

Vex Team A

1.5.8

Generated by Doxygen 1.8.13

Mon Jan 15 2018 11:02:31

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>InTheZoneA</b>                                 | <b>1</b> |
| <b>2</b> | <b>Namespace Index</b>                            | <b>1</b> |
| 2.1      | Namespace List . . . . .                          | 1        |
| <b>3</b> | <b>Data Structure Index</b>                       | <b>2</b> |
| 3.1      | Data Structures . . . . .                         | 2        |
| <b>4</b> | <b>File Index</b>                                 | <b>2</b> |
| 4.1      | File List . . . . .                               | 2        |
| <b>5</b> | <b>Namespace Documentation</b>                    | <b>4</b> |
| 5.1      | testMath Namespace Reference . . . . .            | 5        |
| 5.1.1    | Function Documentation . . . . .                  | 5        |
| <b>6</b> | <b>Data Structure Documentation</b>               | <b>5</b> |
| 6.1      | _matrix Struct Reference . . . . .                | 5        |
| 6.1.1    | Detailed Description . . . . .                    | 5        |
| 6.1.2    | Field Documentation . . . . .                     | 5        |
| 6.2      | accelerometer_odometry Struct Reference . . . . . | 6        |
| 6.2.1    | Detailed Description . . . . .                    | 6        |
| 6.2.2    | Field Documentation . . . . .                     | 7        |
| 6.3      | cord Struct Reference . . . . .                   | 7        |
| 6.3.1    | Detailed Description . . . . .                    | 7        |
| 6.3.2    | Field Documentation . . . . .                     | 8        |
| 6.4      | encoder_odemtry Struct Reference . . . . .        | 8        |
| 6.4.1    | Detailed Description . . . . .                    | 8        |
| 6.4.2    | Field Documentation . . . . .                     | 8        |
| 6.5      | lcd_buttons Struct Reference . . . . .            | 9        |

|        |                                  |    |
|--------|----------------------------------|----|
| 6.5.1  | Detailed Description             | 9  |
| 6.5.2  | Field Documentation              | 10 |
| 6.6    | list_iterator_t Struct Reference | 10 |
| 6.6.1  | Detailed Description             | 11 |
| 6.6.2  | Field Documentation              | 11 |
| 6.7    | list_node Struct Reference       | 11 |
| 6.7.1  | Detailed Description             | 11 |
| 6.7.2  | Field Documentation              | 12 |
| 6.8    | list_t Struct Reference          | 12 |
| 6.8.1  | Detailed Description             | 13 |
| 6.8.2  | Field Documentation              | 13 |
| 6.9    | location Struct Reference        | 14 |
| 6.9.1  | Detailed Description             | 14 |
| 6.9.2  | Field Documentation              | 14 |
| 6.10   | menu_t Struct Reference          | 15 |
| 6.10.1 | Detailed Description             | 15 |
| 6.10.2 | Field Documentation              | 16 |
| 6.11   | polar_cord Struct Reference      | 20 |
| 6.11.1 | Detailed Description             | 20 |
| 6.11.2 | Field Documentation              | 21 |
| 6.12   | routine_t Struct Reference       | 21 |
| 6.12.1 | Detailed Description             | 21 |
| 6.12.2 | Field Documentation              | 22 |

|          |   |           |
|----------|---|-----------|
| <b>7</b> | <b>File Documentation</b>                     | <b>22</b> |
| 7.1      | include/auto.h File Reference . . . . .       | 22        |
| 7.1.1    | Detailed Description . . . . .                | 23        |
| 7.1.2    | Macro Definition Documentation . . . . .      | 23        |
| 7.2      | auto.h . . . . .                              | 26        |
| 7.3      | include/battery.h File Reference . . . . .    | 26        |
| 7.3.1    | Detailed Description . . . . .                | 27        |
| 7.3.2    | Macro Definition Documentation . . . . .      | 27        |
| 7.3.3    | Function Documentation . . . . .              | 28        |
| 7.4      | battery.h . . . . .                           | 29        |
| 7.5      | include/claw.h File Reference . . . . .       | 29        |
| 7.5.1    | Detailed Description . . . . .                | 30        |
| 7.5.2    | Macro Definition Documentation . . . . .      | 30        |
| 7.5.3    | Enumeration Type Documentation . . . . .      | 32        |
| 7.5.4    | Function Documentation . . . . .              | 33        |
| 7.6      | claw.h . . . . .                              | 34        |
| 7.7      | include/controller.h File Reference . . . . . | 35        |
| 7.7.1    | Detailed Description . . . . .                | 36        |
| 7.7.2    | Macro Definition Documentation . . . . .      | 36        |
| 7.7.3    | Enumeration Type Documentation . . . . .      | 38        |
| 7.7.4    | Function Documentation . . . . .              | 40        |
| 7.8      | controller.h . . . . .                        | 40        |
| 7.9      | include/drive.h File Reference . . . . .      | 41        |
| 7.9.1    | Detailed Description . . . . .                | 42        |
| 7.9.2    | Macro Definition Documentation . . . . .      | 42        |
| 7.9.3    | Typedef Documentation . . . . .               | 42        |
| 7.9.4    | Enumeration Type Documentation . . . . .      | 43        |
| 7.9.5    | Function Documentation . . . . .              | 43        |

|        |   |    |
|--------|---|----|
| 7.10   | drive.h . . . . .                               | 45 |
| 7.11   | include/encoders.h File Reference . . . . .     | 45 |
| 7.11.1 | Detailed Description . . . . .                  | 46 |
| 7.11.2 | Macro Definition Documentation . . . . .        | 46 |
| 7.11.3 | Function Documentation . . . . .                | 47 |
| 7.12   | encoders.h . . . . .                            | 48 |
| 7.13   | include/gyro.h File Reference . . . . .         | 48 |
| 7.13.1 | Macro Definition Documentation . . . . .        | 49 |
| 7.13.2 | Function Documentation . . . . .                | 49 |
| 7.14   | gyro.h . . . . .                                | 50 |
| 7.15   | include/lcd.h File Reference . . . . .          | 50 |
| 7.15.1 | Detailed Description . . . . .                  | 51 |
| 7.15.2 | Macro Definition Documentation . . . . .        | 51 |
| 7.15.3 | Enumeration Type Documentation . . . . .        | 52 |
| 7.15.4 | Function Documentation . . . . .                | 52 |
| 7.16   | lcd.h . . . . .                                 | 56 |
| 7.17   | include/lifter.h File Reference . . . . .       | 56 |
| 7.17.1 | Detailed Description . . . . .                  | 58 |
| 7.17.2 | Macro Definition Documentation . . . . .        | 58 |
| 7.17.3 | Function Documentation . . . . .                | 63 |
| 7.18   | lifter.h . . . . .                              | 68 |
| 7.19   | include/list.h File Reference . . . . .         | 69 |
| 7.19.1 | Macro Definition Documentation . . . . .        | 69 |
| 7.19.2 | Typedef Documentation . . . . .                 | 70 |
| 7.19.3 | Enumeration Type Documentation . . . . .        | 70 |
| 7.19.4 | Function Documentation . . . . .                | 70 |
| 7.20   | list.h . . . . .                                | 74 |
| 7.21   | include/localization.h File Reference . . . . . | 76 |

|  |     |
|--|-----|
| 7.21.1 Detailed Description . . . . .                      | 76  |
| 7.21.2 Macro Definition Documentation . . . . .            | 76  |
| 7.21.3 Function Documentation . . . . .                    | 77  |
| 7.22 localization.h . . . . .                              | 79  |
| 7.23 include/log.h File Reference . . . . .                | 79  |
| 7.23.1 Detailed Description . . . . .                      | 80  |
| 7.23.2 Macro Definition Documentation . . . . .            | 80  |
| 7.23.3 Function Documentation . . . . .                    | 82  |
| 7.24 log.h . . . . .                                       | 85  |
| 7.25 include/main.h File Reference . . . . .               | 85  |
| 7.25.1 Detailed Description . . . . .                      | 86  |
| 7.25.2 Function Documentation . . . . .                    | 86  |
| 7.26 main.h . . . . .                                      | 89  |
| 7.27 include/matrix.h File Reference . . . . .             | 89  |
| 7.27.1 Detailed Description . . . . .                      | 90  |
| 7.27.2 Typedef Documentation . . . . .                     | 91  |
| 7.27.3 Function Documentation . . . . .                    | 91  |
| 7.28 matrix.h . . . . .                                    | 98  |
| 7.29 include/menu.h File Reference . . . . .               | 98  |
| 7.29.1 Detailed Description . . . . .                      | 99  |
| 7.29.2 Typedef Documentation . . . . .                     | 99  |
| 7.29.3 Enumeration Type Documentation . . . . .            | 100 |
| 7.29.4 Function Documentation . . . . .                    | 101 |
| 7.30 menu.h . . . . .                                      | 105 |
| 7.31 include/mobile_goal_intake.h File Reference . . . . . | 105 |
| 7.31.1 Function Documentation . . . . .                    | 106 |
| 7.32 mobile_goal_intake.h . . . . .                        | 107 |
| 7.33 include/motor_ports.h File Reference . . . . .        | 107 |

|   |     |
|---|-----|
| 7.33.1 Detailed Description . . . . .                 | 108 |
| 7.33.2 Macro Definition Documentation . . . . .       | 108 |
| 7.34 motor_ports.h . . . . .                          | 112 |
| 7.35 include/partner.h File Reference . . . . .       | 112 |
| 7.35.1 Enumeration Type Documentation . . . . .       | 112 |
| 7.35.2 Function Documentation . . . . .               | 113 |
| 7.36 partner.h . . . . .                              | 113 |
| 7.37 include/potentiometer.h File Reference . . . . . | 114 |
| 7.37.1 Macro Definition Documentation . . . . .       | 114 |
| 7.38 potentiometer.h . . . . .                        | 114 |
| 7.39 include/routines.h File Reference . . . . .      | 114 |
| 7.39.1 Typedef Documentation . . . . .                | 115 |
| 7.39.2 Function Documentation . . . . .               | 115 |
| 7.40 routines.h . . . . .                             | 116 |
| 7.41 include/sensor_ports.h File Reference . . . . .  | 116 |
| 7.41.1 Macro Definition Documentation . . . . .       | 117 |
| 7.42 sensor_ports.h . . . . .                         | 117 |
| 7.43 include/slew.h File Reference . . . . .          | 118 |
| 7.43.1 Detailed Description . . . . .                 | 118 |
| 7.43.2 Macro Definition Documentation . . . . .       | 119 |
| 7.43.3 Function Documentation . . . . .               | 120 |
| 7.44 slew.h . . . . .                                 | 122 |
| 7.45 include/toggle.h File Reference . . . . .        | 122 |
| 7.45.1 Function Documentation . . . . .               | 123 |
| 7.46 toggle.h . . . . .                               | 125 |
| 7.47 include/vlib.h File Reference . . . . .          | 125 |
| 7.47.1 Detailed Description . . . . .                 | 126 |
| 7.47.2 Function Documentation . . . . .               | 126 |

