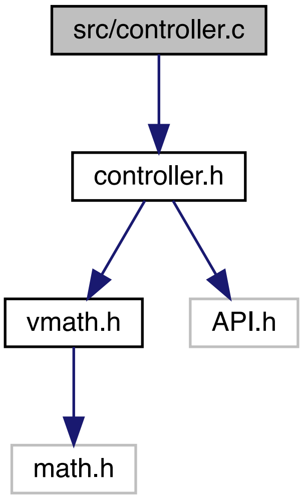
7.54 claw.c

```
00001 #include "claw.h"
00002 #include "log.h"
00003 static enum claw_state state = CLAW_NEUTRAL_STATE;
00004
00009 void update_claw() {
00010   if(joystickGetDigital(CLAW_CLOSE)) {
00011     state = CLAW_CLOSE_STATE;
00012   } else if(joystickGetDigital(CLAW_OPEN)) {
00013     state = CLAW_OPEN_STATE;
00014   } else {
00015     state = CLAW_NEUTRAL_STATE;
00016   }
00017
00018   if(state == CLAW_CLOSE_STATE) {
00019     set_claw_motor(MAX_CLAW_SPEED);
00020   } else if(state == CLAW_OPEN_STATE) {
00021     set_claw_motor(MIN_CLAW_SPEED);
00022   } else {
00023     set_claw_motor(0);
00024   }
00025 }
00026
00031 void set_claw_motor(const int v){
00032   set_motor_immediate(CLAW_MOTOR, v);
00033 }
00034
00035
00040 void open_claw() {
00041   set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);
00042 }
00043
00048 void close_claw() {
00049   set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
00050 }
```

7.55 src/controller.c File Reference

```
#include "controller.h"
```

Include dependency graph for controller.c:



Functions

- struct **cord** **get_joystick_cord** (enum **joystick** **side**, int controller)

  Gets the location of a joystick on the controller.

7.55.1 Function Documentation

7.55.1.1 get_joystick_cord()

```
struct  cord get_joystick_cord (
            enum  joystick side,
            int controller )
```

Gets the location of a joystick on the controller.

Author

Chris Jerrett

Definition at line **7** of file **controller.c**.

References **LEFT_JOY_X**, **LEFT_JOY_Y**, **RIGHT_JOY**, **RIGHT_JOY_X**, **RIGHT_JOY_Y**, **cord::x**, and **cord::y**.

```
                                                                  {
  int x;
  int y;
  //Get the joystick value for either the right or left,
  //depending on the mode
  if(side == RIGHT_JOY) {
    y = joystickGetAnalog(controller, RIGHT_JOY_X);
    x = joystickGetAnalog(controller, RIGHT_JOY_Y);
  } else {
    y = joystickGetAnalog(controller, LEFT_JOY_X);
    x = joystickGetAnalog(controller, LEFT_JOY_Y);
  }
  //Define a coordinate for the joystick value
  struct cord c;
  c.x = x;
  c.y = y;
  return c;
}
```

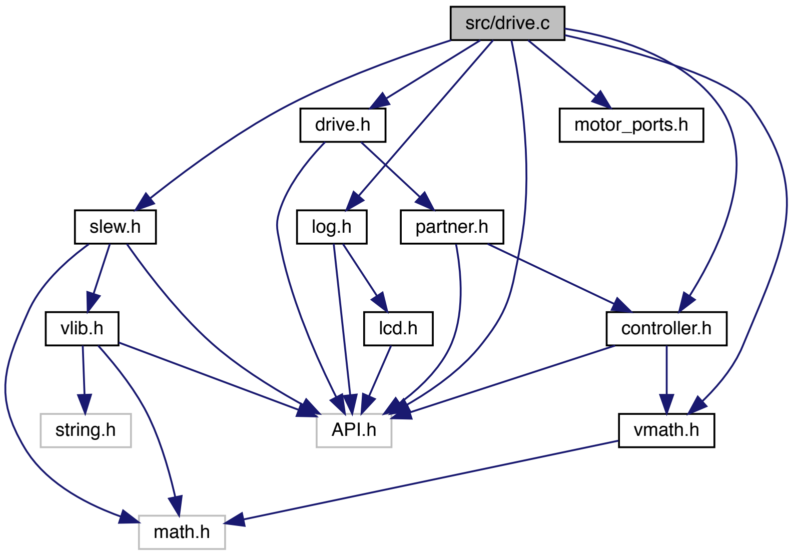
7.56 controller.c

```
#include "controller.h"

struct cord get_joystick_cord(enum joystick side, int controller) {
  int x;
  int y;
  //Get the joystick value for either the right or left,
  //depending on the mode
  if(side == RIGHT_JOY) {
    y = joystickGetAnalog(controller, RIGHT_JOY_X);
    x = joystickGetAnalog(controller, RIGHT_JOY_Y);
  } else {
    y = joystickGetAnalog(controller, LEFT_JOY_X);
    x = joystickGetAnalog(controller, LEFT_JOY_Y);
  }
  //Define a coordinate for the joystick value
  struct cord c;
  c.x = x;
  c.y = y;
  return c;
}
```

7.57 src/drive.c File Reference

```
#include "drive.h"
#include "motor_ports.h"
#include "vmath.h"
#include "controller.h"
#include "slew.h"
#include <API.h>
#include "log.h"
```

Include dependency graph for drive.c:



Functions

- int **getThresh** ()

  Gets the deadzone threshhold on the drive.
- static float **joystickExp** (int joystickVal)

  Applies exponential scale to a joystick value.
- void **set_side_speed** (**side_t side**, int speed)

  sets the speed of one side of the robot.
- void **setThresh** (int t)

  Sets the deadzone threshhold on the drive.
- void **update_drive_motors** ()

  Updates the drive motors during teleop.

Variables

- static int **thresh** = 30

7.57.1 Function Documentation

7.57.1.1 getThresh()

```
int getThresh ( )
```

Gets the deadzone threshhold on the drive.

Author

Christian Desimone

Definition at line **17** of file **drive.c**.

References **thresh**.

```
                {
  return thresh;
}
```

7.57.1.2 joystickExp()

```
static float joystickExp (
              int joystickVal )  [static]
```

Applies exponential scale to a joystick value.

Author

Christian DeSimone, Chris Jerrett

Parameters

joystickVal	the analog value from the joystick

Date

9/21/2017

Definition at line **87** of file **drive.c**.

References **THRESHOLD**.

```
                                           {
    //make the offset negative if moving backwards
    if (abs(joystickVal) < THRESHOLD) {
            return 0;
    }

    int offset;
  //Use the threshold to ensure the joystick values are significant
    if (joystickVal < 0) {
        offset = - (THRESHOLD);
    } else {
        offset = THRESHOLD;
    }
  //Apply the function ((((x/10)^3)/18) + offset) * 0.8 to the joystick value
    return (pow(joystickVal/10 , 3) / 18 + offset) * 0.8;
}
```

7.57.1.3 set_side_speed()

```
void set_side_speed (
              side_t side,
              int speed )
```

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

side	a side enum which indicates the size.
speed	the speed of the side. Can range from -127 - 127 negative being back and positive forwards

Definition at line **68** of file **drive.c**.

References **BOTH**, **LEFT**, **MOTOR_BACK_LEFT**, **MOTOR_BACK_RIGHT**, **MOTOR_FRONT_LEFT**, **MOTOR_FRONT_RIGHT**, **MOTOR_MIDDLE_LEFT**, **MOTOR_MIDDLE_RIGHT**, **RIGHT**, and **set_motor_slew()**.

Referenced by **autonomous()**, and **update_drive_motors()**.

```
                                          {
  if(side == RIGHT || side == BOTH){
    set_motor_slew(MOTOR_BACK_RIGHT , -speed);
    set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
    set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
  }
  if(side == LEFT || side == BOTH){
    set_motor_slew(MOTOR_BACK_LEFT, speed);
    set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
    set_motor_slew(MOTOR_FRONT_LEFT, speed);
  }
}
```

7.57.1.4 setThresh()

```
void setThresh (
              int t )
```

Sets the deadzone threshhold on the drive.

Author

Christian Desimone

Definition at line **25** of file **drive.c**.

References **thresh**.

```
                     {
  thresh = t;
}
```

7.57.1.5 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

Definition at line **34** of file **drive.c**.

References **get_mode()**, **LEFT**, **MASTER**, **PARTNER**, **PARTNER_CONTROLLER_MODE**, **RIGHT**, **set_side_speed()**, **thresh**, **cord::x**, and **cord::y**.

Referenced by **operatorControl()**.

```
                          {
  //Get the joystick values from the controller
  int x = 0;
  int y = 0;
  if(get_mode() == PARTNER_CONTROLLER_MODE) {
    x = (joystickGetAnalog(PARTNER, 3));
    y = (joystickGetAnalog(PARTNER, 1));
  } else {
    x = -(joystickGetAnalog(MASTER, 3));
    y = (joystickGetAnalog(MASTER, 1));
  }
  //Make sure the joystick values are significant enough to change the motors
  if(x < thresh && x > -thresh){
    x = 0;
  }
  if(y < thresh && y > -thresh){
    y = 0;
  }
  //Create motor values for the left and right from the x and y of the joystick
  int r = (x + y);
  int l = -(x - y);

  //Set the drive motors
  set_side_speed(LEFT, l);
  set_side_speed(RIGHT, -r);

}
```

7.57.2 Variable Documentation

7.57.2.1 thresh

```
int thresh = 30  [static]
```

Definition at line **10** of file **drive.c**.

Referenced by **getThresh()**, **setThresh()**, and **update_drive_motors()**.

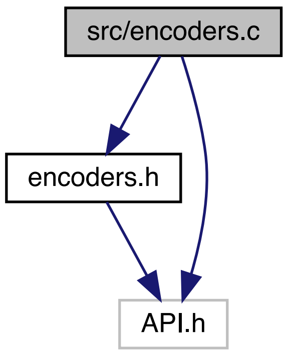
7.58 drive.c

```
00001 #include "drive.h"
00002 #include "motor_ports.h"
00003 #include "vmath.h"
00004 #include "controller.h"
00005 #include "slew.h"
00006 #include "controller.h"
00007 #include <API.h>
00008 #include "log.h"
00009
00010 static int thresh = 30;
00011
00012
00017 int getThresh(){
00018   return thresh;
00019 }
00020
00025 void setThresh(int t){
00026   thresh = t;
00027 }
00028
00034 void update_drive_motors(){
00035   //Get the joystick values from the controller
00036   int x = 0;
00037   int y = 0;
00038   if(get_mode() == PARTNER_CONTROLLER_MODE) {
00039     x = (joystickGetAnalog(PARTNER, 3));
00040     y = (joystickGetAnalog(PARTNER, 1));
00041   } else {
00042     x = -(joystickGetAnalog(MASTER, 3));
00043     y = (joystickGetAnalog(MASTER, 1));
00044   }
00045   //Make sure the joystick values are significant enough to change the motors
00046   if(x < thresh && x > -thresh){
00047     x = 0;
00048   }
00049   if(y < thresh && y > -thresh){
00050     y = 0;
00051   }
00052   //Create motor values for the left and right from the x and y of the joystick
00053   int r = (x + y);
00054   int l = -(x - y);
00055
00056   //Set the drive motors
00057   set_side_speed(LEFT, l);
00058   set_side_speed(RIGHT, -r);
00059
00060 }
00061
00068 void set_side_speed(side_t side, int speed){
00069   if(side == RIGHT || side == BOTH){
00070     set_motor_slew(MOTOR_BACK_RIGHT , -speed);
00071     set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
00072     set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
00073   }
00074   if(side == LEFT || side == BOTH){
00075     set_motor_slew(MOTOR_BACK_LEFT, speed);
00076     set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
00077     set_motor_slew(MOTOR_FRONT_LEFT, speed);
00078   }
}

static float joystickExp(int joystickVal) {
    //make the offset negative if moving backwards
    if (abs(joystickVal) < THRESHOLD) {
            return 0;
    }

    int offset;
  //Use the threshold to ensure the joystick values are significant
    if (joystickVal < 0) {
        offset = - (THRESHOLD);
    } else {
        offset = THRESHOLD;
    }
  //Apply the function ((((x/10)^3)/18) + offset) * 0.8 to the joystick value
    return (pow(joystickVal/10 , 3) / 18 + offset) * 0.8;
}
```

7.59 src/encoders.c File Reference

```
#include "encoders.h"
#include <API.h>
```

Include dependency graph for encoders.c:



Functions

- int **get_encoder_ticks** (unsigned char address)

  Gets the encoder ticks since last reset.
- int **get_encoder_velocity** (unsigned char address)

  Gets the encoder reads.
- bool **init_encoders** ()

  Initializes all motor encoders.

7.59.1 Function Documentation

7.59.1.1 get_encoder_ticks()

```
int get_encoder_ticks (
            unsigned char address )
```

Gets the encoder ticks since last reset.

Author

Chris Jerrett

Date

9/15/2017

Definition at line **23** of file **encoders.c**.

```
                                               {
  int i = 0;
  imeGet(address, &i);
  return i;
}
```

7.59.1.2 get_encoder_velocity()

```
int get_encoder_velocity (
            unsigned char address )
```

Gets the encoder reads.

Author

Chris Jerrett

Date

9/15/2017

Definition at line **34** of file **encoders.c**.

```
                                                  {
  int i = 0;
  imeGetVelocity(address, &i);
  return i;
}
```

7.59.1.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

IME_NUMBER (p. 48)

Definition at line **10** of file **encoders.c**.

References **IME_NUMBER**.

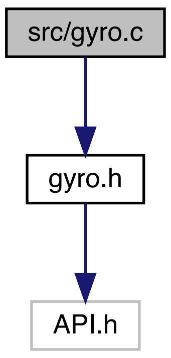
```
00010                       {
00011   #ifdef IME_NUMBER
00012   return imeInitializeAll() == IME_NUMBER;
00013   #else
00014   return imeInitializeAll();
00015   #endif
00016 }
```

7.60 encoders.c

```
00001 #include "encoders.h"
00002 #include <API.h>
00003 
00010 bool init_encoders() {
00011   #ifdef IME_NUMBER
00012   return imeInitializeAll() == IME_NUMBER;
00013   #else
00014   return imeInitializeAll();
00015   #endif
00016 }
00017 
00023 int get_encoder_ticks(unsigned char address) {
00024   int i = 0;
00025   imeGet(address, &i);
00026   return i;
00027 }
00028 
00034 int get_encoder_velocity(unsigned char address) {
00035   int i = 0;
00036   imeGetVelocity(address, &i);
00037   return i;
00038 }
```

7.61 src/gyro.c File Reference

`#include "gyro.h"`
Include dependency graph for gyro.c:



Functions

- float **get_main_gyro_angluar_velocity** ()
- bool **init_main_gyro** ()

Variables

- static Gyro **main_gyro**

7.61.1 Function Documentation

7.61.1.1 get_main_gyro_angluar_velocity()

```
float get_main_gyro_angluar_velocity ( )
```

Definition at line **11** of file **gyro.c**.

References **GYRO_PORT**.

```
                                        {
  uint32_t port = GYRO_PORT;
  int32_t reading = (int32_t)analogReadCalibratedHR(port + 1);
    return 0;
}
```

7.61.1.2 init_main_gyro()

```
bool init_main_gyro ( )
```

Definition at line **5** of file **gyro.c**.

References **GYRO_MULTIPLIER**, **GYRO_PORT**, and **main_gyro**.

```
                        {
  main_gyro = gyroInit(GYRO_PORT, GYRO_MULTIPLIER);
  return main_gyro != NULL;
}
```

7.61.2 Variable Documentation

7.61.2.1 main_gyro

```
Gyro main_gyro  [static]
```

Definition at line **3** of file **gyro.c**.

Referenced by **init_main_gyro()**.

7.62 gyro.c

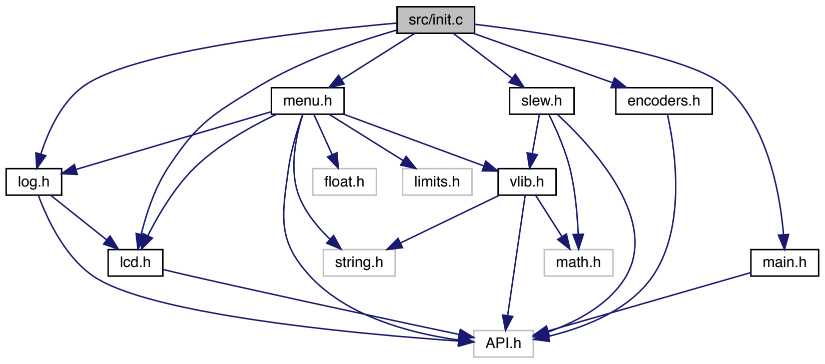
```
#include "gyro.h"

static Gyro main_gyro;

bool init_main_gyro() {
  main_gyro = gyroInit(GYRO_PORT, GYRO_MULTIPLIER);
  return main_gyro != NULL;
}


float get_main_gyro_angluar_velocity() {
  uint32_t port = GYRO_PORT;
  int32_t reading = (int32_t)analogReadCalibratedHR(port + 1);
    return 0;
}
```

7.63 src/init.c File Reference

File for initialization code.

```
#include "main.h"
#include "slew.h"
#include "lcd.h"
#include "log.h"
#include "encoders.h"
#include "menu.h"
```

Include dependency graph for init.c:



Functions

- void **initialize** ()
- void **initializeIO** ()

7.63.1 Detailed Description

File for initialization code.

This file should contain the user **initialize()** (p. 156) function and any functions related to it.

Any copyright is dedicated to the Public Domain. `http://creativecommons.org/publicdomain/zero/1.↩ 0/`

PROS contains FreeRTOS (`http://www.freertos.org`) whose source code may be obtained from `http↩ ://sourceforge.net/projects/freertos/files/` or on request.

Definition in file **init.c**.

7.63.2 Function Documentation

7.63.2.1 initialize()

```
void initialize ( )
```

Runs user initialization code. This function will be started in its own task with the default priority and stack size once when the robot is starting up. It is possible that the VEXnet communication link may not be fully established at this time, so reading from the VEX Joystick may fail.

This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global variables, and IMEs.

This function must exit relatively promptly, or the **operatorControl()** (p. 84) and **autonomous()** (p. 83) tasks will not start. An autonomous mode selection menu like the pre_auton() in other environments can be implemented in this task if desired.

Definition at line **47** of file **init.c**.

References **display_menu()**, **info()**, **init_error()**, **init_main_lcd()**, **init_menu_var()**, and **STRING_TYPE**.

```
                 {
  init_main_lcd(uart1);
  info("LCD Init");
  menu_t *t = init_menu_var(STRING_TYPE, "TEST Menu", 5, "1","2","3","4","5");
  init_error(true, uart2);
  display_menu(t);
  setTeamName("9228A");
}
```

7.63.2.2 initializeIO()

```
void initializeIO ( )
```

Runs pre-initialization code. This function will be started in kernel mode one time while the VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.

The purpose of this function is solely to set the default pin modes (pinMode()) and port states (digitalWrite()) of limit switches, push buttons, and solenoids. It can also safely configure a UART port (usartOpen()) but cannot set up an LCD (lcdInit()).

Definition at line **30** of file **init.c**.

```
                   {
    watchdogInit();
}
```

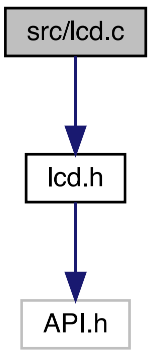
7.64 init.c

```
#include "main.h"
#include "slew.h"
#include "lcd.h"
#include "log.h"
#include "encoders.h"
#include "menu.h"

/*
 * Runs pre-initialization code. This function will be started in kernel mode one time while the
 * VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.
 *
 * The purpose of this function is solely to set the default pin modes (pinMode()) and port
 * states (digitalWrite()) of limit switches, push buttons, and solenoids. It can also safely
 * configure a UART port (usartOpen()) but cannot set up an LCD (lcdInit()).
 *
 * AKA DON'T USE
 * -Chris
 */
void initializeIO() {
    watchdogInit();
}

/*
 * Runs user initialization code. This function will be started in its own task with the default
 * priority and stack size once when the robot is starting up. It is possible that the VEXnet
 * communication link may not be fully established at this time, so reading from the VEX
 * Joystick may fail.
 *
 * This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global
 * variables, and IMEs.
 *
 * This function must exit relatively promptly, or the operatorControl() and autonomous() tasks
 * will not start. An autonomous mode selection menu like the pre_auton() in other environments
 * can be implemented in this task if desired.
 */
void initialize() {
  init_main_lcd(uart1);
  info("LCD Init");
  menu_t *t = init_menu_var(STRING_TYPE, "TEST Menu", 5, "1","2","3","4","5");
  init_error(true, uart2);
  display_menu(t);
  setTeamName("9228A");
}
```

7.65 src/lcd.c File Reference

`#include "lcd.h"`

Include dependency graph for lcd.c:



Functions

- void **init_main_lcd** (FILE ∗lcd)
  Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- static bool **lcd_assert** ()
  Asserts the lcd is initialized Works by checking is the File ∗lcd_port is the default NULL value and thus not set.
- void **lcd_clear** ()
  Clears the lcd.
- **lcd_buttons** **lcd_get_pressed_buttons** ()
  Returns the pressed buttons.
- void **lcd_print** (unsigned int line, const char ∗str)
  prints a string to a line on the lcd
- void **lcd_printf** (unsigned int line, const char ∗format_str,...)
  prints a formated string to a line on the lcd. Smilar to printf
- void **lcd_set_backlight** (bool **state**)
  sets the backlight of the lcd
- void **promt_confirmation** (const char ∗confirm_text)
  Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Variables

- static FILE ∗ **lcd_port** = NULL

7.65.1 Function Documentation

7.65.1.1 init_main_lcd()

```
void init_main_lcd (
            FILE * lcd )
```

Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.

Parameters

lcd	the urart port of the lcd screen

See also

uart1
uart2

Author

Chris Jerrett

Date

9/9/2017

Definition at line **61** of file **lcd.c**.

References **lcd_clear()**, and **lcd_port**.

Referenced by **initialize()**.

```
                               {
  lcd_port = lcd;
  lcdInit(lcd);
  lcd_clear();
}
```

7.65.1.2 lcd_assert()

```
static bool lcd_assert ( )  [static]
```

Asserts the lcd is initialized Works by checking is the File ∗lcd_port is the default NULL value and thus not set.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **13** of file **lcd.c**.

References **lcd_port**.

Referenced by **lcd_clear()**, **lcd_get_pressed_buttons()**, **lcd_print()**, **lcd_printf()**, **lcd_set_backlight()**, and **promt_confirmation()**.

```
                            {
  if(lcd_port == NULL) {
    printf("LCD NULL!");
    return false;
  }
  return true;
}
```

7.65.1.3 lcd_clear()

`void lcd_clear ( )`

Clears the lcd.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **47** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

Referenced by **display_menu()**, and **init_main_lcd()**.

```
                   {
  lcd_assert();
  lcdClear(lcd_port);
}
```

7.65.1.4 lcd_get_pressed_buttons()

`lcd_buttons lcd_get_pressed_buttons ( )`

Returns the pressed buttons.

Returns

a struct containing the states of all three buttons.

Author

Chris Jerrett

Date

9/9/2017

See also

lcd_buttons (p. 15)

Definition at line **28** of file **lcd.c**.

References **lcd_assert()**, **lcd_port**, **lcd_buttons::left**, **lcd_buttons::middle**, **PRESSED**, **RELEASED**, and **lcd_buttons::right**.

Referenced by **display_menu()**, and **promt_confirmation()**.

```
                                     {
  lcd_assert();
  unsigned int btn_binary = lcdReadButtons(lcd_port);
  bool left = btn_binary & 0x1;//0001
  bool middle = btn_binary & 0x2;//0010
  bool right = btn_binary & 0x4;//0100
  lcd_buttons btns;
  btns.left = left ? PRESSED : RELEASED;
  btns.middle = middle ? PRESSED : RELEASED;
  btns.right = right ? PRESSED : RELEASED;

  return btns;
}
```

7.65.1.5 lcd_print()

```
void lcd_print (
            unsigned int line,
            const char * str )
```

prints a string to a line on the lcd

Parameters

line	the line to print on
str	string to print

Author

Chris Jerrett

Date

9/9/2017

Definition at line **74** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

Referenced by **display_menu()**, and **promt_confirmation()**.

```
00074                                              {
00075   lcd_assert();
00076   lcdSetText(lcd_port, line, str);
00077 }
```

7.65.1.6 lcd_printf()

```
void lcd_printf (
            unsigned int line,
            const char * format_str,
             ... )
```

prints a formated string to a line on the lcd. Smilar to printf

Parameters

line	the line to print on
format_str	format string string to print

Author

Chris Jerrett

Date

9/9/2017

Definition at line **86** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

```
                                                                    {
  lcd_assert();
  lcdPrint(lcd_port, line, format_str);
}
```

7.65.1.7 lcd_set_backlight()

```
void lcd_set_backlight (
            bool state )
```

sets the backlight of the lcd

Parameters

state	a boolean representing the state of the backlight. true = on, false = off.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **97** of file **lcd.c**.

References **lcd_assert()**, and **lcd_port**.

```
                                  {
  lcd_assert();
  lcdSetBacklight(lcd_port, state);
}
```

7.65.1.8 promt_confirmation()

```
void promt_confirmation (
            const char * confirm_text )
```

Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Parameters

confirm_text	the text for the user to confirm.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **111** of file **lcd.c**.

References **lcd_assert()**, **lcd_get_pressed_buttons()**, **lcd_print()**, and **PRESSED**.

```
                                             {
  lcd_assert();
  lcd_print(1, confirm_text);
  while(lcd_get_pressed_buttons().middle != PRESSED){
    delay(200);
  }
}
```

7.65.2 Variable Documentation

7.65.2.1 lcd_port

```
FILE* lcd_port = NULL  [static]
```

The port of the initialized lcd

Definition at line **4** of file **lcd.c**.

Referenced by **init_main_lcd()**, **lcd_assert()**, **lcd_clear()**, **lcd_get_pressed_buttons()**, **lcd_print()**, **lcd_↩ printf()**, and **lcd_set_backlight()**.

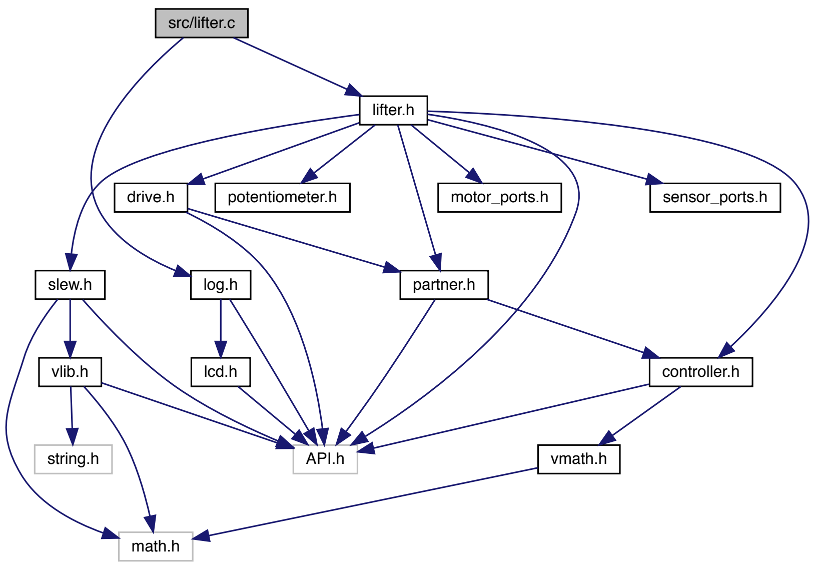
7.66 lcd.c

```
#include "lcd.h"

static FILE *lcd_port = NULL;

 static bool lcd_assert() {
  if(lcd_port == NULL) {
    printf("LCD NULL!");
    return false;
  }
  return true;
}

lcd_buttons lcd_get_pressed_buttons(){
  lcd_assert();
  unsigned int btn_binary = lcdReadButtons(lcd_port);
  bool left = btn_binary & 0x1;//0001
  bool middle = btn_binary & 0x2;//0010
  bool right = btn_binary & 0x4;//0100
  lcd_buttons btns;
  btns.left = left ? PRESSED : RELEASED;
  btns.middle = middle ? PRESSED : RELEASED;
  btns.right = right ? PRESSED : RELEASED;

  return btns;
}

void lcd_clear() {
  lcd_assert();
  lcdClear(lcd_port);
}

void init_main_lcd(FILE *lcd) {
  lcd_port = lcd;
  lcdInit(lcd);
  lcd_clear();
}

void lcd_print(unsigned int line, const char *str) {
  lcd_assert();
  lcdSetText(lcd_port, line, str);
}

void lcd_printf(unsigned int line, const char *format_str, ...) {
  lcd_assert();
  lcdPrint(lcd_port, line, format_str);
}

void lcd_set_backlight(bool state) {
  lcd_assert();
  lcdSetBacklight(lcd_port, state);
}

void promt_confirmation(const char *confirm_text) {
  lcd_assert();
  lcd_print(1, confirm_text);
  while(lcd_get_pressed_buttons().middle != PRESSED){
    delay(200);
  }
}
```

7.67 src/lifter.c File Reference

```
#include "lifter.h"
#include "log.h"
```

Include dependency graph for lifter.c:



Functions

- double **getLifterHeight** ()

  Gets the height of the lifter in inches.
- int **getLifterTicks** ()

  Gets the value of the lifter pot.
- float **lifterPotentiometerToDegree** (int x)

  height of the lifter in degrees from 0 height
- void **lower_main_lifter** ()

  Lowers the main lifter.
- void **lower_secondary_lifter** ()

  Lowers the secondary lifter.
- void **raise_main_lifter** ()

  Raises the main lifter.
- void **raise_secondary_lifter** ()

  Raises the main lifter.
- void **set_lifter_pos** (int pos)

  Sets the lifter positions to the given value.
- void **set_main_lifter_motors** (const int v)

  Sets the main lifter motors to the given value.
- void **set_secondary_lifter_motors** (const int v)

  Sets the secondary lifter motors to the given value.
- void **update_lifter** ()

  Updates the lifter in teleop.

7.67.1 Function Documentation

7.67.1.1 getLifterHeight()

```
double getLifterHeight ( )
```

Gets the height of the lifter in inches.

Returns

the height of the lifter.

Author

Chris Jerrett

Date

9/17/2017

Definition at line **133** of file **lifter.c**.

References **getLifterTicks()**.

```
                              {
  unsigned int ticks = getLifterTicks();
  return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) + 0.0198 * ticks + 2.3033);
}
```

7.67.1.2 getLifterTicks()

```
int getLifterTicks ( )
```

Gets the value of the lifter pot.

Returns

the value of the pot.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **122** of file **lifter.c**.

References **LIFTER**.