|        |   |     |
|--------|---|-----|
| 7.48   | vlib.h . . . . .                          | 128 |
| 7.49   | include/vmath.h File Reference . . . . .  | 129 |
| 7.49.1 | Detailed Description . . . . .            | 129 |
| 7.49.2 | Macro Definition Documentation . . . . .  | 130 |
| 7.49.3 | Function Documentation . . . . .          | 130 |
| 7.50   | vmath.h . . . . .                         | 133 |
| 7.51   | README.md File Reference . . . . .        | 133 |
| 7.52   | README.md . . . . .                       | 133 |
| 7.53   | src/auto.c File Reference . . . . .       | 133 |
| 7.53.1 | Detailed Description . . . . .            | 133 |
| 7.53.2 | Function Documentation . . . . .          | 134 |
| 7.54   | auto.c . . . . .                          | 134 |
| 7.55   | src/battery.c File Reference . . . . .    | 136 |
| 7.55.1 | Function Documentation . . . . .          | 136 |
| 7.56   | battery.c . . . . .                       | 137 |
| 7.57   | src/claw.c File Reference . . . . .       | 138 |
| 7.57.1 | Function Documentation . . . . .          | 138 |
| 7.57.2 | Variable Documentation . . . . .          | 140 |
| 7.58   | claw.c . . . . .                          | 140 |
| 7.59   | src/controller.c File Reference . . . . . | 140 |
| 7.59.1 | Function Documentation . . . . .          | 140 |
| 7.60   | controller.c . . . . .                    | 141 |
| 7.61   | src/drive.c File Reference . . . . .      | 141 |
| 7.61.1 | Function Documentation . . . . .          | 142 |
| 7.61.2 | Variable Documentation . . . . .          | 144 |
| 7.62   | drive.c . . . . .                         | 145 |
| 7.63   | src/encoders.c File Reference . . . . .   | 145 |
| 7.63.1 | Function Documentation . . . . .          | 146 |



|        |  |     |
|--------|--|-----|
| 7.64   | encoders.c . . . . .                         | 147 |
| 7.65   | src/gyro.c File Reference . . . . .          | 147 |
| 7.65.1 | Function Documentation . . . . .             | 148 |
| 7.65.2 | Variable Documentation . . . . .             | 148 |
| 7.66   | gyro.c . . . . .                             | 149 |
| 7.67   | src/init.c File Reference . . . . .          | 149 |
| 7.67.1 | Detailed Description . . . . .               | 149 |
| 7.67.2 | Function Documentation . . . . .             | 150 |
| 7.67.3 | Variable Documentation . . . . .             | 150 |
| 7.68   | init.c . . . . .                             | 151 |
| 7.69   | src/lcd.c File Reference . . . . .           | 151 |
| 7.69.1 | Function Documentation . . . . .             | 152 |
| 7.69.2 | Variable Documentation . . . . .             | 156 |
| 7.70   | lcd.c . . . . .                              | 157 |
| 7.71   | src/lifter.c File Reference . . . . .        | 158 |
| 7.71.1 | Function Documentation . . . . .             | 158 |
| 7.71.2 | Variable Documentation . . . . .             | 164 |
| 7.72   | lifter.c . . . . .                           | 165 |
| 7.73   | src/list.c File Reference . . . . .          | 167 |
| 7.73.1 | Function Documentation . . . . .             | 167 |
| 7.74   | list.c . . . . .                             | 170 |
| 7.75   | src/list_iterator.c File Reference . . . . . | 172 |
| 7.75.1 | Function Documentation . . . . .             | 173 |
| 7.76   | list_iterator.c . . . . .                    | 174 |
| 7.77   | src/list_node.c File Reference . . . . .     | 174 |
| 7.77.1 | Function Documentation . . . . .             | 175 |
| 7.78   | list_node.c . . . . .                        | 175 |
| 7.79   | src/localization.c File Reference . . . . .  | 175 |

|        |   |     |
|--------|---|-----|
| 7.79.1 | Macro Definition Documentation . . . . .          | 176 |
| 7.79.2 | Function Documentation . . . . .                  | 177 |
| 7.79.3 | Variable Documentation . . . . .                  | 180 |
| 7.80   | localization.c . . . . .                          | 181 |
| 7.81   | src/log.c File Reference . . . . .                | 182 |
| 7.81.1 | Function Documentation . . . . .                  | 183 |
| 7.81.2 | Variable Documentation . . . . .                  | 186 |
| 7.82   | log.c . . . . .                                   | 186 |
| 7.83   | src/matrix.c File Reference . . . . .             | 187 |
| 7.83.1 | Function Documentation . . . . .                  | 188 |
| 7.84   | matrix.c . . . . .                                | 195 |
| 7.85   | src/menu.c File Reference . . . . .               | 198 |
| 7.85.1 | Function Documentation . . . . .                  | 199 |
| 7.86   | menu.c . . . . .                                  | 204 |
| 7.87   | src/mobile_goal_intake.c File Reference . . . . . | 206 |
| 7.87.1 | Function Documentation . . . . .                  | 206 |
| 7.88   | mobile_goal_intake.c . . . . .                    | 207 |
| 7.89   | src/opcontrol.c File Reference . . . . .          | 208 |
| 7.89.1 | Detailed Description . . . . .                    | 208 |
| 7.89.2 | Function Documentation . . . . .                  | 208 |
| 7.90   | opcontrol.c . . . . .                             | 209 |
| 7.91   | src/partner.c File Reference . . . . .            | 209 |
| 7.91.1 | Function Documentation . . . . .                  | 210 |
| 7.91.2 | Variable Documentation . . . . .                  | 210 |
| 7.92   | partner.c . . . . .                               | 211 |
| 7.93   | src/routines.c File Reference . . . . .           | 211 |
| 7.93.1 | Function Documentation . . . . .                  | 211 |
| 7.93.2 | Variable Documentation . . . . .                  | 212 |

|   |     |
|---|-----|
| 7.94 routines.c . . . . .                           | 213 |
| 7.95 src/slew.c File Reference . . . . .            | 214 |
| 7.95.1 Function Documentation . . . . .             | 214 |
| 7.95.2 Variable Documentation . . . . .             | 217 |
| 7.96 slew.c . . . . .                               | 218 |
| 7.97 src/toggle.c File Reference . . . . .          | 218 |
| 7.97.1 Function Documentation . . . . .             | 219 |
| 7.97.2 Variable Documentation . . . . .             | 220 |
| 7.98 toggle.c . . . . .                             | 221 |
| 7.99 src/vlib.c File Reference . . . . .            | 222 |
| 7.99.1 Function Documentation . . . . .             | 222 |
| 7.100vlib.c . . . . .                               | 224 |
| 7.101src/vmath.c File Reference . . . . .           | 225 |
| 7.101.1 Function Documentation . . . . .            | 225 |
| 7.102vmath.c . . . . .                              | 228 |
| 7.103test_code/testMath.py File Reference . . . . . | 229 |
| 7.104testMath.py . . . . .                          | 229 |
| 7.105testMath.py File Reference . . . . .           | 229 |
| 7.106testMath.py . . . . .                          | 230 |

## 1 InTheZoneA

Team A code for In The Zone

## 2 Namespace Index

### 2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

**testMath**

**5**

## 3 Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

|   |           |
|---|-----------|
| <b>_matrix</b>  | <b>5</b>  |
| <b>accelerometer_odometry</b>   | <b>6</b>  |
| <b>cord</b><br>A struct that contains cartesian coordinates   | <b>7</b>  |
| <b>encoder_odemtry</b>  | <b>8</b>  |
| <b>lcd_buttons</b><br>State of the lcd buttons  | <b>9</b>  |
| <b>list_iterator_t</b>  | <b>10</b> |
| <b>list_node</b>  | <b>11</b> |
| <b>list_t</b>   | <b>12</b> |
| <b>location</b>   | <b>14</b> |
| <b>menu_t</b><br>Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via denint↔<br>_menu | <b>15</b> |
| <b>polar_cord</b><br>A struct that contains polar coordinates   | <b>20</b> |
| <b>routine_t</b>  | <b>21</b> |

## 4 File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

|  |            |
|--|------------|
| <b>testMath.py</b>   | <b>229</b> |
| <b>include/ auto.h</b><br>Autonomous declarations and macros                 | <b>22</b>  |
| <b>include/ battery.h</b><br>Battery management related functions            | <b>26</b>  |
| <b>include/ claw.h</b><br>Code for controlling the claw that grabs the cones | <b>29</b>  |

|   |     |
|---|-----|
| <b>include/ controller.h</b>  |     |
| Controller definitions, macros and functions to assist with using the vex controllers   | 35  |
| <b>include/ drive.h</b>   |     |
| Drive base definitions and enumerations   | 41  |
| <b>include/ encoders.h</b>  |     |
| Wrapper around encoder functions  | 45  |
| <b>include/ gyro.h</b>  | 48  |
| <b>include/ lcd.h</b>   |     |
| LCD wrapper functions and macros  | 50  |
| <b>include/ lifter.h</b>  |     |
| Declarations and macros for controlling and manipulating the lifter   | 56  |
| <b>include/ list.h</b>  | 69  |
| <b>include/ localization.h</b>  |     |
| Declarations and macros for determining the location of the robot. [WIP]  | 76  |
| <b>include/ log.h</b>   |     |
| Contains logging functions  | 79  |
| <b>include/ main.h</b>  |     |
| Header file for global functions  | 85  |
| 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 prevalent, be sure to use the function to reclaim some of that memory 89 |     |
| <b>include/ menu.h</b>  |     |
| Contains menu functionality and abstraction   | 98  |
| <b>include/ mobile_goal_intake.h</b>  | 105 |
| Macros for the different motors ports 107   |     |
| <b>include/ partner.h</b>   | 112 |
| <b>include/ potentiometer.h</b>   | 114 |
| <b>include/ routines.h</b>  | 114 |
| <b>include/ sensor_ports.h</b>  | 116 |
| <b>include/ slew.h</b>  |     |
| Contains the slew rate controller wrapper for the motors  | 118 |
| <b>include/ toggle.h</b>  | 122 |
| <b>include/ vlib.h</b>  |     |
| Contains misc helpful functions   | 125 |
| <b>include/ vmath.h</b>   |     |
| Vex Specific Math Functions, includes: Cartesian to polar coordinates   | 129 |

|                                  |     |
|----------------------------------|-----|
| <b>src/ auto.c</b>               |     |
| File for autonomous code         | 133 |
| <b>src/ battery.c</b>            | 136 |
| <b>src/ claw.c</b>               | 138 |
| <b>src/ controller.c</b>         | 140 |
| <b>src/ drive.c</b>              | 141 |
| <b>src/ encoders.c</b>           | 145 |
| <b>src/ gyro.c</b>               | 147 |
| <b>src/ init.c</b>               |     |
| File for initialization code     | 149 |
| <b>src/ lcd.c</b>                | 151 |
| <b>src/ lifter.c</b>             | 158 |
| <b>src/ list.c</b>               | 167 |
| <b>src/ list_iterator.c</b>      | 172 |
| <b>src/ list_node.c</b>          | 174 |
| <b>src/ localization.c</b>       | 175 |
| <b>src/ log.c</b>                | 182 |
| <b>src/ matrix.c</b>             | 187 |
| <b>src/ menu.c</b>               | 198 |
| <b>src/ mobile_goal_intake.c</b> | 206 |
| <b>src/ opcontrol.c</b>          |     |
| File for operator control code   | 208 |
| <b>src/ partner.c</b>            | 209 |
| <b>src/ routines.c</b>           | 211 |
| <b>src/ slew.c</b>               | 214 |
| <b>src/ toggle.c</b>             | 218 |
| <b>src/ vlib.c</b>               | 222 |
| <b>src/ vmath.c</b>              | 225 |
| <b>test_code/ testMath.py</b>    | 229 |

## 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**.

## 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 **16** of file **matrix.h**.

#### 6.1.2 Field Documentation

### 6.1.2.1 data

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

Definition at line 19 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 17 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 18 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 18 of file **localization.c**.