Referenced by **getLifterHeight()**.

```
                        {
  return analogRead(LIFTER);
}
```

7.67.1.3 lifterPotentiometerToDegree()

```
float lifterPotentiometerToDegree (
              int x )
```

height of the lifter in degrees from 0 height

Parameters

x	the pot value

Returns

the positions in degrees

Author

Chris Jerrett

Date

10/13/2017

Definition at line **111** of file **lifter.c**.

References **DEG_MAX**, **INIT_ROTATION**, and **TICK_MAX**.

```
                                        {
  return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
}
```

7.67.1.4 lower_main_lifter()

```
void lower_main_lifter ( )
```

Lowers the main lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **53** of file **lifter.c**.

References **MIN_SPEED**, and **set_main_lifter_motors()**.

```
                          {
  set_main_lifter_motors(MIN_SPEED);
}
```

7.67.1.5 lower_secondary_lifter()

```
void lower_secondary_lifter ( )
```

Lowers the secondary lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **73** of file **lifter.c**.

References **MIN_SPEED**, and **set_secondary_lifter_motors()**.

```
00073                               {
00074   set_secondary_lifter_motors(MIN_SPEED);
00075 }
```

7.67.1.6 raise_main_lifter()

```
void raise_main_lifter ( )
```

Raises the main lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **43** of file **lifter.c**.

References **MAX_SPEED**, and **set_main_lifter_motors()**.

```
00043                          {
00044   set_main_lifter_motors(MAX_SPEED);
00045 }
```

7.67.1.7 raise_secondary_lifter()

```
void raise_secondary_lifter ( )
```

Raises the main lifter.

Author

Christian DeSimone

Date

9/12/2017

Definition at line **63** of file **lifter.c**.

References **MAX_SPEED**, and **set_secondary_lifter_motors()**.

```
00063                              {
00064   set_secondary_lifter_motors(MAX_SPEED);
00065 }
```

7.67.1.8 set_lifter_pos()

```
void set_lifter_pos (
             int pos )
```

Sets the lifter positions to the given value.

Parameters

pos	The height in inches

Author

Chris Jerrett

Date

9/12/2017

Definition at line **33** of file **lifter.c**.

```
00033                              {
00034
00035 }
```

7.67.1.9 set_main_lifter_motors()

```
void set_main_lifter_motors (
            const int v )
```

Sets the main lifter motors to the given value.

Parameters

v	value for the lifter motor. Between -128 - 127, any values outside are clamped.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **22** of file **lifter.c**.

References **MOTOR_LIFT**, and **set_motor_immediate()**.

Referenced by **lower_main_lifter()**, **raise_main_lifter()**, and **update_lifter()**.

```
00022                                       {
00023   set_motor_immediate(MOTOR_LIFT, v);
00024 }
```

7.67.1.10 set_secondary_lifter_motors()

```
void set_secondary_lifter_motors (
            const int v )
```

Sets the secondary lifter motors to the given value.

Parameters

v	value for the lifter motor. Between -128 - 127, any values outside are clamped.

Author

Chris Jerrett

Date

1/6/2018

Definition at line **11** of file **lifter.c**.

References **MOTOR_SECONDARY_LIFTER**, and **set_motor_immediate()**.

Referenced by **lower_secondary_lifter()**, **raise_secondary_lifter()**, and **update_lifter()**.

```
00011                                              {
00012   set_motor_immediate(MOTOR_SECONDARY_LIFTER, v);
00013 }
```

7.67.1.11 update_lifter()

```
void update_lifter ( )
```

Updates the lifter in teleop.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **83** of file **lifter.c**.

References **LIFTER_DOWN**, **LIFTER_UP**, **MAIN_LIFTER_MIN_HEIGHT**, **MAIN_LIFTER_POT**, **MAX_SPEED**, **MIN_SPEED**, **SECONDARY_LIFTER_MAX_HEIGHT**, **SECONDARY_LIFTER_POT_PORT**, **set_main_lifter←_motors()**, and **set_secondary_lifter_motors()**.

Referenced by **operatorControl()**.

```
00083                     {
00084   printf("%d\n", analogRead(1));
00085   if(joystickGetDigital(LIFTER_DOWN) && analogRead(MAIN_LIFTER_POT) <
      MAIN_LIFTER_MIN_HEIGHT) {
00086     set_secondary_lifter_motors(MAX_SPEED);
00087     set_main_lifter_motors(MIN_SPEED);
00088   } else if(joystickGetDigital(LIFTER_DOWN) && analogRead(MAIN_LIFTER_POT) >=
      MAIN_LIFTER_MIN_HEIGHT) {
00089     set_secondary_lifter_motors(MAX_SPEED);
00090     set_main_lifter_motors(0);
00091   } else if(joystickGetDigital(LIFTER_UP) && analogRead(
      SECONDARY_LIFTER_POT_PORT) < SECONDARY_LIFTER_MAX_HEIGHT) {
00092     set_secondary_lifter_motors(MIN_SPEED);
00093     set_main_lifter_motors(0);
00094   } else if(joystickGetDigital(LIFTER_UP) && analogRead(
      SECONDARY_LIFTER_POT_PORT) >= SECONDARY_LIFTER_MAX_HEIGHT) {
00095     set_main_lifter_motors(MAX_SPEED);
00096     set_secondary_lifter_motors(MIN_SPEED);
00097   } else {
00098     set_secondary_lifter_motors(0);
00099     set_main_lifter_motors(0);
00100   }
00101 }
```

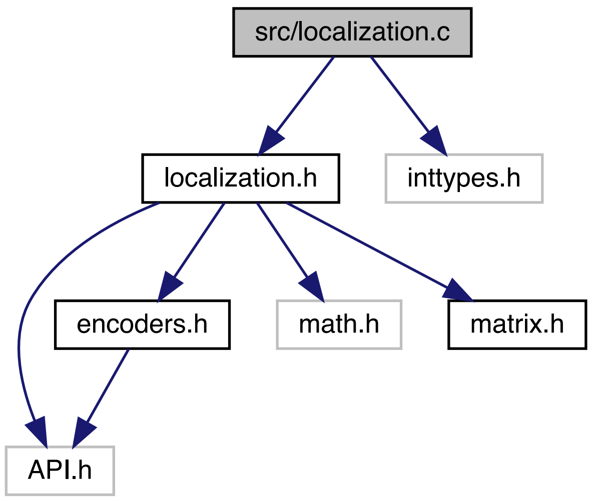
7.68 lifter.c

```
#include "lifter.h"
#include "log.h"

void set_secondary_lifter_motors(const int v) {
  set_motor_immediate(MOTOR_SECONDARY_LIFTER, v);
}

void set_main_lifter_motors(const int v) {
  set_motor_immediate(MOTOR_LIFT, v);
}

void set_lifter_pos(int pos) {

}

void raise_main_lifter(){
  set_main_lifter_motors(MAX_SPEED);
}

void lower_main_lifter(){
  set_main_lifter_motors(MIN_SPEED);
}

void raise_secondary_lifter(){
  set_secondary_lifter_motors(MAX_SPEED);
}

void lower_secondary_lifter(){
  set_secondary_lifter_motors(MIN_SPEED);
}

void update_lifter() {
  printf("%d\n", analogRead(1));
  if(joystickGetDigital(LIFTER_DOWN) && analogRead(MAIN_LIFTER_POT) <
MAIN_LIFTER_MIN_HEIGHT) {
    set_secondary_lifter_motors(MAX_SPEED);
    set_main_lifter_motors(MIN_SPEED);
  } else if(joystickGetDigital(LIFTER_DOWN) && analogRead(MAIN_LIFTER_POT) >=
MAIN_LIFTER_MIN_HEIGHT) {
    set_secondary_lifter_motors(MAX_SPEED);
    set_main_lifter_motors(0);
  } else if(joystickGetDigital(LIFTER_UP) && analogRead(
SECONDARY_LIFTER_POT_PORT) < SECONDARY_LIFTER_MAX_HEIGHT) {
    set_secondary_lifter_motors(MIN_SPEED);
    set_main_lifter_motors(0);
  } else if(joystickGetDigital(LIFTER_UP) && analogRead(
SECONDARY_LIFTER_POT_PORT) >= SECONDARY_LIFTER_MAX_HEIGHT) {
    set_main_lifter_motors(MAX_SPEED);
    set_secondary_lifter_motors(MIN_SPEED);
  } else {
    set_secondary_lifter_motors(0);
    set_main_lifter_motors(0);
  }
}

float lifterPotentiometerToDegree(int x){
  return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
}

int getLifterTicks() {
  return analogRead(LIFTER);
}

double getLifterHeight() {
  unsigned int ticks = getLifterTicks();
  return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) + 0.0198 * ticks + 2.3033);
}
```

7.69 src/localization.c File Reference

```
#include "localization.h"
#include <inttypes.h>
```

Include dependency graph for localization.c:



Data Structures

- struct **accelerometer_odometry**
- struct **encoder_odemtry**

Functions

- static struct **accelerometer_odometry** **calculate_accelerometer_odemetry** ()
- static double **calculate_angle** ()
- static double **calculate_gryo_anglular_velocity** ()
- struct **location** **get_position** ()

  Gets the current posituion of the robot.
- bool **init_localization** (const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int start←_theta)

  Starts the localization process.
- static double **integrate_gyro_w** (int new_w)
- void **update_position** ()

  Updates the position from the localization.

Variables

- static Gyro **g1**
- static int **last_call** = 0
- static TaskHandle **localization_task**
- **matrix** ∗ **state_matrix**

7.69.1 Function Documentation

7.69.1.1 calculate_accelerometer_odemetry()

```
static struct accelerometer_odometry calculate_accelerometer_odemetry ( ) [static]
```

Definition at line **60** of file **localization.c**.

References **last_call**.

Referenced by **update_position()**.

```
                                                                              {
  static double vel_acumm_x = 0;
  static double vel_acumm_y = 0;

  int32_t accel_x_rel = (int32_t) analogReadCalibratedHR(2);
  int32_t accel_y_rel = (int32_t) analogReadCalibratedHR(3);

  //Ignore atom format string errors
  printf("x: %+" PRId32 " y: %+" PRId32 "\n", accel_x_rel, accel_y_rel);

  double delta_time = ((millis() - last_call)/1000.0);
  //double accel_x_abs = (accel_x_rel *  cos(theta) + accel_y_rel * sin(theta)) * delta_time;
  //double accel_y_abs = (accel_y_rel *  cos(theta) + accel_x_rel * sin(theta)) * delta_time;

  //vel_acumm_x += accel_x_abs;
  //vel_acumm_y += accel_y_abs;

  //double new_x = x + vel_acumm_x * delta_time;
  //double new_y = y + vel_acumm_y * delta_time;

  struct accelerometer_odometry od;
  //od.x = new_x;
  //od.y = new_y;
  return od;
}
```

7.69.1.2 calculate_angle()

```
static double calculate_angle ( ) [static]
```

7.69.1.3 calculate_gryo_anglular_velocity()

```
static double calculate_gryo_anglular_velocity ( ) [static]
```

Definition at line **92** of file **localization.c**.

References **g1**, and **LOCALIZATION_UPDATE_FREQUENCY**.

```
                                                      {
  static int last_gyro = 0;
  int current = gyroGet(g1);
  // Calculate w (angluar velocity in degrees per second)
  double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY/1000.0);
  return w;
}
```

7.69.1.4 get_position()

```
struct location get_position ( )
```

Gets the current posituion of the robot.

Parameters

gyro1	The first gyro

Returns

the loacation of the robot as a struct.

Definition at line **31** of file **localization.c**.

```
                                  {
}
```

7.69.1.5 init_localization()

```
bool init_localization (
             const unsigned char gyro1,
             unsigned short multiplier,
             int start_x,
             int start_y,
             int start_theta )
```

Starts the localization process.

Author

Chris Jerrett

Parameters

gyro1	The first gyro The multiplier parameter can tune the gyro to adapt to specific sensors. The default value at this time is 196; higher values will increase the number of degrees reported for a fixed actual rotation, while lower values will decrease the number of degrees reported. If your robot is consistently turning too far, increase the multiplier, and if it is not turning far enough, decrease the multiplier.

Definition at line **100** of file **localization.c**.

References **g1**, **last_call**, **localization_task**, **LOCALIZATION_UPDATE_FREQUENCY**, **makeMatrix()**, and **update_position()**.

```
                  {
  g1 = gyroInit(gyro1, multiplier);
  //init state matrix

  //one dimensional vector with x, y, theta, acceleration in x and y
  state_matrix = makeMatrix(1, 5);
  localization_task = taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
  last_call = millis();
  return true;
}
```

7.69.1.6 integrate_gyro_w()

```
static double integrate_gyro_w (
            int new_w )  [static]
```

Definition at line **86** of file **localization.c**.

References **LOCALIZATION_UPDATE_FREQUENCY**, and **encoder_odemtry::theta**.

```
                                              {
  static double theta = 0;
  double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;
  theta += delta_theta;
}
```

7.69.1.7 update_position()

```
void update_position ( )
```

Updates the position from the localization.

Author

Chris Jerrett

Definition at line **40** of file **localization.c**.

References **calculate_accelerometer_odemetry()**, and **last_call**.

Referenced by **init_localization()**.

```
                        {
  //int curr_theta = calculate_angle();

  struct accelerometer_odometry oddem = calculate_accelerometer_odemetry();
  //printf("x: %d y: %d T: %d\n", a.x, a.y, 0);

  /*int l = 1;
  int vr = get_encoder_velocity(1);
  int vl = get_encoder_velocity(2);
  int theta_dot = (vr - vl) / l;
  int curr_theta = theta + theta_dot;
  double dt = LOCALIZATION_UPDATE_FREQUENCY;
  double v_tot = (vr+vl)/2.0;
  int x_curr = x - v_tot*dt*sin(curr_theta);
  int y_curr = y + v_tot*dt*cos(curr_theta);
  x = x_curr;
  y = y_curr;*/
  last_call = millis();
}
```

7.69.2 Variable Documentation

7.69.2.1 g1

```
Gyro g1 [static]
```

Definition at line **4** of file **localization.c**.

Referenced by **calculate_gryo_anglular_velocity()**, and **init_localization()**.

7.69.2.2 last_call

```
int last_call = 0 [static]
```

Definition at line **7** of file **localization.c**.

Referenced by **calculate_accelerometer_odemetry()**, **init_localization()**, and **update_position()**.

7.69.2.3 localization_task

```
TaskHandle localization_task [static]
```

Definition at line **5** of file **localization.c**.

Referenced by **init_localization()**.

7.69.2.4 state_matrix

```
matrix* state_matrix
```

Definition at line **9** of file **localization.c**.