### 6.2.2 Field Documentation

#### 6.2.2.1 x

```
double accelerometer_odometry::x
```

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

#### 6.2.2.2 y

```
double accelerometer_odometry::y
```

Definition at line 20 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 **12** of file **`localization.c`**.

#### 6.4.2 Field Documentation

#### 6.4.2.1 `theta`

```
double encoder_odemtry::theta
```

Definition at line **15** of file **localization.c**.

Referenced by **calculate\_encoder\_odemetry()**, and **integrate\_gyro\_w()**.

#### 6.4.2.2 `x`

```
double encoder_odemtry::x
```

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

#### 6.4.2.3 `y`

```
double encoder_odemtry::y
```

Definition at line **14** 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 list\_iterator\_t Struct Reference

```
#include <list.h>
```

### Data Fields

- **list\_direction\_t** direction
- **list\_node\_t** \* next

### 6.6.1 Detailed Description

Definition at line 62 of file `list.h`.

### 6.6.2 Field Documentation

#### 6.6.2.1 direction

```
list_direction_t list_iterator_t::direction
```

Definition at line 64 of file `list.h`.

#### 6.6.2.2 next

```
list_node_t* list_iterator_t::next
```

Definition at line 63 of file `list.h`.

Referenced by `list_iterator_new_from_node()`.

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

- include/ `list.h`

## 6.7 list\_node Struct Reference

```
#include <list.h>
```

### Data Fields

- struct `list_node` \* **next**
- struct `list_node` \* **prev**
- void \* **val**

### 6.7.1 Detailed Description

Definition at line 40 of file `list.h`.

## 6.7.2 Field Documentation

### 6.7.2.1 next

```
struct list_node* list_node::next
```

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

Referenced by **list\_destroy()**, **list\_iterator\_next()**, **list\_lpop()**, **list\_lpush()**, **list\_node\_new()**, **list\_remove()**, **list\_rpop()**, and **list\_rpush()**.

### 6.7.2.2 prev

```
struct list_node* list_node::prev
```

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

Referenced by **list\_iterator\_next()**, **list\_lpop()**, **list\_lpush()**, **list\_node\_new()**, **list\_remove()**, **list\_rpop()**, and **list\_rpush()**.

### 6.7.2.3 val

```
void* list_node::val
```

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

Referenced by **list\_destroy()**, **list\_find()**, **list\_node\_new()**, **list\_remove()**, **register\_routine()**, and **routine\_task()**.

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

- include/ **list.h**

## 6.8 list\_t Struct Reference

```
#include <list.h>
```

### Data Fields

- void(\* **free**)(void \*val)
- **list\_node\_t** \* **head**
- unsigned int **len**
- int(\* **match**)(void \*a, void \*b)
- **list\_node\_t** \* **tail**

### 6.8.1 Detailed Description

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

### 6.8.2 Field Documentation

#### 6.8.2.1 free

```
void(* list_t::free) (void *val)
```

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

#### 6.8.2.2 head

```
list_node_t* list_t::head
```

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

Referenced by **list\_iterator\_new()**, and **list\_new()**.

#### 6.8.2.3 len

```
unsigned int list_t::len
```

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

#### 6.8.2.4 match

```
int(* list_t::match) (void *a, void *b)
```

Definition at line 55 of file **list.h**.

#### 6.8.2.5 tail

```
list_node_t* list_t::tail
```

Definition at line 52 of file **list.h**.

Referenced by **list\_iterator\_new()**.

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

- include/ **list.h**

## 6.9 location Struct Reference

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

### Data Fields

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

### 6.9.1 Detailed Description

Vector storing the cartesian cords and an angle

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

### 6.9.2 Field Documentation

#### 6.9.2.1 theta

```
int location::theta
```

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

#### 6.9.2.2 x

```
int location::x
```

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

#### 6.9.2.3 y

```
int location::y
```

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

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

- include/ **localization.h**



## 6.10 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.*

### 6.10.1 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. 98)
- menu\_t** (p. 15)
- create\_menu** (p. 200)
- init\_menu**
- display\_menu** (p. 201)
- menu\_type** (p. 100)
- denint\_menu** (p. 200)

Definition at line **66** of file **menu.h**.

## 6.10.2 Field Documentation

### 6.10.2.1 current

```
int menu_t::current
```

contains the current index of menu.

**Author**

Chris Jerrett

**Date**

9/8/17

Definition at line **142** of file **menu.h**.

Referenced by **calculate\_current\_display()**, **create\_menu()**, **display\_menu()**, and **init\_menu\_int()**.

### 6.10.2.2 length

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

contains the length of options char\*\*.

**Author**

Chris Jerrett

**Date**

9/8/17

Definition at line **86** of file **menu.h**.

Referenced by **calculate\_current\_display()**, and **init\_menu\_var()**.

### 6.10.2.3 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 **102** of file **menu.h**.

Referenced by **calculate\_current\_display()**, **create\_menu()**, and **init\_menu\_int()**.

### 6.10.2.4 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()**.

### 6.10.2.5 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 **94** of file **menu.h**.

Referenced by **calculate\_current\_display()**, **create\_menu()**, and **init\_menu\_int()**.

#### 6.10.2.6 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()**.

#### 6.10.2.7 options

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

contains the array of string options.

##### Author

Chris Jerrett

##### Date

9/8/17

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

Referenced by **calculate\_current\_display()**, **denint\_menu()**, and **init\_menu\_var()**.

#### 6.10.2.8 prompt

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

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

##### Author

Chris Jerrett

##### Date

9/8/17

Definition at line **150** of file **menu.h**.

Referenced by **create\_menu()**, **denint\_menu()**, and **display\_menu()**.

### 6.10.2.9 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()**.

### 6.10.2.10 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 136 of file **menu.h**.

Referenced by **calculate\_current\_display()**, **create\_menu()**, and **init\_menu\_float()**.

#### 6.10.2.11 type

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

contains the type of menu.

##### Author

Chris Jerrett

##### Date

9/8/17

Definition at line **72** 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**

### 6.11 polar\_cord Struct Reference

A struct that contains polar coordinates.

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

##### Data Fields

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

#### 6.11.1 Detailed Description

A struct that contains polar coordinates.

##### Date

9/9/2017

##### Author

Chris Jerrett

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

### 6.11.2 Field Documentation

#### 6.11.2.1 angle

```
float polar_cord::angle
```

the angle of the vector

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

Referenced by **cartesian\_to\_polar()**.

#### 6.11.2.2 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**

## 6.12 routine\_t Struct Reference

```
#include <routines.h>
```

### Data Fields

- **button\_t \* blocked\_buttons**
- **button\_t on\_button**
- **void(\* routine )()**

#### 6.12.1 Detailed Description

Definition at line **3** of file **routines.h**.

## 6.12.2 Field Documentation

### 6.12.2.1 blocked\_buttons

```
button_t* routine_t::blocked_buttons
```

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

Referenced by **register\_routine()**.

### 6.12.2.2 on\_button

```
button_t routine_t::on_button
```

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

Referenced by **register\_routine()**, and **routine\_task()**.

### 6.12.2.3 routine

```
void(* routine_t::routine) ()
```

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

Referenced by **register\_routine()**, and **routine\_task()**.

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

- include/ **routines.h**

## 7 File Documentation

### 7.1 include/auto.h File Reference

Autonomous declarations and macros.

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



## Macros

- **#define DEPLOY\_HEIGHT 500**  
*height at which rubber bands do a do*
- **#define FRONT\_LEFT\_IME 0**  
*Front left motor integrated motor encoder.*
- **#define GOAL\_HEIGHT 1325**  
*The height of the goal using potentiometer readings.*
- **#define HALF\_ROTATE M\_PI**
- **#define LOWEST\_HEIGHT 0**
- **#define MAX\_HEIGHT 3090**
- **#define MID\_LEFT\_DRIVE 1**  
*Middle left motor integrated motor encoder.*
- **#define MID\_RIGHT\_DRIVE 4**  
*Middle right motor integrated motor encoder.*
- **#define MOBILE\_GOAL\_DISTANCE 4000**
- **#define MOBILE\_GOAL\_HEIGHT 3090**
- **#define STOP\_ONE 500**  
*First Stop position for stationary autonomous.*
- **#define ZONE\_DISTANCE 1000**

### 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 DEPLOY\_HEIGHT

```
#define DEPLOY_HEIGHT 500
```

height at which rubber bands do a do

Definition at line **45** of file **auto.h**.

Referenced by **autonomous()**.

#### 7.1.2.2 FRONT\_LEFT\_IME

```
#define FRONT_LEFT_IME 0
```

Front left motor integrated motor encoder.

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

#### 7.1.2.3 GOAL\_HEIGHT

```
#define GOAL_HEIGHT 1325
```

The height of the goal using potentiometer readings.

Definition at line **40** of file **auto.h**.

#### 7.1.2.4 HALF\_ROTATE

```
#define HALF_ROTATE  M_PI
```

Definition at line **57** of file **auto.h**.

Referenced by **autonomous()**.

#### 7.1.2.5 LOWEST\_HEIGHT

```
#define LOWEST_HEIGHT 0
```

Definition at line **47** of file **auto.h**.

Referenced by **autonomous()**.

#### 7.1.2.6 MAX\_HEIGHT

```
#define MAX_HEIGHT 3090
```

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

Referenced by **autonomous()**.

#### 7.1.2.7 MID\_LEFT\_DRIVE

```
#define MID_LEFT_DRIVE 1
```

Middle left motor integrated motor encoder.

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

Referenced by **autonomous()**.

#### 7.1.2.8 MID\_RIGHT\_DRIVE

```
#define MID_RIGHT_DRIVE 4
```

Middle right motor integrated motor encoder.