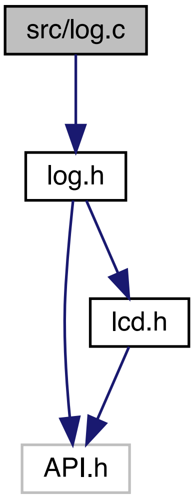
7.70 localization.c

```c
#include "localization.h"
#include <inttypes.h>

static Gyro g1;
static TaskHandle localization_task;

static int last_call = 0;

matrix *state_matrix;

struct encoder_odemtry {
  double x;
  double y;
  double theta;
};

struct accelerometer_odometry {
  double x;
  double y;
};

static double calculate_angle();
static struct accelerometer_odometry calculate_accelerometer_odemetry();

struct location get_position() {

}

void update_position() {
  //int curr_theta = calculate_angle();

  struct accelerometer_odometry oddem = calculate_accelerometer_odemetry();
  //printf("x: %d y: %d T: %d\n", a.x, a.y, 0);

  /*int l = 1;
  int vr = get_encoder_velocity(1);
  int vl = get_encoder_velocity(2);
  int theta_dot = (vr - vl) / l;
  int curr_theta = theta + theta_dot;
  double dt = LOCALIZATION_UPDATE_FREQUENCY;
  double v_tot = (vr+vl)/2.0;
  int x_curr = x - v_tot*dt*sin(curr_theta);
  int y_curr = y + v_tot*dt*cos(curr_theta);
  x = x_curr;
  y = y_curr;*/
  last_call = millis();
}

static struct accelerometer_odometry calculate_accelerometer_odemetry() {
  static double vel_acumm_x = 0;
  static double vel_acumm_y = 0;

  int32_t accel_x_rel = (int32_t) analogReadCalibratedHR(2);
  int32_t accel_y_rel = (int32_t) analogReadCalibratedHR(3);

  //Ignore atom format string errors
  printf("x: %+" PRId32 " y: %+" PRId32 "\n", accel_x_rel, accel_y_rel);

  double delta_time = ((millis() - last_call)/1000.0);
  //double accel_x_abs = (accel_x_rel *  cos(theta) + accel_y_rel * sin(theta)) * delta_time;
  //double accel_y_abs = (accel_y_rel *  cos(theta) + accel_x_rel * sin(theta)) * delta_time;

  //vel_acumm_x += accel_x_abs;
  //vel_acumm_y += accel_y_abs;

  //double new_x = x + vel_acumm_x * delta_time;
  //double new_y = y + vel_acumm_y * delta_time;

  struct accelerometer_odometry od;
  //od.x = new_x;
  //od.y = new_y;
  return od;
}

static double integrate_gyro_w(int new_w) {
  static double theta = 0;
  double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;
  theta += delta_theta;
}

static double calculate_gryo_anglular_velocity() {
  static int last_gyro = 0;
  int current = gyroGet(g1);
  // Calculate w (angluar velocity in degrees per second)
  double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY/1000.0);
  return w;
}

bool init_localization(const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int
start_theta)  {
  g1 = gyroInit(gyro1, multiplier);
  //init state matrix

  //one dimensional vector with x, y, theta, acceleration in x and y
  state_matrix = makeMatrix(1, 5);
  localization_task = taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
  last_call = millis();
  return true;
}
```

7.71 src/log.c File Reference

`#include "log.h"`

Include dependency graph for log.c:



Functions

- void **debug** (const char ∗debug_message)

  prints a info message
- void **error** (const char ∗error_message)

  prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE
- void **info** (const char ∗info_message)

  prints a info message
- void **init_error** (bool use_lcd, FILE ∗lcd)

  Initializes the error lcd system Only required if using lcd.
- static void **log_info** (const char ∗s, const char ∗mess)
- void **warning** (const char ∗warning_message)

  prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

Variables

- static FILE ∗ **log_lcd** = NULL
- unsigned int **log_level** = **INFO**

7.71.1 Function Documentation

7.71.1.1 debug()

```
void debug (
             const char * debug_message )
```

prints a info message

Only will print and display if log_level is greater than info

See also

log_level (p. 182)

Parameters

debug_message	the message

Definition at line **77** of file **log.c**.

References **ERROR**, and **log_level**.

Referenced by **set_motor_immediate()**, and **set_motor_slew()**.

```
                                     {
  if(log_level>ERROR) {
    printf("[INFO]: %s\n", debug_message);
  }
}
```

7.71.1.2 error()

```
void error (
             const char * error_message )
```

prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

log_level (p. 182)

Author

Chris Jerrett

Date

9/10/17

Parameters

error_message	the message

Definition at line **39** of file **log.c**.

References **log_info()**, **log_level**, and **NONE**.

Referenced by **create_menu()**.

```
                                         {
  if(log_level>NONE)
    log_info("ERROR", error_message);
}
```

7.71.1.3 info()

```
void info (
              const char * info_message )
```

prints a info message

Only will print and display if log_level is greater than ERROR

See also

log_level (p. 182)

Parameters

info_message	the message

Definition at line **64** of file **log.c**.

References **ERROR**, **log_info()**, and **log_level**.

Referenced by **init_slew()**, and **initialize()**.

```
                                       {
  if(log_level>ERROR) {
    log_info("INFO", info_message);
  }
}
```

7.71.1.4 init_error()

```
void init_error (
            bool use_lcd,
            FILE * lcd )
```

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

use_lcd	whether to use the lcd
lcd	the lcd

Definition at line **14** of file **log.c**.

References **log_lcd**.

Referenced by **initialize()**.

```
                                    {
  if(use_lcd) {
    lcdInit(lcd);
    log_lcd = lcd;
    lcdClear(log_lcd);
    printf("LCD Init\n");
  }
}
```

7.71.1.5 log_info()

```
static void log_info (
            const char * s,
            const char * mess )  [static]
```

Definition at line **23** of file **log.c**.

References **BOTTOM_ROW**, **log_lcd**, and **TOP_ROW**.

Referenced by **error()**, **info()**, and **warning()**.

```
                                                 {
  printf("[%s]: %s\n", s, mess);
  lcdSetBacklight(log_lcd, false);
  lcdClear(log_lcd);
  lcdPrint(log_lcd, TOP_ROW, s);
  lcdPrint(log_lcd, BOTTOM_ROW, mess);
}
```

7.71.1.6 warning()

```
void warning (
             const char * warning_message )
```

prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

See also

log_level (p. 182)

Author

Chris Jerrett

Date

9/10/17

Parameters

warning_message	the message

Definition at line **52** of file **log.c**.

References **log_info()**, **log_level**, and **WARNING**.

Referenced by **init_slew()**.

```
                                          {
  if(log_level>WARNING)
    log_info("WARNING", warning_message);
}
```

7.71.2 Variable Documentation

7.71.2.1 log_lcd

```
FILE* log_lcd = NULL  [static]
```

Definition at line **4** of file **log.c**.

Referenced by **init_error()**, and **log_info()**.

7.71.2.2 log_level

```
unsigned int log_level = INFO
```

Definition at line **3** of file **log.c**.

Referenced by **debug()**, **error()**, **info()**, and **warning()**.

7.72 log.c

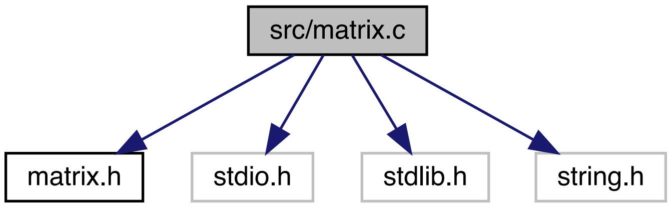
```
00001 #include "log.h"
00002
00003 unsigned int log_level = INFO;
00004 static FILE *log_lcd = NULL;
00005
00014 void init_error(bool use_lcd, FILE *lcd) {
00015   if(use_lcd) {
00016     lcdInit(lcd);
00017     log_lcd = lcd;
00018     lcdClear(log_lcd);
00019     printf("LCD Init\n");
00020   }
00021 }
00022
00023 static void log_info(const char *s, const char *mess) {
00024   printf("[%s]: %s\n", s, mess);
00025   lcdSetBacklight(log_lcd, false);
00026   lcdClear(log_lcd);
00027   lcdPrint(log_lcd, TOP_ROW, s);
00028   lcdPrint(log_lcd, BOTTOM_ROW, mess);
00029 }
00030
00039 void error(const char *error_message) {
00040   if(log_level>NONE)
00041     log_info("ERROR", error_message);
00042 }
00043
00052 void warning(const char *warning_message) {
00053   if(log_level>WARNING)
00054     log_info("WARNING", warning_message);
00055 }
00056
00064 void info(const char *info_message) {
00065   if(log_level>ERROR) {
00066     log_info("INFO", info_message);
00067   }
00068 }
00069
00077 void debug(const char *debug_message) {
00078   if(log_level>ERROR) {
00079     printf("[INFO]: %s\n", debug_message);
00080   }
00081 }
```

7.73 src/matrix.c File Reference

```
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
```

Include dependency graph for matrix.c:



Functions

- void **assert** (int assertion, const char ∗message)

  Asserts a condition is true.
- **matrix** ∗ **copyMatrix** (**matrix** ∗m)

  Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.
- **matrix** ∗ **covarianceMatrix** (**matrix** ∗m)

  returns the covariance of the matrix
- **matrix** ∗ **dotDiagonalMatrix** (**matrix** ∗a, **matrix** ∗b)

  performs a diagonial matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a 1 by a->height matrix of the cross product of each matrix along the diagonal.
- **matrix** ∗ **dotProductMatrix** (**matrix** ∗a, **matrix** ∗b)

  returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a a->height by b->height matrix of the cross product of each matrix.
- void **freeMatrix** (**matrix** ∗m)

  Frees the resources of a matrix.
- **matrix** ∗ **identityMatrix** (int n)

  Returns an identity matrix of size n by n.
- **matrix** ∗ **makeMatrix** (int width, int height)

  Makes a matrix with a width and height parameters.
- **matrix** ∗ **meanMatrix** (**matrix** ∗m)

  Given an "m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.
- **matrix** ∗ **multiplyMatrix** (**matrix** ∗a, **matrix** ∗b)

  Given a two matrices, returns the multiplication of the two.
- void **printMatrix** (**matrix** ∗m)

  Prints a matrix.
- void **rowSwap** (**matrix** ∗a, int p, int q)

  swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q, provided that p and q are less than or equal to the height of matrix A and p and q are different values.
- **matrix** ∗ **scaleMatrix** (**matrix** ∗m, double value)

  scales a matrix.
- double **traceMatrix** (**matrix** ∗m)

  Given an "m rows by n columns" matrix returns the sum.
- **matrix** ∗ **transposeMatrix** (**matrix** ∗m)

  returns the transpose matrix.

7.73.1 Function Documentation

7.73.1.1 assert()

```
void assert (
             int assertion,
             const char * message )
```

Asserts a condition is true.

If the assertion is non-zero (i.e. true), then it returns. If the assertion is zero (i.e. false), then it display the string and aborts the program. This is ment to act like Python's assert keyword.

Definition at line **14** of file **matrix.c**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **makeMatrix()**, **meanMatrix()**, **multiplyMatrix()**, and **rowSwap()**.

```
                                                 {
    if (assertion == 0) {
        fprintf(stderr, "%s\n", message);
        exit(1);
    }
}
```

7.73.1.2 copyMatrix()

```
matrix* copyMatrix (
             matrix * m )
```

Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.

Parameters

m	a pointer to the matrix

Returns

a copied matrix

Definition at line **52** of file **matrix.c**.

References **scaleMatrix()**.

```
                              {
    return scaleMatrix(m, 1);
}
```

7.73.1.3 covarianceMatrix()

```
matrix* covarianceMatrix (
            matrix * m )
```

returns the covariance of the matrix

Parameters

the	matrix

Returns

a matrix with n row and n columns, where each element represents covariance of 2 columns.

Definition at line **168** of file **matrix.c**.

References **assert()**, **_matrix::data**, **freeMatrix()**, **_matrix::height**, **makeMatrix()**, **meanMatrix()**, and **_↩
matrix::width**.

```
                                  {
    int i, j, k = 0;
    matrix* out;
    matrix* mean;
    double* ptrA;
    double* ptrB;
    double* ptrOut;

    assert(m->height > 1, "Height of matrix cannot be zero or one.");

    mean = meanMatrix(m);
    out = makeMatrix(m->width, m->width);
    ptrOut = out->data;

    for (i = 0; i < m->width; i++) {
        for (j = 0; j < m->width; j++) {
             ptrA = &m->data[i];
             ptrB = &m->data[j];
             *ptrOut = 0.0;
             for (k = 0; k < m->height; k++) {
                 *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
                 ptrA += m->width;
                 ptrB += m->width;
             }
             *ptrOut /= m->height - 1;
             ptrOut++;
        }
    }

    freeMatrix(mean);
    return out;
}
```

7.73.1.4 dotDiagonalMatrix()

```
matrix* dotDiagonalMatrix (
            matrix * a,
            matrix * b )
```

performs a diagonial matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and heights, this method returns a 1 by a->height matrix of the cross product of each matrix along the diagonal.

Dot product is essentially the sum-of-squares of two vectors.

If the second paramter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

a	the first matrix
b	the second matrix

Returns

the matrix result

Definition at line **385** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                               {
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;
    int i, j;

    if (b != NULL) {
        assert(a->width == b->width && a->height == b->height, "Matrices must be of the same
dimensionality.");
    }

    // Are we computing the sum of squares of the same matrix?
    if (a == b || b == NULL) {
        b = a; // May not appear safe, but we can do this without risk of losing b.
    }

    out = makeMatrix(1, a->height);
    ptrOut = out->data;
    ptrA = a->data;
    ptrB = b->data;

    for (i = 0; i < a->height; i++) {
        *ptrOut = 0;
        for (j = 0; j < a->width; j++) {
            *ptrOut += *ptrA * *ptrB;
            ptrA++;
            ptrB++;
        }
        ptrOut++;
    }

    return out;
}
```

7.73.1.5 dotProductMatrix()

```
matrix* dotProductMatrix (
            matrix * a,
            matrix * b )
```

returns the matrix dot product. Given a two matrices (or the same matrix twice) with identical widths and different heights, this method returns a a->height by b->height matrix of the cross product of each matrix.

Dot product is essentially the sum-of-squares of two vectors.

Also, if the second paramter is NULL, it is assumed that we are performing a cross product with itself.

Parameters

a	the first matrix
the	second matrix

Returns

the result of the dot product

Definition at line **333** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                            {
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;
    int i, j, k;

    if (b != NULL) {
        assert(a->width == b->width, "Matrices must be of the same dimensionality.");
    }

    // Are we computing the sum of squares of the same matrix?
    if (a == b || b == NULL) {
        b = a; // May not appear safe, but we can do this without risk of losing b.
    }

    out = makeMatrix(b->height, a->height);
    ptrOut = out->data;

    for (i = 0; i < a->height; i++) {
        ptrB = b->data;

        for (j = 0; j < b->height; j++) {
            ptrA = &a->data[ i * a->width ];

            *ptrOut = 0;
            for (k = 0; k < a->width; k++) {
                *ptrOut += *ptrA * *ptrB;
                ptrA++;
                ptrB++;
            }
            ptrOut++;
        }
    }

    return out;
}
```

7.73.1.6 freeMatrix()

```
void freeMatrix (
                matrix * m )
```

Frees the resources of a matrix.

Parameters

the	matrix to free

Definition at line **60** of file **matrix.c**.

References **_matrix::data**.

Referenced by **covarianceMatrix()**.

```
00060                             {
00061     if (m != NULL) {
00062         if (m->data != NULL) {
00063             free(m->data);
00064             m->data = NULL;
00065         }
00066         free(m);
00067     }
00068     return;
00069 }
```

7.73.1.7 identityMatrix()

```
matrix* identityMatrix (
             int n )
```

Returns an identity matrix of size n by n.

Parameters

n	the input matrix.

Returns

the identity matrix parameter.

Definition at line **94** of file **matrix.c**.

References **assert()**, **_matrix::data**, and **makeMatrix()**.

```
00094                                {
00095     int i;
00096     matrix *out;
00097     double* ptr;
00098 
00099     assert(n > 0, "Identity matrix must have value greater than zero.");
00100 
00101     out = makeMatrix(n, n);
00102     ptr = out->data;
00103     for (i = 0; i < n; i++) {
00104         *ptr = 1.0;
00105         ptr += n + 1;
00106     }
00107 
00108     return out;
00109 }
```

7.73.1.8 makeMatrix()

```
matrix* makeMatrix (
             int width,
             int height )
```

Makes a matrix with a width and height parameters.