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

Referenced by **autonomous()**.

#### 7.1.2.9 MOBILE\_GOAL\_DISTANCE

```
#define MOBILE_GOAL_DISTANCE 4000
```

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

Referenced by **autonomous()**.

#### 7.1.2.10 MOBILE\_GOAL\_HEIGHT

```
#define MOBILE_GOAL_HEIGHT 3090
```

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

Referenced by **autonomous()**.

#### 7.1.2.11 STOP\_ONE

```
#define STOP_ONE 500
```

First Stop position for stationary autonomous.

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

### 7.1.2.12 ZONE\_DISTANCE

```
#define ZONE_DISTANCE 1000
```

Definition at line **55** of file **auto.h**.

Referenced by **autonomous()**.

## 7.2 auto.h

```
00001
00007 #ifndef _AUTO_H_
00008 #define _AUTO_H_
00009
00010 #include "claw.h"
00011 #include "drive.h"
00012 #include "lifter.h"
00013 #include "sensor_ports.h"
00014 #include "mobile_goal_intake.h"
00015 #include "localization.h"
00016
00020 #define FRONT_LEFT_IME 0
00021
00025 #define MID_LEFT_DRIVE 1
00026
00030 #define MID_RIGHT_DRIVE 4
00031
00035 #define STOP_ONE 500
00036
00040 #define GOAL_HEIGHT 1325
00041
00045 #define DEPLOY_HEIGHT 500
00046
00047 #define LOWEST_HEIGHT 0
00048
00049 #define MOBILE_GOAL_HEIGHT 3090
00050
00051 #define MOBILE_GOAL_DISTANCE 4000
00052
00053 #define MAX_HEIGHT 3090
00054
00055 #define ZONE_DISTANCE 1000
00056
00057 #define HALF_ROTATE M_PI
00058
00059 #endif
```

## 7.3 include/battery.h File Reference

Battery management related functions.

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

### 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 **14** of file **battery.c**.

Referenced by **battery\_level\_acceptable()**.

#### 7.3.3.2 battery\_level\_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

##### See also

**MIN\_MAIN\_VOLTAGE** (p. 27)

**MIN\_BACKUP\_VOLTAGE** (p. 27)

##### Author

Chris Jerrett

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

References **backup\_battery\_voltage()**, **main\_battery\_voltage()**, **MIN\_BACKUP\_VOLTAGE**, and **MIN\_MAIN\_VOLTAGE**.

Referenced by **initialize()**.

## 7.3.3.3 main\_battery\_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

## Author

Chris Jerrett

Definition at line 8 of file **battery.c**.

Referenced by **battery\_level\_acceptable()**.

## 7.4 battery.h

```
00001
00007 #ifndef _BATTERY_H_
00008 #define _BATTERY_H_
00009
00010 #include <API.h>
00011
00015 #define MIN_MAIN_VOLTAGE 7.8
00016
00020 #define MIN_BACKUP_VOLTAGE 7.8
00021
00026 double main_battery_voltage();
00027
00032 double backup_battery_voltage();
00033
00041 bool battery_level_acceptable();
00042
00043 #endif
```

## 7.5 include/claw.h File Reference

Code for controlling the claw that grabs the cones.

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

## Macros

- **#define CLAW\_CLOSE MASTER, 5, 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, 5, 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, 5, 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, 5, 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**.

#### 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 44 of file **claw.c**.

References **CLAW\_MOTOR**, **MIN\_CLAW\_SPEED**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**.

##### 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 38 of file **claw.c**.

References **CLAW\_MOTOR**, **MAX\_CLAW\_SPEED**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**.

#### 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 32 of file **claw.c**.

References **CLAW\_MOTOR**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**, and **update\_claw()**.

#### 7.5.4.5 update\_claw()

```
void update_claw ( )
```

Updates the claw motor values.

##### Author

Chris Jerrett

Definition at line 10 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()**.

## 7.6 claw.h

```
00001
00007 #ifndef _CLAW_H_
00008 #define _CLAW_H_
00009
00010 #include "controller.h"
00011 #include "motor_ports.h"
00012 #include "sensor_ports.h"
00013 #include "slew.h"
00014 #include <API.h>
00015
00020 #define MAX_CLAW_SPEED 127
00021
00025 #define MIN_CLAW_SPEED -127
00026
00031 #define CLAW_CLOSE MASTER, 5, JOY_UP
00032
00037 #define CLAW_OPEN MASTER, 5, 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 { CLAW_OPEN_STATE, CLAW_CLOSE_STATE, CLAW_NEUTRAL_STATE };
00086
00087 #endif
```

## 7.7 include/controller.h File Reference

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

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

### 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 **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 }**
- 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 using 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 **88** 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 **95** 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 **60** 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 **67** 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 74 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 81 of file **controller.h**.

Referenced by **get\_joystick\_cord()**.

### 7.7.3 Enumeration Type Documentation

#### 7.7.3.1 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   |  |
| JOY2_7R   |  |
| JOY2_7D   |  |
| JOY2_8U   |  |
| JOY2_8L   |  |
| JOY2_8R   |  |
| JOY2_8D   |  |
| LCD_LEFT  |  |
| LCD_CENT  |  |
| LCD_RIGHT |  |

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

## 7.7.3.2 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 102 of file **controller.h**.

## 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**.

## 7.8 controller.h

```
00001
00009 #ifndef _CONTROLLER_H_
00010 #define _CONTROLLER_H_
00011
00012 #include "vmath.h"
00013 #include <API.h>
00014
00015 #define RIGHT_BUTTONS 8
00016 #define LEFT_BUTTONS 7
00017 #define RIGHT_BUMPERS 5
00018 #define LEFT_BUMPERS 6
00019
00023 typedef enum {
00024     JOY1_5D = 0,
00025     JOY1_5U = 1,
00026     JOY1_6D = 2,
00027     JOY1_6U = 3,
00028     JOY1_7U = 4,
00029     JOY1_7L = 5,
00030     JOY1_7R = 6,
00031     JOY1_7D = 7,
00032     JOY1_8U = 8,
00033     JOY1_8L = 9,
00034     JOY1_8R = 10,
00035     JOY1_8D = 11,
00036
00037     JOY2_5D = 12,
00038     JOY2_5U = 13,
```

```

00039     JOY2_6D = 14,
00040     JOY2_6U = 15,
00041     JOY2_7U = 16,
00042     JOY2_7L = 17,
00043     JOY2_7R = 18,
00044     JOY2_7D = 19,
00045     JOY2_8U = 20,
00046     JOY2_8L = 21,
00047     JOY2_8R = 22,
00048     JOY2_8D = 23,
00049
00050     LCD_LEFT = 24,
00051     LCD_CENT = 25,
00052     LCD_RIGHT = 26
00053 } button_t;
00054
00060 #define MASTER 1
00061
00067 #define PARTNER 2
00068
00074 #define RIGHT_JOY_X 1
00075
00081 #define RIGHT_JOY_Y 2
00082
00088 #define LEFT_JOY_X 4
00089
00095 #define LEFT_JOY_Y 3
00096
00102 enum joystick {
00104     RIGHT_JOY,
00106     LEFT_JOY,
00107 };
00108
00113 struct cord get_joystick_cord(enum joystick side, int controller);
00114
00115 #endif

```

## 7.9 include/drive.h File Reference

Drive base definitions and enumerations.

```

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

```

### 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 threshold 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 27 of file **drive.h**.

## 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

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

Definition at line **62** 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()**.

#### 7.9.5.2 setThresh()

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

Sets the deadzone threshold on the drive.

##### Author

Chris Jerrett  
Christian Desimone

Definition at line **21** of file **drive.c**.

References **thresh**.

## 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 28 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()`.

## 7.10 drive.h

```
00001
00008 #ifndef _DRIVE_H_
00009 #define _DRIVE_H_
00010
00011 #include "partner.h"
00012 #include <API.h>
00013
00018 #define THRESHOLD 10
00019
00027 typedef enum side { LEFT, BOTH, RIGHT } side_t;
00028
00036 void set_side_speed(side_t side, int speed);
00037
00042 void setThresh(int t);
00043
00049 void update_drive_motors();
00050
00051 #endif
```

## 7.11 include/encoders.h File Reference

wrapper around encoder functions

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

## Macros

- `#define IME_NUMBER 2`

*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 2
```

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

#### See also

**init\_encoders()** (p. 47)

#### Author

Chris Jerrett

#### Date

9/9/2017

#### See also

**IME\_NUMBER** (p. 46)

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 **30** of file **encoders.c**.

Referenced by **calculate\_encoder\_angle()**, and **calculate\_encoder\_odometry()**.

#### 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 **41** of file **encoders.c**.

### 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. 46)

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

References **error()**, and **IME\_NUMBER**.

Referenced by **initialize()**.

## 7.12 encoders.h

```
00001
00007 #ifndef _ENCODERS_H_
00008 #define _ENCODERS_H_
00009
00010 #include <API.h>
00011
00020 #define IME_NUMBER 2
00021
00028 bool init_encoders();
00029
00035 int get_encoder_ticks(unsigned char address);
00036
00042 int get_encoder_velocity(unsigned char address);
00043
00044 #endif
```

## 7.13 include/gyro.h File Reference

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

#### 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 10 of file **gyro.c**.

References **GYRO\_PORT**.

#### 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**.

## 7.14 gyro.h

```

00001 #ifndef _GYRO_H_
00002 #define _GYRO_H_
00003
00004 #include "API.h"
00005
00006 #define GYRO_PORT 1
00007 #define GYRO_MULTIPLIER 0
00008
00009 bool init_main_gyro();
00010 float get_main_gyro_angluar_velocity();
00011
00012 #endif

```

## 7.15 include/lcd.h File Reference

LCD wrapper functions and macros.

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

### 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 formatted string to a line on the lcd. Similar to printf*
- void **lcd\_set\_backlight** (bool state)  
*sets the backlight of the lcd*
- void **prompt\_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 or 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**.

### 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

|            |                                 |
|------------|---------------------------------|
| <i>lcd</i> | the uart port of the lcd screen |
|------------|---------------------------------|

#### See also

uart1  
uart2

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line **62** of file **lcd.c**.

References **lcd\_clear()**, and **lcd\_port**.

Referenced by **initialize()**.

**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()**.

**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. 9)

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 **prompt\_confirmation()**.

#### 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

|             |                      |
|-------------|----------------------|
| <i>line</i> | the line to print on |
| <i>str</i>  | string to print      |

##### Author

Chris Jerrett

##### Date

9/9/2017

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

References **lcd\_assert()**, and **lcd\_port**.

Referenced by **display\_menu()**, and **prompt\_confirmation()**.

#### 7.15.4.5 `lcd_printf()`

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

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

##### Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>line</i>       | the line to print on          |
| <i>format_str</i> | format string string to print |

##### Author

Chris Jerrett



**Date**

9/9/2017

Definition at line 87 of file **lcd.c**.References **lcd\_assert()**, and **lcd\_port**.**7.15.4.6 lcd\_set\_backlight()**

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

sets the backlight of the lcd

**Parameters**

|              |  |
|--------------|--|
| <i>state</i> | a boolean representing the state of the backlight. true = on, false = off. |
|--------------|--|

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line 99 of file **lcd.c**.References **lcd\_assert()**, and **lcd\_port**.**7.15.4.7 prompt\_confirmation()**

```
void prompt_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**

|                     |                                   |
|---------------------|-----------------------------------|
| <i>confirm_text</i> | the text for the user to confirm. |
|---------------------|-----------------------------------|

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line 113 of file `lcd.c`.References `lcd_assert()`, `lcd_get_pressed_buttons()`, `lcd_print()`, and `PRESSED`.**7.16 lcd.h**

```

00001
00008 #ifndef _LCD_H_
00009 #define _LCD_H_
00010
00011 #include <API.h>
00012
00018 #define TOP_ROW 1
00019
00025 #define BOTTOM_ROW 2
00026
00036 typedef enum {
00038     RELEASED = false,
00040     PRESSED = true,
00041 } button_state;
00042
00048 typedef struct {
00049     button_state left;
00050     button_state middle;
00051     button_state right;
00052 } lcd_buttons;
00053
00061 lcd_buttons lcd_get_pressed_buttons();
00062
00068 void lcd_clear();
00069
00080 void init_main_lcd(FILE *lcd);
00081
00089 void lcd_print(unsigned int line, const char *str);
00090
00098 void lcd_printf(unsigned int line, const char *format_str, ...);
00099
00107 void lcd_set_backlight(bool state);
00108
00118 void prompt_confirmation(const char *confirm_text);
00119
00120 #endif

```

**7.17 include/lifter.h File Reference**

Declarations and macros for controlling and manipulating the lifter.

```

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

```

## 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\_DOWN MASTER, 6, 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\_UP MASTER, 6, 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\_D** 0  
*The derivative constant for the main lifter PID.*
- **#define MAIN\_LIFTER\_I** 0.0000001  
*The integral constant for the main lifter PID.*
- **#define MAIN\_LIFTER\_MIN\_HEIGHT** 1700
- **#define MAIN\_LIFTER\_P** 0  
*The proportional constant for the main lifter PID.*
- **#define MAIN\_LIFTER\_POT** 1
- **#define SECONDARY\_LIFTER\_D** 0  
*The derivative constant for the secondary lifter PID.*
- **#define SECONDARY\_LIFTER\_DOWN MASTER, 8, JOY\_DOWN**  
*The secondary lifter down controller params.*
- **#define SECONDARY\_LIFTER\_I** 0.000  
*The integral constant for the secondary lifter PID.*
- **#define SECONDARY\_LIFTER\_MAX\_HEIGHT** 3120
- **#define SECONDARY\_LIFTER\_MIN\_HEIGHT** 2000
- **#define SECONDARY\_LIFTER\_P** .05  
*The proportional constant for the secondary lifter PID.*
- **#define SECONDARY\_LIFTER\_POT\_PORT** 2
- **#define SECONDARY\_LIFTER\_UP MASTER, 8, JOY\_UP**  
*The secondary lifter up controller params.*
- **#define THRESHOLD** 10  
*The threshold of a significant 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 **62** 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\_DOWN

```
#define LIFTER_DOWN MASTER, 6, JOY_DOWN
```

The lifter down controller params.

Definition at line **72** of file **lifter.h**.

Referenced by **main\_lifter\_update()**.

### 7.17.2.4 LIFTER\_DOWN\_PARTNER

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

The lifter down controller params for the partner.

Definition at line **97** of file **lifter.h**.

### 7.17.2.5 LIFTER\_DRIVER\_LOAD

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

Height to raise lifter to driver preload height.

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

### 7.17.2.6 LIFTER\_UP

```
#define LIFTER_UP MASTER, 6, JOY_UP
```

The lifter up controller params.

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

Referenced by **main\_lifter\_update()**.

#### 7.17.2.7 LIFTER\_UP\_PARTNER

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

The lifter up controller params for the partner.

Definition at line **92** of file **lifter.h**.

#### 7.17.2.8 MAIN\_LIFTER\_D

```
#define MAIN_LIFTER_D 0
```

The derivative constant for the main lifter PID.

Definition at line **47** of file **lifter.h**.

#### 7.17.2.9 MAIN\_LIFTER\_I

```
#define MAIN_LIFTER_I 0.0000001
```

The integral constant for the main lifter PID.

Definition at line **52** of file **lifter.h**.

#### 7.17.2.10 MAIN\_LIFTER\_MIN\_HEIGHT

```
#define MAIN_LIFTER_MIN_HEIGHT 1700
```

Definition at line **107** of file **lifter.h**.

#### 7.17.2.11 MAIN\_LIFTER\_P

```
#define MAIN_LIFTER_P 0
```

The proportional constant for the main lifter PID.

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

#### 7.17.2.12 MAIN\_LIFTER\_POT

```
#define MAIN_LIFTER_POT 1
```

Definition at line **105** of file **lifter.h**.

Referenced by **autonomous()**, and **main\_lifter\_update()**.

#### 7.17.2.13 SECONDARY\_LIFTER\_D

```
#define SECONDARY_LIFTER_D 0
```

The derivative constant for the secondary lifter PID.

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

Referenced by **secondary\_lifter\_update()**.

#### 7.17.2.14 SECONDARY\_LIFTER\_DOWN

```
#define SECONDARY_LIFTER_DOWN MASTER, 8, JOY_DOWN
```

The secondary lifter down controller params.

Definition at line **82** of file **lifter.h**.

Referenced by **secondary\_lifter\_update()**.

#### 7.17.2.15 SECONDARY\_LIFTER\_I

```
#define SECONDARY_LIFTER_I 0.000
```

The integral constant for the secondary lifter PID.

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

Referenced by **secondary\_lifter\_update()**.

#### 7.17.2.16 SECONDARY\_LIFTER\_MAX\_HEIGHT

```
#define SECONDARY_LIFTER_MAX_HEIGHT 3120
```

Definition at line **101** of file **lifter.h**.

#### 7.17.2.17 SECONDARY\_LIFTER\_MIN\_HEIGHT

```
#define SECONDARY_LIFTER_MIN_HEIGHT 2000
```

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

#### 7.17.2.18 SECONDARY\_LIFTER\_P

```
#define SECONDARY_LIFTER_P .05
```

The proportional constant for the secondary lifter PID.

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

Referenced by **secondary\_lifter\_update()**.

#### 7.17.2.19 SECONDARY\_LIFTER\_POT\_PORT

```
#define SECONDARY_LIFTER_POT_PORT 2
```

Definition at line **99** of file **lifter.h**.

Referenced by **autonomous()**, and **secondary\_lifter\_update()**.

#### 7.17.2.20 SECONDARY\_LIFTER\_UP

```
#define SECONDARY_LIFTER_UP MASTER, 8, JOY_UP
```

The secondary lifter up controller params.

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

Referenced by **secondary\_lifter\_update()**.

#### 7.17.2.21 THRESHOLD

```
#define THRESHOLD 10
```

The threshold of a significant speed for the lifter.

Definition at line **57** 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 **216** of file **lifter.c**.

References **getLifterTicks()**.

#### 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 **207** of file **lifter.c**.

References **LIFTER**.

Referenced by **getLifterHeight()**.

#### 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 **196** of file **lifter.c**.

References **DEG\_MAX**, **INIT\_ROTATION**, and **TICK\_MAX**.

**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 **70** of file **lifter.c**.

References **MAX\_SPEED**, and **set\_main\_lifter\_motors()**.

#### 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 **86** of file **lifter.c**.

References **MAX\_SPEED**, and **set\_secondary\_lifter\_motors()**.

#### 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 **62** of file **lifter.c**.

References **MAX\_SPEED**, and **set\_main\_lifter\_motors()**.

Referenced by **autostack\_routine()**.

#### 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 **78** of file **lifter.c**.

References **MIN\_SPEED**, and **set\_secondary\_lifter\_motors()**.

#### 7.17.3.8 set\_lifter\_pos()

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

Sets the lifter positions to the given value.

**Parameters**

|            |                      |
|------------|----------------------|
| <i>pos</i> | The height in inches |
|------------|----------------------|

**Author**

Chris Jerrett

**Date**

9/12/2017

Definition at line **54** 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**

|          |   |
|----------|---|
| <i>v</i> | value for the lifter motor. Between -128 - 127, any values outside are clamped. |
|----------|---|

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line **45** of file **lifter.c**.References **MOTOR\_LIFT**, and **set\_motor\_slew()**.Referenced by **autonomous()**, **autostack\_routine()**, **lower\_main\_lifter()**, **main\_lifter\_update()**, and **raise\_main\_lifter()**.**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

|                |   |
|----------------|---|
| <code>v</code> | value for the lifter motor. Between -128 - 127, any values outside are clamped. |
|----------------|---|

#### Author

Chris Jerrett

#### Date

1/6/2018

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

References **MOTOR\_SECONDARY\_LIFTER**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**, **lower\_secondary\_lifter()**, **raise\_secondary\_lifter()**, and **secondary\_lifter\_↔update()**.

#### 7.17.3.11 update\_lifter()

```
void update_lifter ( )
```

Updates the lifter in teleop.

#### Author

Chris Jerrett

#### Date

9/9/2017

Definition at line **183** of file **lifter.c**.

References **main\_lifter\_update()**, **secondary\_lifter\_update()**, and **secondary\_override**.

Referenced by **operatorControl()**.