Parameters

width	The width of the matrix
height	the height of the matrix

Returns

the new matrix

Definition at line **27** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, and **_matrix::width**.

Referenced by **covarianceMatrix()**, **dotDiagonalMatrix()**, **dotProductMatrix()**, **identityMatrix()**, **init_↩ localization()**, **meanMatrix()**, **multiplyMatrix()**, **scaleMatrix()**, and **transposeMatrix()**.

```
                                           {
    matrix* out;
    assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
    out = (matrix*) malloc(sizeof(matrix));

    assert(out != NULL, "Out of memory.");

    out->width = width;
    out->height = height;
    out->data = (double*) malloc(sizeof(double) * width * height);

    assert(out->data != NULL, "Out of memory.");

    memset(out->data, 0.0, width * height * sizeof(double));

    return out;
}
```

7.73.1.9 meanMatrix()

```
matrix* meanMatrix (
            matrix * m )
```

Given an "m rows by n columns" matrix, return a matrix where each element represents the mean of that full column. the matrix.

Returns

matrix with 1 row and n columns each element represents the mean of that full column.

Definition at line **142** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **covarianceMatrix()**.

```
                               {
    int i, j;
    matrix* out;

    assert(m->height > 0, "Height of matrix cannot be zero.");

    out = makeMatrix(m->width, 1);

    for (i = 0; i < m->width; i++) {
        double* ptr;
        out->data[i] = 0.0;
        ptr = &m->data[i];
        for (j = 0; j < m->height; j++) {
            out->data[i] += *ptr;
            ptr += out->width;
        }
        out->data[i] /= (double) m->height;
    }
    return out;
}
```

7.73.1.10 multiplyMatrix()

```
matrix* multiplyMatrix (
            matrix * a,
            matrix * b )
```

Given a two matrices, returns the multiplication of the two.

Parameters

a	the first matrix
b	the seconf matrix return the result of the multiplication

Definition at line **230** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                              {
    int i, j, k;
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;

    assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");

    out = makeMatrix(b->width, a->height);
    ptrOut = out->data;

    for (i = 0; i < a->height; i++) {

        for (j = 0; j < b->width; j++) {
            ptrA = &a->data[ i * a->width ];
            ptrB = &b->data[ j ];

            *ptrOut = 0;
            for (k = 0; k < a->width; k++) {
                *ptrOut += *ptrA * *ptrB;
                ptrA++;
                ptrB += b->width;
            }
            ptrOut++;
        }
    }

    return out;
}
```

7.73.1.11 printMatrix()

```
void printMatrix (
            matrix * m )
```

Prints a matrix.

Parameters

the	matrix

Definition at line **75** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
                            {
    int i, j;
    double* ptr = m->data;
    printf("%d %d\n", m->width, m->height);
    for (i = 0; i < m->height; i++) {
        for (j = 0; j < m->width; j++) {
            printf(" %9.6f", *(ptr++));
        }
        printf("\n");
    }
    return;
}
```

7.73.1.12 rowSwap()

```
void rowSwap (
               matrix * a,
              int p,
              int q )
```

swaps the rows of a matrix. This method changes the input matrix. Given a matrix, this algorithm will swap rows p and q, provided that p and q are less than or equal to the height of matrix A and p and q are different values.

Parameters

the	matrix to swap. This method changes the input matrix.
the	first row
the	second row

Definition at line **290** of file **matrix.c**.

References **assert()**, **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
                                         {
    int i;
    double temp;
    double* pRow;
    double* qRow;

    assert(a->height > 2, "Matrix must have at least two rows to swap.");
    assert(p < a->height && q < a->height, "Values p and q must be less than the height of the matrix.");

    // If p and q are equal, do nothing.
    if (p == q) {
        return;
    }

    pRow = a->data + (p * a->width);
    qRow = a->data + (q * a->width);

    // Swap!
    for (i = 0; i < a->width; i++) {
        temp = *pRow;
        *pRow = *qRow;
        *qRow = temp;
        pRow++;
        qRow++;
    }

    return;
}
```

7.73.1.13 scaleMatrix()

```
matrix* scaleMatrix (
            matrix * m,
            double value )
```

scales a matrix.

Parameters

m	the matrix to scale
the	value to scale by

Returns

a new matrix where each element in the input matrix is multiplied by the scalar value

Definition at line **268** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

Referenced by **copyMatrix()**.

```
                                             {
    int i, elements = m->width * m->height;
    matrix* out = makeMatrix(m->width, m->height);
    double* ptrM = m->data;
    double* ptrOut = out->data;

    for (i = 0; i < elements; i++) {
        *(ptrOut++) = *(ptrM++) * value;
    }

    return out;
}
```

7.73.1.14 traceMatrix()

```
double traceMatrix (
            matrix * m )
```

Given an "m rows by n columns" matrix returns the sum.

Given an "m rows by n columns" matrix.

Returns

the sum of the elements along the diagonal.

Definition at line **116** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, and **_matrix::width**.

```
                              {
    int i;
    int size;
    double* ptr = m->data;
    double sum = 0.0;

    if (m->height < m->width) {
        size = m->height;
    }
    else {
        size = m->width;
    }

    for (i = 0; i < size; i++) {
        sum += *ptr;
        ptr += m->width + 1;
    }

    return sum;
}
```

7.73.1.15 transposeMatrix()

matrix* transposeMatrix (
matrix * *m*)

returns the transpose matrix.

Parameters

the	matrix to transpose.

Returns

the transposed matrix.

Definition at line **206** of file **matrix.c**.

References **_matrix::data**, **_matrix::height**, **makeMatrix()**, and **_matrix::width**.

```
                                   {
    matrix* out = makeMatrix(m->height, m->width);
    double* ptrM = m->data;
    int i, j;

    for (i = 0; i < m->height; i++) {
        double* ptrOut;
        ptrOut = &out->data[i];
        for (j = 0; j < m->width; j++) {
            *ptrOut = *ptrM;
            ptrM++;
            ptrOut += out->width;
        }
    }

    return out;
}
```

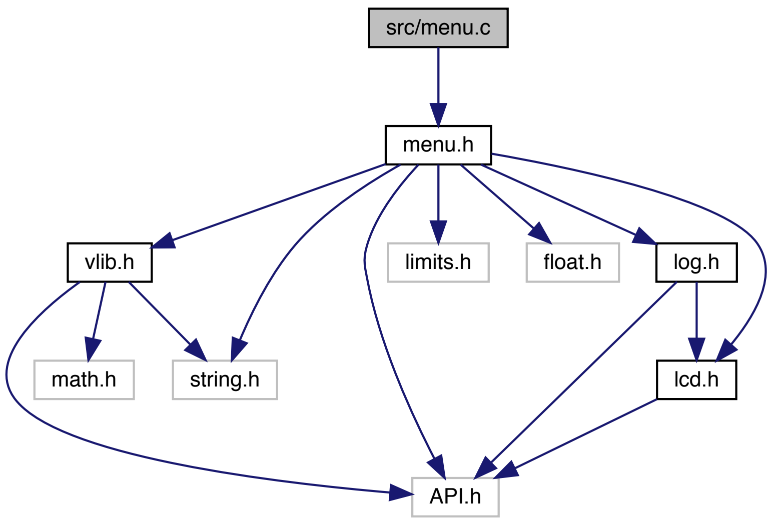
7.74 matrix.c

```
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void assert(int assertion, const char* message) {
    if (assertion == 0) {
        fprintf(stderr, "%s\n", message);
        exit(1);
    }
}

matrix* makeMatrix(int width, int height) {
    matrix* out;
    assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
    out = (matrix*) malloc(sizeof(matrix));

    assert(out != NULL, "Out of memory.");

    out->width = width;
    out->height = height;
    out->data = (double*) malloc(sizeof(double) * width * height);

    assert(out->data != NULL, "Out of memory.");

    memset(out->data, 0.0, width * height * sizeof(double));

    return out;
}

matrix* copyMatrix(matrix* m) {
    return scaleMatrix(m, 1);
}

void freeMatrix(matrix* m) {
    if (m != NULL) {
        if (m->data != NULL) {
            free(m->data);
            m->data = NULL;
        }
        free(m);
    }
    return;
}

void printMatrix(matrix* m) {
    int i, j;
    double* ptr = m->data;
    printf("%d %d\n", m->width, m->height);
    for (i = 0; i < m->height; i++) {
        for (j = 0; j < m->width; j++) {
            printf(" %9.6f", *(ptr++));
        }
        printf("\n");
    }
    return;
}

matrix* identityMatrix(int n) {
    int i;
    matrix *out;
    double* ptr;

    assert(n > 0, "Identity matrix must have value greater than zero.");

    out = makeMatrix(n, n);
    ptr = out->data;
    for (i = 0; i < n; i++) {
        *ptr = 1.0;
        ptr += n + 1;
    }

    return out;
}

double traceMatrix(matrix* m) {
    int i;
    int size;
    double* ptr = m->data;
    double sum = 0.0;

    if (m->height < m->width) {
        size = m->height;
    }
    else {
        size = m->width;
    }

    for (i = 0; i < size; i++) {
        sum += *ptr;
        ptr += m->width + 1;
    }

    return sum;
}

matrix* meanMatrix(matrix* m) {
    int i, j;
    matrix* out;

    assert(m->height > 0, "Height of matrix cannot be zero.");

    out = makeMatrix(m->width, 1);

    for (i = 0; i < m->width; i++) {
        double* ptr;
        out->data[i] = 0.0;
        ptr = &m->data[i];
        for (j = 0; j < m->height; j++) {
            out->data[i] += *ptr;
            ptr += out->width;
        }
        out->data[i] /= (double) m->height;
    }
    return out;
}

matrix* covarianceMatrix(matrix* m) {
    int i, j, k = 0;
    matrix* out;
    matrix* mean;
    double* ptrA;
    double* ptrB;
    double* ptrOut;

    assert(m->height > 1, "Height of matrix cannot be zero or one.");

    mean = meanMatrix(m);
    out = makeMatrix(m->width, m->width);
    ptrOut = out->data;

    for (i = 0; i < m->width; i++) {
        for (j = 0; j < m->width; j++) {
             ptrA = &m->data[i];
             ptrB = &m->data[j];
             *ptrOut = 0.0;
             for (k = 0; k < m->height; k++) {
                 *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
                 ptrA += m->width;
                 ptrB += m->width;
             }
             *ptrOut /= m->height - 1;
             ptrOut++;
        }
    }

    freeMatrix(mean);
    return out;
}

matrix* transposeMatrix(matrix* m) {
    matrix* out = makeMatrix(m->height, m->width);
    double* ptrM = m->data;
    int i, j;

    for (i = 0; i < m->height; i++) {
        double* ptrOut;
        ptrOut = &out->data[i];
        for (j = 0; j < m->width; j++) {
            *ptrOut = *ptrM;
            ptrM++;
            ptrOut += out->width;
        }
    }

    return out;
}

matrix* multiplyMatrix(matrix* a, matrix* b) {
    int i, j, k;
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;

    assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");

    out = makeMatrix(b->width, a->height);
    ptrOut = out->data;

    for (i = 0; i < a->height; i++) {

        for (j = 0; j < b->width; j++) {
            ptrA = &a->data[ i * a->width ];
            ptrB = &b->data[ j ];

            *ptrOut = 0;
            for (k = 0; k < a->width; k++) {
                *ptrOut += *ptrA * *ptrB;
                ptrA++;
                ptrB += b->width;
            }
            ptrOut++;
        }
    }

    return out;
}

matrix* scaleMatrix(matrix* m, double value) {
    int i, elements = m->width * m->height;
    matrix* out = makeMatrix(m->width, m->height);
    double* ptrM = m->data;
    double* ptrOut = out->data;

    for (i = 0; i < elements; i++) {
        *(ptrOut++) = *(ptrM++) * value;
    }

    return out;
}

void rowSwap(matrix* a, int p, int q) {
    int i;
    double temp;
    double* pRow;
    double* qRow;

    assert(a->height > 2, "Matrix must have at least two rows to swap.");
    assert(p < a->height && q < a->height, "Values p and q must be less than the height of the matrix.");

    // If p and q are equal, do nothing.
    if (p == q) {
        return;
    }

    pRow = a->data + (p * a->width);
    qRow = a->data + (q * a->width);

    // Swap!
    for (i = 0; i < a->width; i++) {
        temp = *pRow;
        *pRow = *qRow;
        *qRow = temp;
        pRow++;
        qRow++;
    }

    return;
}

matrix* dotProductMatrix(matrix* a, matrix* b) {
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;
    int i, j, k;

    if (b != NULL) {
        assert(a->width == b->width, "Matrices must be of the same dimensionality.");
    }

    // Are we computing the sum of squares of the same matrix?
    if (a == b || b == NULL) {
        b = a; // May not appear safe, but we can do this without risk of losing b.
    }

    out = makeMatrix(b->height, a->height);
    ptrOut = out->data;

    for (i = 0; i < a->height; i++) {
        ptrB = b->data;

        for (j = 0; j < b->height; j++) {
            ptrA = &a->data[ i * a->width ];

            *ptrOut = 0;
            for (k = 0; k < a->width; k++) {
                *ptrOut += *ptrA * *ptrB;
                ptrA++;
                ptrB++;
            }
            ptrOut++;
        }
    }

    return out;
}

matrix* dotDiagonalMatrix(matrix* a, matrix* b) {
    matrix* out;
    double* ptrOut;
    double* ptrA;
    double* ptrB;
    int i, j;

    if (b != NULL) {
        assert(a->width == b->width && a->height == b->height, "Matrices must be of the same
 dimensionality.");
    }

    // Are we computing the sum of squares of the same matrix?
    if (a == b || b == NULL) {
        b = a; // May not appear safe, but we can do this without risk of losing b.
    }

    out = makeMatrix(1, a->height);
    ptrOut = out->data;
    ptrA = a->data;
    ptrB = b->data;

    for (i = 0; i < a->height; i++) {
        *ptrOut = 0;
        for (j = 0; j < a->width; j++) {
            *ptrOut += *ptrA * *ptrB;
            ptrA++;
            ptrB++;
        }
        ptrOut++;
    }

    return out;
}
```

7.75 src/menu.c File Reference

```
#include "menu.h"
```

Include dependency graph for menu.c:



Functions

- static void **calculate_current_display** (char ∗rtn, **menu_t** ∗menu)
- static **menu_t** ∗ **create_menu** (enum **menu_type** type, const char ∗prompt)

  Static function that handles creation of menu. Menu must be freed or will cause memory leak
- void **denint_menu** (**menu_t** ∗menu)

  Destroys a menu Menu must be freed or will cause memory leak
- int **display_menu** (**menu_t** ∗menu)

  Displays a menu contex. Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.
- **menu_t** ∗ **init_menu_float** (enum **menu_type** type, float **min**, float **max**, float step, const char ∗prompt)

  Creates a menu context, but does not display. Menu must be freed or will cause memory leak!
- **menu_t** ∗ **init_menu_int** (enum **menu_type** type, int **min**, int **max**, int step, const char ∗prompt)

  Creates a menu context, but does not display. Menu must be freed or will cause memory leak
- **menu_t** ∗ **init_menu_var** (enum **menu_type** type, const char ∗prompt, int nums,...)

  Creates a menu context, but does not display. Menu must be freed or will cause memory leak

7.75.1 Function Documentation

7.75.1.1 calculate_current_display()

```
static void calculate_current_display (
            char * rtn,
             menu_t * menu )  [static]
```

Definition at line **103** of file **menu.c**.

References **menu_t::current**, **FLOAT_TYPE**, **ftoaa()**, **INT_TYPE**, **menu_t::length**, **max()**, **menu_t::max**, **menu_t::max_f**, **min()**, **menu_t::min**, **menu_t::min_f**, **menu_t::options**, **menu_t::step**, **menu_t::step_f**, **STRING_TYPE**, and **menu_t::type**.

Referenced by **display_menu()**.

```
                                                                 {
  if(menu->type == STRING_TYPE){
  int index = menu->current % menu->length;
  sprintf(rtn, "%s", menu->options[index]);
  printf("%s\n", rtn);
  return;
  }
  if(menu->type == INT_TYPE) {
    int step = (menu->step);
    int min = (menu->min);
    int max = (menu->max);
    int value = menu->current * step;
    if(value < min) {
      value = min;
      menu->current++;
    }
    if(value > max) {
      value = max;
      menu->current--;
    }
    sprintf(rtn, "%d", value);
  }
  if(menu->type == FLOAT_TYPE) {
    float step = (menu->step_f);
    float min = (menu->min_f);
    float max = (menu->max_f);
    float value = menu->current * step;
    value = value < min ? min : value;
    value = value > max ? max : value;


    ftoaa(value, rtn, 5);
  }
}
```

7.75.1.2 create_menu()

```
static  menu_t * create_menu (
            enum  menu_type type,
            const char * prompt )  [static]
```

Static function that handles creation of menu. *Menu must be freed or will cause memory leak*

Author

Chris Jerrett

Date

9/8/17

Definition at line **12** of file **menu.c**.

References **menu_t::current**, **error()**, **menu_t::max**, **menu_t::max_f**, **menu_t::min**, **menu_t::min_f**, **menu_t::prompt**, **menu_t::step**, **menu_t::step_f**, and **menu_t::type**.

Referenced by **init_menu_float()**, **init_menu_int()**, and **init_menu_var()**.

```
                                                                           {
  menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
  if (!menu) {
    error("Menu Malloc");
  }
  menu->type = type;
  // Add one for null terminator
  size_t strlength = strlen(prompt) + 1;
  menu->prompt = (char*) malloc(strlength * sizeof(char));
  memcpy(menu->prompt, prompt, strlength);
  menu->max = INT_MAX;
  menu->min = INT_MIN;
  menu->step = 1;
  menu->min_f = FLT_MIN;
  menu->max_f = FLT_MAX;
  menu->step_f = 1;
  menu->current = 0;

  return menu;
}
```

7.75.1.3 denint_menu()

```
void denint_menu (
                menu_t * menu )
```

Destroys a menu *Menu must be freed or will cause memory leak*

Parameters

menu	the menu to free

See also

menu

Author

Chris Jerrett

Date

9/8/17

Definition at line **186** of file **menu.c**.

References **menu_t::options**, and **menu_t::prompt**.

```
                                  {
  free(menu->prompt);
  if(menu->options != NULL) free(menu->options);
  free(menu);
}
```

7.75.1.4 display_menu()

```
int display_menu (
                **menu_t** * menu )
```

Displays a menu contex. *Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.*

Parameters

menu	the menu to display

See also

menu_type (p. 100)

Author

Chris Jerrett

Date

9/8/17

Definition at line **148** of file **menu.c**.

References **calculate_current_display()**, **menu_t::current**, **lcd_clear()**, **lcd_get_pressed_buttons()**, **lcd_print()**, **PRESSED**, **menu_t::prompt**, **RELEASED**, and **TOP_ROW**.

Referenced by **initialize()**.

```
                                  {
  lcd_print(TOP_ROW, menu->prompt);
  printf("printed prompt\n");
  //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
  char val[16];
  while(lcd_get_pressed_buttons().middle == RELEASED) {
    calculate_current_display(val, menu);

    if(lcd_get_pressed_buttons().right == PRESSED) {
      menu->current += 1;
    }
    if(lcd_get_pressed_buttons().left == PRESSED) {
      menu->current -= 1;
    }
    printf("%s\n", val);
    printf("%d\n",menu->current);
    lcd_print(2, val);
    delay(300);
  }
  printf("%d\n", menu->current);
  printf("return\n");
  lcd_clear();
  lcd_print(1, "Thk Cm Agn");
  lcd_print(2, val);
  delay(800);
  lcd_clear();
  return menu->current;
}
```

7.75.1.5 init_menu_float()

```
menu_t* init_menu_float (
            enum  menu_type type,
            float min,
            float max,
            float step,
            const char * prompt )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak!*

Parameters

type	the type of menu

See also

menu_type (p. 100)

Parameters

min	the minimum value
max	the maximum value
step	the step value
prompt	the prompt to display to user

Author

Chris Jerrett

Date

9/8/17

Definition at line **95** of file **menu.c**.

References **create_menu()**, **max()**, **menu_t::max_f**, **min()**, **menu_t::min_f**, and **menu_t::step_f**.

```
                                                                                     {
  menu_t* menu = create_menu(type, prompt);
  menu->min_f = min;
  menu->max_f = max;
  menu->step_f = step;
  return menu;
}
```

7.75.1.6 init_menu_int()

```
menu_t* init_menu_int (
            enum  menu_type type,
            int min,
            int max,
            int step,
            const char * prompt )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

type	the type of menu

See also

menu_type (p. 100)

Parameters

min	the minimum value
max	the maximum value
step	the step value
prompt	the prompt to display to user

Author

Chris Jerrett

Date

9/8/17

Definition at line **73** of file **menu.c**.

References **create_menu()**, **menu_t::current**, **max()**, **menu_t::max**, **min()**, **menu_t::min**, and **menu_t::step**.

```
                                                                                                       {
  menu_t* menu = create_menu(type, prompt);
  menu->min = min;
  menu->max = max;
  menu->step = step;
  menu->current = 0;
  return menu;
}
```

7.75.1.7 init_menu_var()

```
menu_t* init_menu_var (
            enum menu_type type,
            const char * prompt,
            int nums,
             ... )
```

Creates a menu context, but does not display. *Menu must be freed or will cause memory leak*

Parameters

type	the type of menu

See also

menu_type (p. 100)

Parameters

nums	the number of elements passed to function
prompt	the prompt to display to user
options	the options to display for user

Author

Chris Jerrett

Date

9/8/17

Definition at line **45** of file **menu.c**.

References **create_menu()**, **menu_t::length**, and **menu_t::options**.

Referenced by **initialize()**.

```
00045                                                                                 {
00046   menu_t* menu = create_menu(type, prompt);
00047   va_list ap;
00048   char **options_array = (char**)calloc(sizeof(char*), nums);
00049   va_start(ap, nums);
00050   for(int i = 0; i < nums; i++){
00051     options_array[i] = (char*) va_arg(ap, char*);
00052     printf("%s\n", options_array[i]);
00053   }
00054   va_end(ap);
00055   menu->options = options_array;
00056   menu->length = nums;
00057   return menu;
00058 }
```

7.76 menu.c

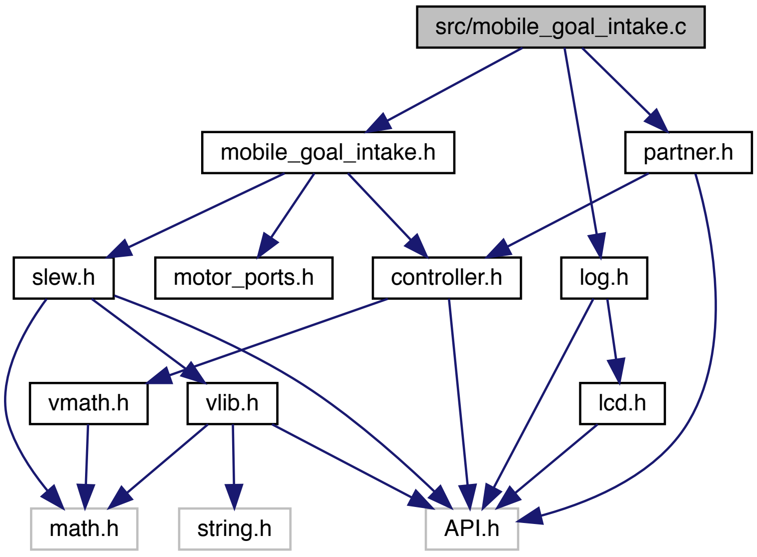
```
00001 #include "menu.h"
00002
00003 static menu_t* create_menu(enum menu_type type, const char *prompt);
00004 static void calculate_current_display(char* rtn, menu_t *menu);
00005
00012 static menu_t* create_menu(enum menu_type type, const char *prompt) {
00013   menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
00014   if (!menu) {
00015     error("Menu Malloc");
00016   }
00017   menu->type = type;
00018   // Add one for null terminator
00019   size_t strlength = strlen(prompt) + 1;
00020   menu->prompt = (char*) malloc(strlength * sizeof(char));
00021   memcpy(menu->prompt, prompt, strlength);
00022   menu->max = INT_MAX;
00023   menu->min = INT_MIN;
00024   menu->step = 1;
00025   menu->min_f = FLT_MIN;
00026   menu->max_f = FLT_MAX;
00027   menu->step_f = 1;
00028   menu->current = 0;
00029
00030   return menu;
```

```
}

menu_t* init_menu_var(enum menu_type type, const char *prompt, int nums,...){
  menu_t* menu = create_menu(type, prompt);
  va_list ap;
  char **options_array = (char**)calloc(sizeof(char*), nums);
  va_start(ap, nums);
  for(int i = 0; i < nums; i++){
    options_array[i] = (char*) va_arg(ap, char*);
    printf("%s\n", options_array[i]);
  }
  va_end(ap);
  menu->options = options_array;
  menu->length = nums;
  return menu;
}

menu_t* init_menu_int(enum menu_type type, int min, int max, int step, const char* prompt){
  menu_t* menu = create_menu(type, prompt);
  menu->min = min;
  menu->max = max;
  menu->step = step;
  menu->current = 0;
  return menu;
}

menu_t* init_menu_float(enum menu_type type, float min, float max, float step, const char* prompt){
  menu_t* menu = create_menu(type, prompt);
  menu->min_f = min;
  menu->max_f = max;
  menu->step_f = step;
  return menu;
}

static void calculate_current_display(char* rtn, menu_t *menu) {
  if(menu->type == STRING_TYPE){
  int index = menu->current % menu->length;
  sprintf(rtn, "%s", menu->options[index]);
  printf("%s\n", rtn);
  return;
  }
  if(menu->type == INT_TYPE) {
    int step = (menu->step);
    int min = (menu->min);
    int max = (menu->max);
    int value = menu->current * step;
    if(value < min) {
      value = min;
      menu->current++;
    }
    if(value > max) {
      value = max;
      menu->current--;
    }
    sprintf(rtn, "%d", value);
  }
  if(menu->type == FLOAT_TYPE) {
    float step = (menu->step_f);
    float min = (menu->min_f);
    float max = (menu->max_f);
    float value = menu->current * step;
    value = value < min ? min : value;
    value = value > max ? max : value;


    ftoaa(value, rtn, 5);
  }
}

int display_menu(menu_t *menu){
  lcd_print(TOP_ROW, menu->prompt);
  printf("printed prompt\n");
  //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
  char val[16];
  while(lcd_get_pressed_buttons().middle == RELEASED) {
    calculate_current_display(val, menu);

    if(lcd_get_pressed_buttons().right == PRESSED) {
      menu->current += 1;
    }
    if(lcd_get_pressed_buttons().left == PRESSED) {
      menu->current -= 1;
    }
    printf("%s\n", val);
    printf("%d\n",menu->current);
    lcd_print(2, val);
    delay(300);
  }
  printf("%d\n", menu->current);
  printf("return\n");
  lcd_clear();
  lcd_print(1, "Thk Cm Agn");
  lcd_print(2, val);
  delay(800);
  lcd_clear();
  return menu->current;
}

void denint_menu(menu_t *menu){
  free(menu->prompt);
  if(menu->options != NULL) free(menu->options);
  free(menu);
}
```

7.77 src/mobile_goal_intake.c File Reference

```
#include "mobile_goal_intake.h"
#include "partner.h"
#include "log.h"
```

Include dependency graph for mobile_goal_intake.c:



Functions

- static void **lower_intake** ()
- static void **raise_intake** ()
- static void **set_intake_motor** (int n)
- void **update_intake** ()

  updates the mobile goal intake in teleop.

7.77.1 Function Documentation

7.77.1.1 lower_intake()

```
static void lower_intake ( )  [static]
```

Definition at line **9** of file **mobile_goal_intake.c**.

References **set_intake_motor()**.

Referenced by **update_intake()**.

```
00009                          {
00010   set_intake_motor(-100);
00011 }
```

7.77.1.2 raise_intake()

```
static void raise_intake ( )  [static]
```

Definition at line **13** of file **mobile_goal_intake.c**.

References **set_intake_motor()**.

Referenced by **update_intake()**.

```
00013                          {
00014   set_intake_motor(100);
00015 }
```

7.77.1.3 set_intake_motor()

```
static void set_intake_motor (
             int n )  [static]
```

Definition at line **5** of file **mobile_goal_intake.c**.

References **INTAKE_MOTOR**, and **set_motor_immediate()**.

Referenced by **lower_intake()**, **raise_intake()**, and **update_intake()**.

```
00005                                  {
00006   set_motor_immediate(INTAKE_MOTOR, n);
00007 }
```

7.77.1.4 update_intake()

```
void update_intake ( )
```

updates the mobile goal intake in teleop.

Author

Chris Jerrett

Definition at line **20** of file **mobile_goal_intake.c**.

References **lower_intake()**, **MASTER**, **raise_intake()**, and **set_intake_motor()**.

Referenced by **operatorControl()**.

```
00020                         {
00021   if(joystickGetDigital(MASTER, 7, JOY_UP)) {
00022     raise_intake();
00023   }
00024   else if(joystickGetDigital(MASTER, 7, JOY_DOWN)){
00025     lower_intake();
00026   }
00027   else set_intake_motor(0);
00028 }
```

7.78 mobile_goal_intake.c

```
00001 #include "mobile_goal_intake.h"
00002 #include "partner.h"
00003 #include "log.h"
00004
00005 static void set_intake_motor(int n) {
00006   set_motor_immediate(INTAKE_MOTOR, n);
00007 }
00008
00009 static void lower_intake() {
00010   set_intake_motor(-100);
00011 }
00012
00013 static void raise_intake() {
00014   set_intake_motor(100);
00015 }
00016
00020 void update_intake() {
00021   if(joystickGetDigital(MASTER, 7, JOY_UP)) {
00022     raise_intake();
00023   }
00024   else if(joystickGetDigital(MASTER, 7, JOY_DOWN)){
00025     lower_intake();
00026   }
00027   else set_intake_motor(0);
00028 }
```

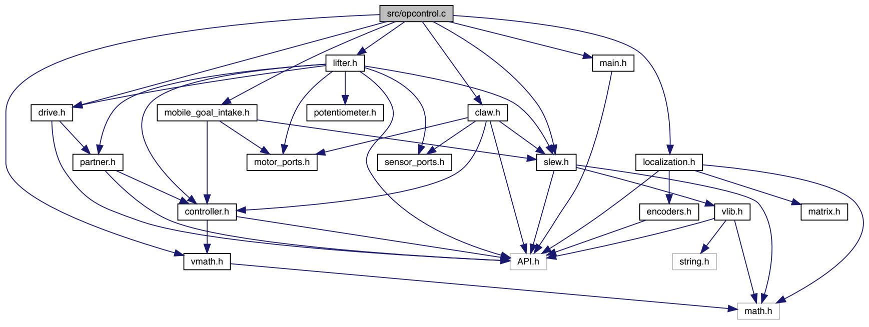
7.79 src/opcontrol.c File Reference

File for operator control code.