## 7.18 lifter.h

```

00001
00007 #ifndef _LIFTER_H_
00008 #define _LIFTER_H_
00009
00010 #include "controller.h"
00011 #include "drive.h"
00012 #include "motor_ports.h"
00013 #include "partner.h"
00014 #include "potentiometer.h"
00015 #include "sensor_ports.h"
00016 #include "slew.h"
00017 #include <API.h>
00018
00022 #define INIT_ROTATION 680
00023
00027 #define SECONDARY_LIFTER_P .05
00028
00032 #define SECONDARY_LIFTER_D 0
00033
00037 #define SECONDARY_LIFTER_I 0.000
00038
00042 #define MAIN_LIFTER_P 0
00043
00047 #define MAIN_LIFTER_D 0
00048
00052 #define MAIN_LIFTER_I 0.0000001
00053
00057 #define THRESHOLD 10
00058
00062 #define HEIGHT 19.1 - 3.8
00063
00067 #define LIFTER_UP MASTER, 6, JOY_UP
00068
00072 #define LIFTER_DOWN MASTER, 6, JOY_DOWN
00073
00077 #define SECONDARY_LIFTER_UP MASTER, 8, JOY_UP
00078
00082 #define SECONDARY_LIFTER_DOWN MASTER, 8, JOY_DOWN
00083
00087 #define LIFTER_DRIVER_LOAD MASTER, RIGHT_BUTTONS, JOY_RIGHT
00088
00092 #define LIFTER_UP_PARTNER PARTNER, 5, JOY_UP
00093
00097 #define LIFTER_DOWN_PARTNER PARTNER, 5, JOY_DOWN
00098
00099 #define SECONDARY_LIFTER_POT_PORT 2
00100
00101 #define SECONDARY_LIFTER_MAX_HEIGHT 3120
00102
00103 #define SECONDARY_LIFTER_MIN_HEIGHT 2000
00104
00105 #define MAIN_LIFTER_POT 1
00106
00107 #define MAIN_LIFTER_MIN_HEIGHT 1700
00108
00117 void set_secondary_lifter_motors(const int v);
00118
00127 void set_main_lifter_motors(const int v);
00128
00136 void set_lifter_pos(int pos);
00137
00144 void raise_main_lifter();
00145
00152 void lower_main_lifter();
00153
00160 void raise_secondary_lifter();
00161
00168 void lower_secondary_lifter();
00169
00176 void update_lifter();
00177
00186 float lifterPotentiometerToDegree(int x);
00187
00195 int getLifterTicks();
00196
00204 double getLifterHeight();
00205
00206 #endif

```

## 7.19 include/list.h File Reference

```
#include <stdlib.h>
```

### Data Structures

- struct **list\_iterator\_t**
- struct **list\_node**
- struct **list\_t**

### Macros

- #define **LIST\_FREE** free
- #define **LIST\_MALLOC** malloc

### Typedefs

- typedef struct **list\_node** **list\_node\_t**

### Enumerations

- enum **list\_direction\_t** { **LIST\_HEAD**, **LIST\_TAIL** }

### Functions

- **list\_node\_t** \* **list\_at** ( **list\_t** \*self, int index)
- void **list\_destroy** ( **list\_t** \*self)
- **list\_node\_t** \* **list\_find** ( **list\_t** \*self, void \*val)
- void **list\_iterator\_destroy** ( **list\_iterator\_t** \*self)
- **list\_iterator\_t** \* **list\_iterator\_new** ( **list\_t** \*list, **list\_direction\_t** direction)
- **list\_iterator\_t** \* **list\_iterator\_new\_from\_node** ( **list\_node\_t** \*node, **list\_direction\_t** direction)
- **list\_node\_t** \* **list\_iterator\_next** ( **list\_iterator\_t** \*self)
- **list\_node\_t** \* **list\_lpop** ( **list\_t** \*self)
- **list\_node\_t** \* **list\_lpush** ( **list\_t** \*self, **list\_node\_t** \*node)
- **list\_t** \* **list\_new** ()
- **list\_node\_t** \* **list\_node\_new** (void \*val)
- void **list\_remove** ( **list\_t** \*self, **list\_node\_t** \*node)
- **list\_node\_t** \* **list\_rpop** ( **list\_t** \*self)
- **list\_node\_t** \* **list\_rpush** ( **list\_t** \*self, **list\_node\_t** \*node)

#### 7.19.1 Macro Definition Documentation

### 7.19.1.1 LIST\_FREE

```
#define LIST_FREE free
```

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

Referenced by **list\_destroy()**, **list\_iterator\_destroy()**, and **list\_remove()**.

### 7.19.1.2 LIST\_MALLOC

```
#define LIST_MALLOC malloc
```

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

Referenced by **list\_iterator\_new\_from\_node()**, **list\_new()**, and **list\_node\_new()**.

## 7.19.2 Typedef Documentation

### 7.19.2.1 list\_node\_t

```
typedef struct list_node list_node_t
```

## 7.19.3 Enumeration Type Documentation

### 7.19.3.1 list\_direction\_t

```
enum list_direction_t
```

#### Enumerator

|           |  |
|-----------|--|
| LIST_HEAD |  |
| LIST_TAIL |  |

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

## 7.19.4 Function Documentation



#### 7.19.4.1 list\_at()

```
list_node_t* list_at (
    list_t * self,
    int index )
```

Definition at line 162 of file list.c.

References LIST\_HEAD, list\_iterator\_destroy(), list\_iterator\_new(), list\_iterator\_next(), and LIST\_TAIL.

#### 7.19.4.2 list\_destroy()

```
void list_destroy (
    list_t * self )
```

Definition at line 30 of file list.c.

References LIST\_FREE, list\_node::next, and list\_node::val.

Referenced by deinit\_routines().

#### 7.19.4.3 list\_find()

```
list_node_t* list_find (
    list_t * self,
    void * val )
```

Definition at line 136 of file list.c.

References LIST\_HEAD, list\_iterator\_destroy(), list\_iterator\_new(), list\_iterator\_next(), and list\_node::val.

#### 7.19.4.4 list\_iterator\_destroy()

```
void list_iterator_destroy (
    list_iterator_t * self )
```

Definition at line 52 of file list\_iterator.c.

References LIST\_FREE.

Referenced by list\_at(), list\_find(), and routine\_task().

#### 7.19.4.5 `list_iterator_new()`

```
list_iterator_t* list_iterator_new (
    list_t * list,
    list_direction_t direction )
```

Definition at line 15 of file `list_iterator.c`.

References `list_t::head`, `LIST_HEAD`, `list_iterator_new_from_node()`, and `list_t::tail`.

Referenced by `list_at()`, `list_find()`, and `routine_task()`.

#### 7.19.4.6 `list_iterator_new_from_node()`

```
list_iterator_t* list_iterator_new_from_node (
    list_node_t * node,
    list_direction_t direction )
```

Definition at line 25 of file `list_iterator.c`.

References `LIST_MALLOC`, and `list_iterator_t::next`.

Referenced by `list_iterator_new()`.

#### 7.19.4.7 `list_iterator_next()`

```
list_node_t* list_iterator_next (
    list_iterator_t * self )
```

Definition at line 40 of file `list_iterator.c`.

References `LIST_HEAD`, `list_node::next`, and `list_node::prev`.

Referenced by `list_at()`, `list_find()`, and `routine_task()`.

#### 7.19.4.8 `list_lpop()`

```
list_node_t* list_lpop (
    list_t * self )
```

Definition at line 93 of file `list.c`.

References `list_node::next`, and `list_node::prev`.

## 7.19.4.9 list\_lpush()

```
list_node_t* list_lpush (
    list_t * self,
    list_node_t * node )
```

Definition at line 114 of file **list.c**.

References **list\_node::next**, and **list\_node::prev**.

## 7.19.4.10 list\_new()

```
list_t* list_new ( )
```

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

References **list\_t::head**, and **LIST\_MALLOC**.

Referenced by **init\_routine()**.

## 7.19.4.11 list\_node\_new()

```
list_node_t* list_node_new (
    void * val )
```

Definition at line 14 of file **list\_node.c**.

References **LIST\_MALLOC**, **list\_node::next**, **list\_node::prev**, and **list\_node::val**.

Referenced by **register\_routine()**.

## 7.19.4.12 list\_remove()

```
void list_remove (
    list_t * self,
    list_node_t * node )
```

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

References **LIST\_FREE**, **list\_node::next**, **list\_node::prev**, and **list\_node::val**.

## 7.19.4.13 list\_rpop()

```
list_node_t* list_rpop (
    list_t * self )
```

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

References **list\_node::next**, and **list\_node::prev**.

## 7.19.4.14 list\_rpush()

```
list_node_t* list_rpush (
    list_t * self,
    list_node_t * node )
```

Definition at line 51 of file **list.c**.

References **list\_node::next**, and **list\_node::prev**.

Referenced by **register\_routine()**.

## 7.20 list.h

```
00001
00002 //
00003 // list.h
00004 //
00005 // Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
00006 //
00007
00008 #ifndef LIST_H
00009 #define LIST_H
00010
00011 #ifdef __cplusplus
00012 extern "C" {
00013 #endif
00014
00015 #include <stdlib.h>
00016
00017 // Memory management macros
00018
00019 #ifndef LIST_MALLOC
00020 #define LIST_MALLOC malloc
00021 #endif
00022
00023 #ifndef LIST_FREE
00024 #define LIST_FREE free
00025 #endif
00026
00027 /*
00028  * list_t iterator direction.
00029  */
00030
00031 typedef enum {
00032     LIST_HEAD
00033     , LIST_TAIL
00034 } list_direction_t;
00035
00036 /*
00037  * list_t node struct.
00038  */
00039
00040 typedef struct list_node {
```

```
00041     struct list_node *prev;
00042     struct list_node *next;
00043     void *val;
00044 } list_node_t;
00045
00046 /*
00047  * list_t struct.
00048  */
00049
00050 typedef struct {
00051     list_node_t *head;
00052     list_node_t *tail;
00053     unsigned int len;
00054     void (*free)(void *val);
00055     int (*match)(void *a, void *b);
00056 } list_t;
00057
00058 /*
00059  * list_t iterator struct.
00060  */
00061
00062 typedef struct {
00063     list_node_t *next;
00064     list_direction_t direction;
00065 } list_iterator_t;
00066
00067 // Node prototypes.
00068
00069 list_node_t *
00070 list_node_new(void *val);
00071
00072 // list_t prototypes.
00073
00074 list_t *
00075 list_new();
00076
00077 list_node_t *
00078 list_rpush(list_t *self, list_node_t *node);
00079
00080 list_node_t *
00081 list_lpush(list_t *self, list_node_t *node);
00082
00083 list_node_t *
00084 list_find(list_t *self, void *val);
00085
00086 list_node_t *
00087 list_at(list_t *self, int index);
00088
00089 list_node_t *
00090 list_rpop(list_t *self);
00091
00092 list_node_t *
00093 list_lpop(list_t *self);
00094
00095 void
00096 list_remove(list_t *self, list_node_t *node);
00097
00098 void
00099 list_destroy(list_t *self);
00100
00101 // list_t iterator prototypes.
00102
00103 list_iterator_t *
00104 list_iterator_new(list_t *list, list_direction_t direction);
00105
00106 list_iterator_t *
00107 list_iterator_new_from_node(list_node_t *node, list_direction_t direction);
00108
00109 list_node_t *
00110 list_iterator_next(list_iterator_t *self);
00111
00112 void
00113 list_iterator_destroy(list_iterator_t *self);
00114
00115 #ifdef __cplusplus
00116 }
00117 #endif
00118
00119 #endif /* LIST_H */
```

## 7.21 include/localization.h File Reference

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

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

### Data Structures

- struct **location**

### Macros

- #define **LOCALIZATION\_UPDATE\_FREQUENCY** 0.500

### Functions

- int **calculate\_encoder\_angle** ()
- struct **location** **get\_position** ()  
*Gets the current position 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.*

#### 7.21.1 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**.

#### 7.21.2 Macro Definition Documentation

### 7.21.2.1 LOCALIZATION\_UPDATE\_FREQUENCY

```
#define LOCALIZATION_UPDATE_FREQUENCY 0.500
```

How often the localization code updates the position.

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

Referenced by **calculate\_gryo Angular\_velocity()**, **init\_localization()**, and **integrate\_gyro\_w()**.

## 7.21.3 Function Documentation

### 7.21.3.1 calculate\_encoder\_angle()

```
int calculate_encoder_angle ( )
```

Definition at line 101 of file **localization.c**.

References **CPR**, **get\_encoder\_ticks()**, and **WIDTH**.

Referenced by **autonomous()**.

### 7.21.3.2 get\_position()

```
struct location get_position ( )
```

Gets the current position of the robot.

#### Parameters

|              |                |
|--------------|----------------|
| <i>gyro1</i> | The first gyro |
|--------------|----------------|

#### Returns

the location of the robot as a struct.

Definition at line 32 of file **localization.c**.

### 7.21.3.3 `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

|              |   |
|--------------|---|
| <i>gyro1</i> | 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 **123** of file **localization.c**.

References **g1**, **last\_call**, **localization\_task**, **LOCALIZATION\_UPDATE\_FREQUENCY**, **makeMatrix()**, and **update\_position()**.

### 7.21.3.4 `update_position()`

```
void update_position ( )
```

Updates the position from the localization.

#### Author

Chris Jerrett

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

References **calculate\_accelerometer\_odometry()**, and **last\_call**.

Referenced by **init\_localization()**.



## 7.22 localization.h

```
00001
00008 #ifndef _LOCALIZATION_H_
00009 #define _LOCALIZATION_H_
00010
00011 #include "encoders.h"
00012 #include "matrix.h"
00013 #include <API.h>
00014 #include <math.h>
00015
00019 #define LOCALIZATION_UPDATE_FREQUENCY 0.500
00020
00024 struct location {
00025     int x;
00026     int y;
00027     int theta;
00028 };
00029
00042 bool init_localization(const unsigned char gyrol, unsigned short multiplier,
00043                      int start_x, int start_y, int start_theta);
00044
00051 struct location get_position();
00052
00058 void update_position();
00059
00060 int calculate_encoder_angle();
00061
00062 #endif
```

## 7.23 include/log.h File Reference

Contains logging functions.

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

### 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*

### 7.23.1 Detailed Description

Contains logging functions.

#### Author

Chris Jerrett

#### Date

9/16/2017

Definition in file **log.h**.

### 7.23.2 Macro Definition Documentation

#### 7.23.2.1 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**.

### 7.23.2.2 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()**.

### 7.23.2.3 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**.

### 7.23.2.4 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()**.

### 7.23.2.5 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()**.

## 7.23.3 Function Documentation

### 7.23.3.1 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. 186)

#### Parameters

|                      |             |
|----------------------|-------------|
| <i>debug_message</i> | the message |
|----------------------|-------------|

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

References **ERROR**, and **log\_level**.

Referenced by **set\_motor\_immediate()**, and **set\_motor\_slew()**.

### 7.23.3.2 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. 186)

Author

Chris Jerrett

Date

9/10/17

Parameters

|                      |             |
|----------------------|-------------|
| <i>error_message</i> | the message |
|----------------------|-------------|

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

References **log\_info()**, **log\_level**, and **NONE**.

Referenced by **assert()**, **create\_menu()**, **init\_encoders()**, and **initialize()**.

### 7.23.3.3 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. 186)

Parameters

|                     |             |
|---------------------|-------------|
| <i>info_message</i> | the message |
|---------------------|-------------|

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

References **ERROR**, **log\_info()**, and **log\_level**.

Referenced by **initialize()**.

#### 7.23.3.4 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

|                |                        |
|----------------|------------------------|
| <i>use_lcd</i> | whether to use the lcd |
| <i>lcd</i>     | the lcd                |

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

References **log\_lcd**.

Referenced by **initialize()**.

#### 7.23.3.5 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. 186)

##### Author

Chris Jerrett

##### Date

9/10/17

## Parameters

|                              |             |
|------------------------------|-------------|
| <code>warning_message</code> | the message |
|------------------------------|-------------|

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

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

Referenced by `init_slew()`.

## 7.24 log.h

```

00001
00007 #ifndef _LOG_H_
00008 #define _LOG_H_
00009
00010 #include "lcd.h"
00011 #include <API.h>
00012
00019 #define NONE 0
00020
00027 #define ERROR 1
00028
00035 #define WARNING 2
00036
00042 #define INFO 3
00043
00050 #define DEBUG 4
00051
00060 void init_error(bool use_lcd, FILE *lcd);
00061
00070 void error(const char *error_message);
00071
00080 void warning(const char *warning_message);
00081
00089 void info(const char *info_message);
00090
00098 void debug(const char *debug_message);
00099
00100 #endif

```

## 7.25 include/main.h File Reference

Header file for global functions.

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

## Functions

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

### 7.25.1 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.

Copyright (c) 2011-2014, Purdue University ACM SIG BOTS. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Purdue University ACM SIG BOTS nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL PURDUE UNIVERSITY ACM SIG BOTS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

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**.

### 7.25.2 Function Documentation



### 7.25.2.1 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.88) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

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

References **BOTH**, **calculate\_encoder\_angle()**, **close\_claw()**, **deinit\_slew()**, **DEPLOY\_HEIGHT**, **HALF\_ROTATE**, **init\_slew()**, **LEFT**, **lower\_intake()**, **LOWEST\_HEIGHT**, **MAIN\_LIFTER\_POT**, **MAX\_HEIGHT**, **MAX\_SPEED**, **MID\_LEFT\_DRIVE**, **MID\_RIGHT\_DRIVE**, **MIN\_SPEED**, **MOBILE\_GOAL\_DISTANCE**, **MOBILE\_GOAL\_HEIGHT**, **open\_claw()**, **raise\_intake()**, **RIGHT**, **SECONDARY\_LIFTER\_POT\_PORT**, **set\_claw\_motor()**, **set\_intake\_motor()**, **set\_main\_lifter\_motors()**, **set\_secondary\_lifter\_motors()**, **set\_side\_speed()**, and **ZONE\_DISTANCE**.

### 7.25.2.2 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.88) and **autonomous()** (p.86) 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 50 of file **init.c**.

References **battery\_level\_acceptable()**, **error()**, **info()**, **init\_encoders()**, **init\_error()**, **init\_main\_lcd()**, **init\_menu\_var()**, **lifter\_ultrasonic**, and **STRING\_TYPE**.

### 7.25.2.3 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 **35** of file **init.c**.

### 7.25.2.4 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 **49** of file **opcontrol.c**.

References **buttonInit()**, **init\_routine()**, **init\_slew()**, **update\_claw()**, **update\_drive\_motors()**, **update\_intake()**, and **update\_lifter()**.

## 7.26 main.h

```

00001
00044 #ifndef MAIN_H_
00045
00046 // This prevents multiple inclusion, which isn't bad for this file but is good
00047 // practice
00048 #define MAIN_H_
00049
00050 #include <API.h>
00051
00052 // Allow usage of this file in C++ programs
00053 #ifdef __cplusplus
00054 extern "C" {
00055 #endif
00056
00057 // #define AUTO_DEBUG
00058
00059 // A function prototype looks exactly like its declaration, but with a semicolon
00060 // instead of actual code. If a function does not match a prototype, compile
00061 // errors will occur.
00062
00063 // Prototypes for initialization, operator control and autonomous
00064
00082 void autonomous();
00093 void initializeIO();
00107 void initialize();
00129 void operatorControl();
00130
00131 // End C++ export structure
00132 #ifdef __cplusplus
00133 }
00134 #endif
00135
00136 #endif

```

## 7.27 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 prevalent, be sure to use the function to reclaim some of that memory.

## 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 diagonal 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.*

### 7.27.1 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 prevalent, be sure to use the function to reclaim some of that memory.

Definition in file **matrix.h**.

### 7.27.2 Typedef Documentation

#### 7.27.2.1 matrix

```
typedef struct _matrix matrix
```

A struct representing a matrix

### 7.27.3 Function Documentation

#### 7.27.3.1 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 15 of file **matrix.c**.

References **error()**.

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

#### 7.27.3.2 copyMatrix()

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

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

##### Parameters

|          |                         |
|----------|-------------------------|
| <i>m</i> | a pointer to the matrix |
|----------|-------------------------|

**Returns**

a copied matrix

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

References **scaleMatrix()**.

**7.27.3.3 covarianceMatrix()**

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

returns the covariance of the matrix

**Parameters**

|            |        |
|------------|--------|
| <i>the</i> | matrix |
|------------|--------|

**Returns**

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

Definition at line **169** of file **matrix.c**.

References **assert()**, **\_matrix::data**, **freeMatrix()**, **\_matrix::height**, **makeMatrix()**, **meanMatrix()**, and **\_matrix↵::width**.

**7.27.3.4 dotDiagonalMatrix()**

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

performs a diagonal 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**

|          |                   |
|----------|-------------------|
| <i>a</i> | the first matrix  |
| <i>b</i> | the second matrix |

### Returns

the matrix result

Definition at line **389** of file **matrix.c**.

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

#### 7.27.3.5 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

|            |                  |
|------------|------------------|
| <i>a</i>   | the first matrix |
| <i>the</i> | second matrix    |

### Returns

the result of the dot product

Definition at line **336** of file **matrix.c**.

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

#### 7.27.3.6 freeMatrix()

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

Frees the resources of a matrix.

### Parameters

|            |                |
|------------|----------------|
| <i>the</i> | matrix to free |
|------------|----------------|

Definition at line 59 of file **matrix.c**.

References **\_matrix::data**.

Referenced by **covarianceMatrix()**.

#### 7.27.3.7 identityMatrix()

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

Returns an identity matrix of size *n* by *n*.

##### Parameters

|          |                              |
|----------|------------------------------|
| <i>n</i> | the input matrix. parameter. |
| <i>n</i> | the input matrix.            |

##### Returns

the identity matrix parameter.

Definition at line 93 of file **matrix.c**.

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

#### 7.27.3.8 makeMatrix()

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

Makes a matrix with a width and height parameters.

##### Parameters

|               |                          |
|---------------|--------------------------|
| <i>width</i>  | The width of the matrix  |
| <i>height</i> | the height of the matrix |

##### Returns

the new matrix

Definition at line 28 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()`.

### 7.27.3.9 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()`.

### 7.27.3.10 multiplyMatrix()

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

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

#### Parameters

|          |   |
|----------|---|
| <i>a</i> | the first matrix  |
| <i>b</i> | the seconf matrix return the result of the multiplication |

Definition at line 231 of file `matrix.c`.

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

### 7.27.3.11 printMatrix()

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

Prints a matrix.

**Parameters**

|            |        |
|------------|--------|
| <i>the</i> | matrix |
|------------|--------|

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

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

**7.27.3.12 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**

|            |   |
|------------|---|
| <i>the</i> | matrix to swap. This method changes the input matrix. |
| <i>the</i> | first row   |
| <i>the</i> | second row  |

Definition at line **292** of file **matrix.c**.