```
#include "main.h"
#include "slew.h"
#include "drive.h"
#include "lifter.h"
```

```
#include "localization.h"
#include "claw.h"
#include "mobile_goal_intake.h"
#include "vmath.h"
```

Include dependency graph for opcontrol.c:



Functions

- void **operatorControl** ()

7.79.1 Detailed Description

File for operator control code.

This file should contain the user **operatorControl()** (p. 210) function and any functions related to it.

Any copyright is dedicated to the Public Domain. `http://creativecommons.org/publicdomain/zero/1.↩ 0/`

PROS contains FreeRTOS (`http://www.freertos.org`) whose source code may be obtained from `http↩ ://sourceforge.net/projects/freertos/files/` or on request.

Definition in file **opcontrol.c**.

7.79.2 Function Documentation

7.79.2.1 operatorControl()

```
void operatorControl ( )
```

Runs the user operator control code. This function will be started in its own task with the default priority and stack size whenever the robot is enabled via the Field Management System or the VEX Competition Switch in the operator control mode. If the robot is disabled or communications is lost, the operator control task will be stopped by the kernel. Re-enabling the robot will restart the task, not resume it from where it left off.

If no VEX Competition Switch or Field Management system is plugged in, the VEX Cortex will run the operator control task. Be warned that this will also occur if the VEX Cortex is tethered directly to a computer via the USB A to A cable without any VEX Joystick attached.

Code running in this task can take almost any action, as the VEX Joystick is available and the scheduler is operational. However, proper use of delay() or taskDelayUntil() is highly recommended to give other tasks (including system tasks such as updating LCDs) time to run.

This task should never exit; its should end with some kind of infinite loop, even if empty.

Definition at line **42** of file **opcontrol.c**.

References **init_slew()**, **update_claw()**, **update_drive_motors()**, **update_intake()**, and **update_lifter()**.

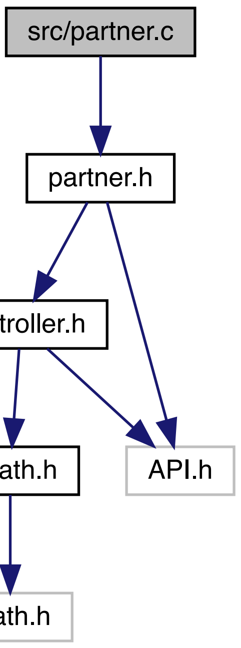
```
00042                         {
00043
00044     init_slew();
00045     delay(10);
00046     while (1) {
00047         update_claw();
00048         delay(5);
00049         update_intake();
00050         delay(5);
00051         update_lifter();
00052         delay(5);
00053         update_drive_motors();
00054         delay(25);
00055
00056     }
00057 }
```

7.80 opcontrol.c

```
00001
00013 #include "main.h"
00014 #include "slew.h"
00015 #include "drive.h"
00016
00017 #include "lifter.h"
00018 #include "localization.h"
00019 #include "claw.h"
00020 #include "mobile_goal_intake.h"
00021 #include "vmath.h"
00022 #include "lifter.h"
00023
00024
00042 void operatorControl() {
00043
00044     init_slew();
00045     delay(10);
00046     while (1) {
00047         update_claw();
00048         delay(5);
00049         update_intake();
00050         delay(5);
00051         update_lifter();
00052         delay(5);
00053         update_drive_motors();
00054         delay(25);
00055
00056     }
00057 }
```

7.81 src/partner.c File Reference

```
#include "partner.h"
```
Include dependency graph for partner.c:



Functions

- enum **CONTROLL_MODE get_mode** ()
- void **update_control** ()

    Updates the controller mode between Driver and Partner modes.

Variables

- static enum **CONTROLL_MODE mode** = **MAIN_CONTROLLER_MODE**

7.81.1 Function Documentation

7.81.1.1 get_mode()

```
enum CONTROLL_MODE get_mode ( )
```

Definition at line **5** of file **partner.c**.

References **mode**.

Referenced by **update_drive_motors()**.

```
                          {
  return mode;
}
```

7.81.1.2 update_control()

```
void update_control ( )
```

Updates the controller mode between Driver and Partner modes.

Author

Chris Jerrett

Definition at line **9** of file **partner.c**.

References **MAIN_CONTROLLER_MODE**, **mode**, **PARTNER**, and **PARTNER_CONTROLLER_MODE**.

```
                      {
  if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
    mode = MAIN_CONTROLLER_MODE;
  } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
    mode = PARTNER_CONTROLLER_MODE;
  }
}
```

7.81.2 Variable Documentation

7.81.2.1 mode

```
enum CONTROLL_MODE mode = MAIN_CONTROLLER_MODE [static]
```

Definition at line **3** of file **partner.c**.

Referenced by **get_mode()**, and **update_control()**.

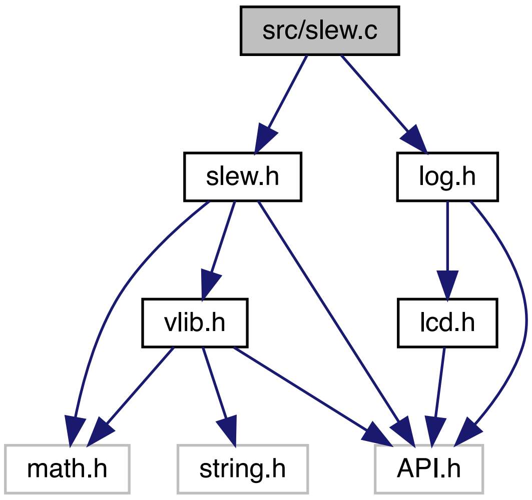
7.82 partner.c

```
#include "partner.h"

static enum CONTROLL_MODE mode = MAIN_CONTROLLER_MODE;

enum CONTROLL_MODE get_mode() {
  return mode;
}

void update_control() {
  if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
    mode = MAIN_CONTROLLER_MODE;
  } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
    mode = PARTNER_CONTROLLER_MODE;
  }
}
```

7.83 src/slew.c File Reference

```
#include "slew.h"
#include "log.h"
```

Include dependency graph for slew.c:



Functions

- void **deinitslew** ()
  Deinitializes the slew rate controller and frees memory.
- void **init_slew** ()
  Initializes the slew rate controller.
- void **set_motor_immediate** (int motor, int speed)
  Sets the motor speed ignoring the slew controller.
- void **set_motor_slew** (int motor, int speed)
  Sets motor speed wrapped inside the slew rate controller.
- void **updateMotors** ()
  Closes the distance between the desired motor value and the current motor value by half for each motor.

Variables

- static bool **initialized** = false
- static int **motors_curr_speeds** [10]
- static int **motors_set_speeds** [10]
- static TaskHandle **slew** = NULL
- static Mutex **speeds_mutex**

7.83.1 Function Documentation

7.83.1.1 deinitslew()

```
void deinitslew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line **58** of file **slew.c**.

References **initialized**, **motors_curr_speeds**, **motors_set_speeds**, and **slew**.

Referenced by **autonomous()**.

```
                 {
  taskDelete(slew);
  memset(motors_set_speeds, 0, sizeof(int) * 10);
  memset(motors_curr_speeds, 0, sizeof(int) * 10);
  initialized = false;
}
```

7.83.1.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

Definition at line **40** of file **slew.c**.

References **info()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, **slew**, **speeds_mutex**, **update↩ Motors()**, and **warning()**.

Referenced by **autonomous()**, **operatorControl()**, **set_motor_immediate()**, and **set_motor_slew()**.

```
00040                {
00041   if(initialized) {
00042     warning("Trying to init already init slew");
00043   }
00044   memset(motors_set_speeds, 0, sizeof(int) * 10);
00045   memset(motors_curr_speeds, 0, sizeof(int) * 10);
00046   motorStopAll();
00047   info("Did Init Slew");
00048   speeds_mutex = mutexCreate();
00049   slew = taskRunLoop(updateMotors, 100);
00050   initialized = true;
00051 }
```

7.83.1.3 set_motor_immediate()

```
void set_motor_immediate (
              int motor,
              int speed )
```

Sets the motor speed ignoring the slew controller.

Parameters

motor	the motor port to use
speed	the speed to use, between -127 and 127

Author

Chris Jerrett

Date

9/14/17

Definition at line **89** of file **slew.c**.

References **debug()**, **init_slew()**, **initialized**, **motors_curr_speeds**, **motors_set_speeds**, and **speeds_↩ mutex**.

Referenced by **close_claw()**, **open_claw()**, **set_claw_motor()**, **set_intake_motor()**, **set_main_lifter_motors()**, and **set_secondary_lifter_motors()**.

```
                                              {
  if(!initialized) {
    debug("Slew Not Initialized! Initializing");
    init_slew();
  }
  motorSet(motor, speed);
  mutexTake(speeds_mutex, 10);
  motors_curr_speeds[motor-1] = speed;
  motors_set_speeds[motor-1] = speed;
  mutexGive(speeds_mutex);
}
```

7.83.1.4 set_motor_slew()

```
void set_motor_slew (
              int motor,
              int speed )
```

Sets motor speed wrapped inside the slew rate controller.

Parameters

motor	the motor port to use
speed	the speed to use, between -127 and 127

Author

Chris Jerrett

Date

9/14/17

Definition at line **72** of file **slew.c**.

References **debug()**, **init_slew()**, **initialized**, **motors_set_speeds**, and **speeds_mutex**.

Referenced by **set_side_speed()**.

```
00072                                           {
00073   if(!initialized) {
00074     debug("Slew Not Initialized! Initializing");
00075     init_slew();
00076   }
00077   mutexTake(speeds_mutex, 10);
00078   motors_set_speeds[motor-1] = speed;
00079   mutexGive(speeds_mutex);
00080 }
```

7.83.1.5 updateMotors()

```
void updateMotors ( )
```

Closes the distance between the desired motor value and the current motor value by half for each motor.

Author

Chris Jerrett

Date

9/14/17

Definition at line **18** of file **slew.c**.

References **motors_curr_speeds**, **motors_set_speeds**, and **speeds_mutex**.

Referenced by **init_slew()**.

```
00018                     {
00019   //Take back half approach
00020   //Not linear but equal to setSpeed(1-(1/2)^x)
00021   for(unsigned int i = 0; i < 9; i++) {
00022     if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
00023     mutexTake(speeds_mutex, 10);
00024     int set_speed = (motors_set_speeds[i]);
00025     int curr_speed = motors_curr_speeds[i];
00026     mutexGive(speeds_mutex);
00027     int diff = set_speed - curr_speed;
00028     int offset = diff;
00029     int n = curr_speed + offset;
00030     motors_curr_speeds[i] = n;
00031     motorSet(i+1, n);
00032   }
00033 }
```

7.83.2 Variable Documentation

7.83.2.1 initialized

```
bool initialized = false  [static]
```

Definition at line **11** of file **slew.c**.

Referenced by **deinitslew()**, **init_slew()**, **set_motor_immediate()**, and **set_motor_slew()**.

7.83.2.2 motors_curr_speeds

```
int motors_curr_speeds[10]  [static]
```

Definition at line **7** of file **slew.c**.

Referenced by **deinitslew()**, **init_slew()**, **set_motor_immediate()**, and **updateMotors()**.

7.83.2.3 motors_set_speeds

```
int motors_set_speeds[10]  [static]
```

Definition at line **6** of file **slew.c**.

Referenced by **deinitslew()**, **init_slew()**, **set_motor_immediate()**, **set_motor_slew()**, and **updateMotors()**.

7.83.2.4 slew

```
TaskHandle slew = NULL  [static]
```

Definition at line **9** of file **slew.c**.

Referenced by **deinitslew()**, and **init_slew()**.

7.83.2.5 speeds_mutex

```
Mutex speeds_mutex  [static]
```

Definition at line **4** of file **slew.c**.

Referenced by **init_slew()**, **set_motor_immediate()**, **set_motor_slew()**, and **updateMotors()**.

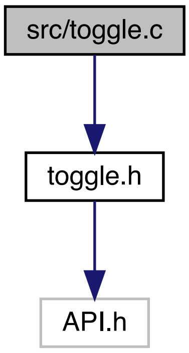
7.84 slew.c

```
#include "slew.h"
#include "log.h"

static Mutex speeds_mutex;

static int motors_set_speeds[10];
static int motors_curr_speeds[10];

static TaskHandle slew = NULL; //TaskHandle is of type void*

static bool initialized = false;

void updateMotors(){
  //Take back half approach
  //Not linear but equal to setSpeed(1-(1/2)^x)
  for(unsigned int i = 0; i < 9; i++) {
    if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
    mutexTake(speeds_mutex, 10);
    int set_speed = (motors_set_speeds[i]);
    int curr_speed = motors_curr_speeds[i];
    mutexGive(speeds_mutex);
    int diff = set_speed - curr_speed;
    int offset = diff;
    int n = curr_speed + offset;
    motors_curr_speeds[i] = n;
    motorSet(i+1, n);
  }
}

void init_slew(){
  if(initialized) {
    warning("Trying to init already init slew");
  }
  memset(motors_set_speeds, 0, sizeof(int) * 10);
  memset(motors_curr_speeds, 0, sizeof(int) * 10);
  motorStopAll();
  info("Did Init Slew");
  speeds_mutex = mutexCreate();
  slew = taskRunLoop(updateMotors, 100);
  initialized = true;
}

void deinitslew(){
  taskDelete(slew);
  memset(motors_set_speeds, 0, sizeof(int) * 10);
  memset(motors_curr_speeds, 0, sizeof(int) * 10);
  initialized = false;
}

void set_motor_slew(int motor, int speed){
  if(!initialized) {
    debug("Slew Not Initialized! Initializing");
    init_slew();
  }
  mutexTake(speeds_mutex, 10);
  motors_set_speeds[motor-1] = speed;
  mutexGive(speeds_mutex);
}

void set_motor_immediate(int motor, int speed) {
  if(!initialized) {
    debug("Slew Not Initialized! Initializing");
    init_slew();
  }
  motorSet(motor, speed);
  mutexTake(speeds_mutex, 10);
  motors_curr_speeds[motor-1] = speed;
  motors_set_speeds[motor-1] = speed;
  mutexGive(speeds_mutex);
}
```

7.85 src/toggle.c File Reference

```
#include "toggle.h"
```

Include dependency graph for toggle.c:



Functions

- bool **buttonGetState** (**button_t** button)

  Returns the current status of a button (pressed or not pressed)
- void **buttonInit** ()

  Initializes the buttons array.
- bool **buttonIsNewPress** (**button_t** button)

  Detects if button is a new press from most recent check by comparing previous value to current value.

Variables

- bool **buttonPressed** [27]

7.85.1 Function Documentation

7.85.1.1 buttonGetState()

```
bool buttonGetState (
              button_t  )
```

Returns the current status of a button (pressed or not pressed)

Parameters

button	The button to detect from the Buttons enumeration.

Returns

true (pressed) or false (not pressed)

Definition at line **25** of file **toggle.c**.

References **LCD_CENT**, **LCD_LEFT**, and **LCD_RIGHT**.

Referenced by **buttonIsNewPress()**.

```
                                    {
    bool currentButton = false;

    // Determine how to get the current button value (from what function) and where it
    // is, then get it.
    if (button < LCD_LEFT) {
        // button is a joystick button
        unsigned char joystick;
        unsigned char buttonGroup;
        unsigned char buttonLocation;

        button_t newButton;
        if (button <= 11) {
            // button is on joystick 1
            joystick = 1;
            newButton = button;
        }
        else {
            // button is on joystick 2
            joystick = 2;
            // shift button down to joystick 1 buttons in order to
            // detect which button on joystick is queried
            newButton = (button_t)(button - 12);
        }

        switch (newButton) {
        case 0:
            buttonGroup = 5;
            buttonLocation = JOY_DOWN;
            break;
        case 1:
            buttonGroup = 5;
            buttonLocation = JOY_UP;
            break;
        case 2:
            buttonGroup = 6;
            buttonLocation = JOY_DOWN;
            break;
        case 3:
            buttonGroup = 6;
            buttonLocation = JOY_UP;
            break;
        case 4:
            buttonGroup = 7;
            buttonLocation = JOY_UP;
            break;
        case 5:
            buttonGroup = 7;
            buttonLocation = JOY_LEFT;
            break;
        case 6:
            buttonGroup = 7;
            buttonLocation = JOY_RIGHT;
            break;
        case 7:
            buttonGroup = 7;
            buttonLocation = JOY_DOWN;
            break;
        case 8:
            buttonGroup = 8;
            buttonLocation = JOY_UP;
            break;
        case 9:
            buttonGroup = 8;
            buttonLocation = JOY_LEFT;
            break;
        case 10:
            buttonGroup = 8;
            buttonLocation = JOY_RIGHT;
            break;
        case 11:
            buttonGroup = 8;
            buttonLocation = JOY_DOWN;
            break;
        default:
            break;
        }
        currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
    }
    else {
        // button is on LCD
        if (button == LCD_LEFT)
            currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);

        if (button == LCD_CENT)
            currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);

        if (button == LCD_RIGHT)
            currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
    }
    return currentButton;
}
```

7.85.1.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons array.

Initializes the buttons.

Definition at line **20** of file **toggle.c**.

References **buttonPressed**.

```
                   {
    for (int i = 0; i < 27; i++)
        buttonPressed[i] = false;
}
```

7.85.1.3 buttonIsNewPress()

```
bool buttonIsNewPress (
                button_t button )
```

Detects if button is a new press from most recent check by comparing previous value to current value.

Parameters

button	The button to detect from the Buttons enumeration (see include/buttons.h).

Returns

true or false depending on if there was a change in button state.

Example code:

```
...
if(buttonIsNewPress(JOY1_8D))
    digitalWrite(1, !digitalRead(1));
...
```

Definition at line **135** of file **toggle.c**.

References **buttonGetState()**, and **buttonPressed**.

```
                                        {
    bool currentButton = buttonGetState(button);

    if (!currentButton) // buttons is not currently pressed
        buttonPressed[button] = false;

    if (currentButton && !buttonPressed[button]) {
        // button is currently pressed and was not detected as being pressed during last check
        buttonPressed[button] = true;
        return true;
    }
    else return false; // button is not pressed or was already detected
}
```

7.85.2 Variable Documentation

7.85.2.1 buttonPressed

```
bool buttonPressed[27]
```

Represents the array of "wasPressed" for all 27 available buttons.

Definition at line **15** of file **toggle.c**.

Referenced by **buttonInit()**, and **buttonIsNewPress()**.

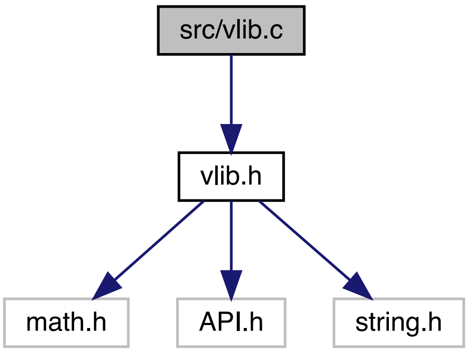
7.86 toggle.c

```

#include "toggle.h"

bool buttonPressed[27];

void buttonInit() {
    for (int i = 0; i < 27; i++)
        buttonPressed[i] = false;
}

bool buttonGetState(button_t button) {
    bool currentButton = false;

    // Determine how to get the current button value (from what function) and where it
    // is, then get it.
    if (button < LCD_LEFT) {
        // button is a joystick button
        unsigned char joystick;
        unsigned char buttonGroup;
        unsigned char buttonLocation;

        button_t newButton;
        if (button <= 11) {
            // button is on joystick 1
            joystick = 1;
            newButton = button;
        }
        else {
            // button is on joystick 2
            joystick = 2;
            // shift button down to joystick 1 buttons in order to
            // detect which button on joystick is queried
            newButton = (button_t)(button - 12);
        }

        switch (newButton) {
        case 0:
            buttonGroup = 5;
            buttonLocation = JOY_DOWN;
            break;
        case 1:
            buttonGroup = 5;
            buttonLocation = JOY_UP;
            break;
        case 2:
            buttonGroup = 6;
            buttonLocation = JOY_DOWN;
            break;
        case 3:
            buttonGroup = 6;
            buttonLocation = JOY_UP;
            break;
        case 4:
            buttonGroup = 7;
            buttonLocation = JOY_UP;
            break;
        case 5:
            buttonGroup = 7;
            buttonLocation = JOY_LEFT;
            break;
        case 6:
            buttonGroup = 7;
            buttonLocation = JOY_RIGHT;
            break;
        case 7:
            buttonGroup = 7;
            buttonLocation = JOY_DOWN;
            break;
        case 8:
            buttonGroup = 8;
            buttonLocation = JOY_UP;
            break;
        case 9:
            buttonGroup = 8;
            buttonLocation = JOY_LEFT;
            break;
        case 10:
            buttonGroup = 8;
            buttonLocation = JOY_RIGHT;
            break;
        case 11:
            buttonGroup = 8;
            buttonLocation = JOY_DOWN;
            break;
        default:
            break;
        }
        currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
    }
    else {
        // button is on LCD
        if (button == LCD_LEFT)
            currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);

        if (button == LCD_CENT)
            currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);

        if (button == LCD_RIGHT)
            currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
    }
    return currentButton;
}

bool buttonIsNewPress(button_t button) {
    bool currentButton = buttonGetState(button);

    if (!currentButton) // buttons is not currently pressed
        buttonPressed[button] = false;

    if (currentButton && !buttonPressed[button]) {
        // button is currently pressed and was not detected as being pressed during last check
        buttonPressed[button] = true;
        return true;
    }
    else return false; // button is not pressed or was already detected
}
```

7.87 src/vlib.c File Reference

`#include "vlib.h"`

Include dependency graph for vlib.c:



Functions

- void **ftoaa** (float a, char ∗buffer, int precision)

  converts a float to string.
- int **itoaa** (int a, char ∗buffer, int digits)

  converts a int to string.
- void **reverse** (char ∗str, int len)

  reverses a string 'str' of length 'len'

7.87.1 Function Documentation

7.87.1.1 ftoaa()

```
void ftoaa (
            float a,
            char * buffer,
            int precision )
```

converts a float to string.

Parameters

a	the float
buffer	the string the float will be written to.
precision	digits after the decimal to write

Author

Christian DeSimone

Date

9/26/2017

Definition at line **55** of file **vlib.c**.

References **itoaa()**.

Referenced by **calculate_current_display()**.

```
                                                {

  // Extract integer part
  int ipart = (int)a;

  // Extract floating part
  float fpart = a - (float)ipart;

  // convert integer part to string
  int i = itoaa(ipart, buffer, 0);

  // check for display option after point
  if(precision != 0) {
    buffer[i] = '.';  // add dot

    // Get the value of fraction part up to given num.
    // of points after dot. The third parameter is needed
    // to handle cases like 233.007
    fpart = fpart * pow(10, precision);

    itoaa((int)fpart, buffer + i + 1, precision);
  }
}
```

7.87.1.2 itoaa()

```
int itoaa (
            int a,
            char * buffer,
            int digits )
```

converts a int to string.

Parameters

a	the integer
buffer	the string the int will be written to.
digits	the number of digits to be written

Returns

the digits

Author

Chris Jerrett, Christian DeSimone

Date

9/9/2017

Definition at line **30** of file **vlib.c**.

References **reverse()**.

Referenced by **ftoaa()**.

```
                                             {
  int i = 0;
   while (a) {
       buffer[i++] = (a%10) + '0';
       a = a/10;
   }

   // If number of digits required is more, then
   // add 0s at the beginning
   while (i < digits)
       buffer[i++] = '0';

   reverse(buffer, i);
   buffer[i] = '\0';
   return i;
}
```

7.87.1.3 reverse()

```
void reverse (
               char * str,
               int len )
```

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

str	the string to reverse
len	the length

Definition at line **10** of file **vlib.c**.

Referenced by **itoaa()**.

```
00010                                    {
00011     int i=0, j=len-1, temp;
00012     while (i<j) {
00013         temp = str[i];
00014         str[i] = str[j];
00015         str[j] = temp;
00016         i++; j--;
00017     }
00018 }
```

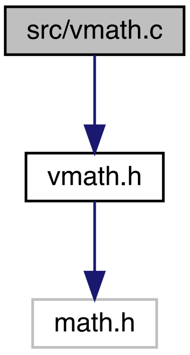
7.88 vlib.c

```
00001 #include "vlib.h"
00002 
00010 void reverse(char *str, int len) {
00011     int i=0, j=len-1, temp;
00012     while (i<j) {
00013         temp = str[i];
00014         str[i] = str[j];
00015         str[j] = temp;
00016         i++; j--;
00017     }
00018 }
00019 
00020 
00030 int itoaa(int a, char *buffer, int digits) {
00031   int i = 0;
00032    while (a) {
00033        buffer[i++] = (a%10) + '0';
00034        a = a/10;
00035    }
00036 
00037    // If number of digits required is more, then
00038    // add 0s at the beginning
00039    while (i < digits)
00040        buffer[i++] = '0';
00041 
00042    reverse(buffer, i);
00043    buffer[i] = '\0';
00044    return i;
00045 }
00046 
00055 void ftoaa(float a, char *buffer, int precision) {
00056 
00057   // Extract integer part
00058   int ipart = (int)a;
00059 
00060   // Extract floating part
00061   float fpart = a - (float)ipart;
00062 
00063   // convert integer part to string
00064   int i = itoaa(ipart, buffer, 0);
00065 
00066   // check for display option after point
00067   if(precision != 0) {
00068     buffer[i] = '.';  // add dot
00069 
00070     // Get the value of fraction part up to given num.
00071     // of points after dot. The third parameter is needed
00072     // to handle cases like 233.007
00073     fpart = fpart * pow(10, precision);
00074 
00075     itoaa((int)fpart, buffer + i + 1, precision);
00076   }
00077 }
```

7.89 src/vmath.c File Reference

```
#include "vmath.h"
```

Include dependency graph for vmath.c:



Functions

- struct **polar_cord** **cartesian_cord_to_polar** (struct **cord** cords)

  Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.
- struct **polar_cord** **cartesian_to_polar** (float x, float y)

  Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- int **max** (int a, int b)

  the min of two values
- int **min** (int a, int b)

  the min of two values
- double **sind** (double angle)

  sine of a angle in degrees

7.89.1 Function Documentation

7.89.1.1 cartesian_cord_to_polar()

```
struct  polar_cord cartesian_cord_to_polar (
            struct  cord cords )
```

Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

cords	the cartesian cords

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 22)
cord (p. 13)

Definition at line **55** of file **vmath.c**.

References **cartesian_to_polar()**.

```
                                                        {
  return cartesian_to_polar(cords.x, cords.y);
}
```

7.89.1.2 cartesian_to_polar()

```
struct polar_cord cartesian_to_polar (
            float x,
            float y )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

x	float value of the x cartesian coordinate.
y	float value of the y cartesian coordinate.

Returns

a struct containing the angle and magnitude.

See also

> **polar_cord** (p. 22)

Definition at line **14** of file **vmath.c**.

References **polar_cord::angle**, and **polar_cord::magnitue**.

Referenced by **cartesian_cord_to_polar()**.

```
                                                        {
  float degree = 0;
  double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));

  if(x < 0){
    degree += 180.0;
  }
  else if(x > 0 && y < 0){
    degree += 360.0;
  }

  if(x != 0 && y != 0){
    degree += atan((float)y / (float)x);
  }
  else if(x == 0 && y > 0){
    degree = 90.0;
  }
  else if(y == 0 && x < 0){
    degree = 180.0;
  }
  else if(x == 0 && y < 0){
    degree = 270.0;
  }

  struct polar_cord p;
  p.angle = degree;
  p.magnitue = magnitude;
  return p;
}
```

7.89.1.3 max()

```
int max (
              int a,
              int b )
```

the min of two values

Parameters

a	the first
b	the second

Returns

> the smaller of a and b

Definition at line **84** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
                         {
  if(a > b) return a;
  return b;
}
```

7.89.1.4 min()

```
int min (
            int a,
            int b )
```

the min of two values

Parameters

a	the first
b	the second

Returns

the smaller of a and b

Definition at line **73** of file **vmath.c**.

Referenced by **calculate_current_display()**, **init_menu_float()**, and **init_menu_int()**.

```
                         {
  if(a < b) return a;
  return b;
}
```

7.89.1.5 sind()

```
double sind (
            double angle )
```

sine of a angle in degrees

Definition at line **62** of file **vmath.c**.

References **M_PI**.

```
                              {
    double angleradians = angle * M_PI / 180.0f;
    return sin(angleradians);
}
```

7.90 vmath.c

```
#include "vmath.h"

struct polar_cord cartesian_to_polar(float x, float y) {
  float degree = 0;
  double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));

  if(x < 0){
    degree += 180.0;
  }
  else if(x > 0 && y < 0){
    degree += 360.0;
  }

  if(x != 0 && y != 0){
    degree += atan((float)y / (float)x);
  }
  else if(x == 0 && y > 0){
    degree = 90.0;
  }
  else if(y == 0 && x < 0){
    degree = 180.0;
  }
  else if(x == 0 && y < 0){
    degree = 270.0;
  }

  struct polar_cord p;
  p.angle = degree;
  p.magnitue = magnitude;
  return p;
}

struct polar_cord cartesian_cord_to_polar(struct cord cords) {
  return cartesian_to_polar(cords.x, cords.y);
}

double sind(double angle) {
    double angleradians = angle * M_PI / 180.0f;
    return sin(angleradians);
}

int min(int a, int b) {
  if(a < b) return a;
  return b;
}

int max(int a, int b) {
  if(a > b) return a;
  return b;
}
```

7.91 test_code/testMath.py File Reference

Namespaces

- **testMath**

Functions

- def **testMath.test** (l1, l2)

7.92 testMath.py

```
from math import *

def test(l1, l2):
    print(l1, l2)
    print("\n")
    theta = l1-l2
    x = ((l2)/(theta) + .5) - ((l2)/(theta) + .5) * cos(theta)
    y = ((l2)/(theta) + .5) * sin(theta)
    print(x)
    print(y)
    print(degrees(theta))
    print("\n")
    print("\n")

test(1.0, .5)
test(.5, 1.0)
test(2.0, .5)
test(.5, 2.0)
test(1.0, 3.5)
test(3.5, 1.0)
test(5, .3)
test(.3, 5)
test(1.0, 0)
```

7.93 testMath.py File Reference

Namespaces

- **testMath**

Functions

- def **testMath.test** (l1, l2)

7.94 testMath.py

```
from math import *

def test(l1, l2):
    print(l1, l2)
    print("\n")
    theta = l1-l2
    x = ((l2)/(theta) + .5) - ((l2)/(theta) + .5) * cos(theta)
    y = ((l2)/(theta) + .5) * sin(theta)
    print(x)
    print(y)
    print(degrees(theta))
    print("\n")
    print("\n")

test(1.0, .5)
test(.5, 1.0)
test(2.0, .5)
test(.5, 2.0)
test(1.0, 3.5)
test(3.5, 1.0)
test(5, .3)
test(.3, 5)
test(1.0, 0)
```