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

**7.27.3.13 scaleMatrix()**

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

scales a matrix.

**Parameters**

|            |                     |
|------------|---------------------|
| <i>m</i>   | the matrix to scale |
| <i>the</i> | value to scale by   |

**Returns**

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

Definition at line 270 of file **matrix.c**.

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

Referenced by **copyMatrix()**.

#### 7.27.3.14 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 115 of file **matrix.c**.

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

#### 7.27.3.15 transposeMatrix()

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

returns the transpose matrix.

##### Parameters

|            |                      |
|------------|----------------------|
| <i>the</i> | matrix to transpose. |
|------------|----------------------|

##### Returns

the transposed matrix.

Definition at line 207 of file **matrix.c**.

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

## 7.28 matrix.h

```

00001
00010 #ifndef _MATRIX_H_
00011 #define _MATRIX_H_
00012
00016 typedef struct _matrix {
00017     int height;
00018     int width;
00019     double *data;
00020 } matrix;
00021
00030 void assert(int assertion, const char *message);
00031
00035 matrix *makeMatrix(int width, int height);
00036
00044 matrix *copyMatrix(matrix *m);
00045
00050 void freeMatrix(matrix *m);
00051
00056 void printMatrix(matrix *m);
00057
00063 matrix *identityMatrix(int n);
00064
00070 double traceMatrix(matrix *m);
00071
00077 matrix *transposeMatrix(matrix *m);
00078
00086 matrix *meanMatrix(matrix *m);
00087
00094 matrix *multiplyMatrix(matrix *a, matrix *b);
00095
00103 matrix *scaleMatrix(matrix *m, double value);
00104
00111 matrix *covarianceMatrix(matrix *m);
00112
00122 void rowSwap(matrix *a, int p, int q);
00137 matrix *dotProductMatrix(matrix *a, matrix *b);
00138
00153 matrix *dotDiagonalMatrix(matrix *a, matrix *b);
00154
00155 #endif

```

## 7.29 include/menu.h File Reference

Contains menu functionality and abstraction.

```

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

```

### 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

- 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*

### 7.29.1 Detailed Description

Contains menu functionality and abstraction.

#### Author

Chris Jerrett

#### Date

9/9/2017

Definition in file **menu.h**.

### 7.29.2 Typedef Documentation

### 7.29.2.1 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. 98)
- menu\_t** (p. 15)
- create\_menu** (p. 200)
- `init_menu`
- display\_menu** (p. 201)
- menu\_type** (p. 100)
- denint\_menu** (p. 200)

## 7.29.3 Enumeration Type Documentation

### 7.29.3.1 menu\_type

```
enum menu_type
```

Represents the different types of menus.

#### Author

Chris Jerrett

#### Date

9/8/17

#### See also

- menu.h** (p. 98)
- menu\_t** (p. 15)
- create\_menu** (p. 200)
- `init_menu`
- display\_menu** (p. 101)
- 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 **30** of file **menu.h**.

## 7.29.4 Function Documentation

## 7.29.4.1 denint\_menu()

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

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

## Parameters

|             |                  |
|-------------|------------------|
| <i>menu</i> | the menu to free |
|-------------|------------------|

## See also

menu

## Author

Chris Jerrett

## Date

9/8/17

Definition at line **203** of file **menu.c**.

References **menu\_t::options**, and **menu\_t::prompt**.

## 7.29.4.2 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

|             |                     |
|-------------|---------------------|
| <i>menu</i> | the menu to display |
|-------------|---------------------|

## See also

**menu\_type** (p. 100)

## Author

Chris Jerrett

## Date

9/8/17

Definition at line **164** 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**.

## 7.29.4.3 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

|             |                  |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

## See also

**menu\_type** (p. 100)

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>min</i>    | the minimum value             |
| <i>max</i>    | the maximum value             |
| <i>step</i>   | the step value                |
| <i>prompt</i> | the prompt to display to user |



**Author**

Chris Jerrett

**Date**

9/8/17

Definition at line 111 of file **menu.c**.References **create\_menu()**, **max()**, **menu\_t::max\_f**, **min()**, **menu\_t::min\_f**, and **menu\_t::step\_f**.**7.29.4.4 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**

|             |                  |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

**See also****menu\_type** (p. 100)**Parameters**

|               |                               |
|---------------|-------------------------------|
| <i>min</i>    | the minimum value             |
| <i>max</i>    | the maximum value             |
| <i>step</i>   | the step value                |
| <i>prompt</i> | the prompt to display to user |

**Author**

Chris Jerrett

**Date**

9/8/17

Definition at line **88** of file **menu.c**.

References **create\_menu()**, **menu\_t::current**, **max()**, **menu\_t::max**, **min()**, **menu\_t::min**, and **menu\_t::step**.

#### 7.29.4.5 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

|             |                  |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

##### See also

**menu\_type** (p. 100)

##### Parameters

|                |   |
|----------------|---|
| <i>nums</i>    | the number of elements passed to function |
| <i>prompt</i>  | the prompt to display to user             |
| <i>options</i> | the options to display for user           |

##### Author

Chris Jerrett

##### Date

9/8/17

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

References **create\_menu()**, **menu\_t::length**, and **menu\_t::options**.

Referenced by **initialize()**.

## 7.30 menu.h

```

00001
00008 #ifndef _MENU_H_
00009 #define _MENU_H_
00010
00011 #include "API.h"
00012 #include "lcd.h"
00013 #include <float.h>
00014 #include <limits.h>
00015 #include <string.h>
00016 #include <vlib.h>
00017 #include "log.h"
00018
00030 enum menu_type {
00037     INT_TYPE,
00044     FLOAT_TYPE,
00050     STRING_TYPE
00051 };
00052
00066 typedef struct menu_t {
00072     enum menu_type type;
00073
00079     char **options;
00080
00086     unsigned int length;
00087
00094     int min;
00095
00102     int max;
00103
00111     int step;
00112
00119     float min_f;
00120
00127     float max_f;
00128
00136     float step_f;
00142     int current;
00150     char *prompt;
00151 } menu_t;
00152
00165 menu_t *init_menu_var(enum menu_type type, const char *prompt, int nums, ...);
00166
00180 menu_t *init_menu_int(enum menu_type type, int min, int max, int step,
00181                       const char *prompt);
00182
00196 menu_t *init_menu_float(enum menu_type type, float min, float max, float step,
00197                          const char *prompt);
00198
00209 int display_menu(menu_t *menu);
00210
00220 void denint_menu(menu_t *menu);
00221
00222 #endif

```

## 7.31 include/mobile\_goal\_intake.h File Reference

```

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

```

## Functions

- void **lower\_intake** ()  
*lowers the intake*
- void **raise\_intake** ()

*raises the intake*

- void **set\_intake\_motor** (int n)

*sets the intake motor*

- void **update\_intake** ()

*updates the mobile goal intake in teleop.*

### 7.31.1 Function Documentation

#### 7.31.1.1 lower\_intake()

```
void lower_intake ( )
```

lowers the intake

Definition at line 7 of file **mobile\_goal\_intake.c**.

References **set\_intake\_motor**().

Referenced by **autonomous()**, and **update\_intake()**.

#### 7.31.1.2 raise\_intake()

```
void raise_intake ( )
```

raises the intake

Definition at line 9 of file **mobile\_goal\_intake.c**.

References **set\_intake\_motor**().

Referenced by **autonomous()**, and **update\_intake()**.

#### 7.31.1.3 set\_intake\_motor()

```
void set_intake_motor (
    int n )
```

sets the intake motor

Author

Chris Jerrett

Definition at line 5 of file **mobile\_goal\_intake.c**.

References **INTAKE\_MOTOR**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**, **lower\_intake()**, **raise\_intake()**, and **update\_intake()**.

## 7.31.1.4 update\_intake()

```
void update_intake ( )
```

updates the mobile goal intake in teleop.

## Author

Chris Jerrett

Definition at line 14 of file **mobile\_goal\_intake.c**.

References **lower\_intake()**, **MASTER**, **raise\_intake()**, and **set\_intake\_motor()**.

Referenced by **operatorControl()**.

## 7.32 mobile\_goal\_intake.h

```
00001 #ifndef _MOBLE_GOAL_INTAKE_
00002 #define _MOBLE_GOAL_INTAKE_
00003
00004 #include "controller.h"
00005 #include "motor_ports.h"
00006 #include "slew.h"
00007
00013 void update_intake();
00014
00019 void set_intake_motor(int n);
00020
00024 void raise_intake();
00025
00029 void lower_intake();
00030
00031
00032 #endif
```

## 7.33 include/motor\_ports.h File Reference

The motor port definitions

Macros for the different motors ports.

## Macros

- **#define \_MOTOR\_PORTS\_H\_**
- **#define CLAW\_MOTOR 10**
- **#define INTAKE\_MOTOR 8**
- **#define MAX\_SPEED 127**  
*The max speed of a motor.*
- **#define MIN\_SPEED -128**  
*The min speed of a motor.*
- **#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**

### 7.33.1 Detailed Description

The motor port definitions

Macros for the different motors ports.

Definition in file **motor\_ports.h**.

### 7.33.2 Macro Definition Documentation

#### 7.33.2.1 \_MOTOR\_PORTS\_H\_

```
#define _MOTOR_PORTS_H_
```

Definition at line 7 of file **motor\_ports.h**.

#### 7.33.2.2 CLAW\_MOTOR

```
#define CLAW_MOTOR 10
```

Definition at line 62 of file **motor\_ports.h**.

Referenced by **close\_claw()**, **open\_claw()**, and **set\_claw\_motor()**.

### 7.33.2.3 INTAKE\_MOTOR

```
#define INTAKE_MOTOR 8
```

Definition at line **64** of file **motor\_ports.h**.

Referenced by **set\_intake\_motor()**.

### 7.33.2.4 MAX\_SPEED

```
#define MAX_SPEED 127
```

The max speed of a motor.

Definition at line **12** of file **motor\_ports.h**.

Referenced by **autonomous()**, **lower\_main\_lifter()**, **lower\_secondary\_lifter()**, **main\_lifter\_update()**, **raise\_↔  
main\_lifter()**, and **secondary\_lifter\_update()**.

### 7.33.2.5 MIN\_SPEED

```
#define MIN_SPEED -128
```

The min speed of a motor.

Definition at line **17** of file **motor\_ports.h**.

Referenced by **autonomous()**, **main\_lifter\_update()**, **raise\_secondary\_lifter()**, and **secondary\_lifter\_update()**.

### 7.33.2.6 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 **58** of file **motor\_ports.h**.

Referenced by **set\_side\_speed()**.

### 7.33.2.7 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 **52** of file **motor\_ports.h**.

Referenced by **set\_side\_speed()**.

### 7.33.2.8 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 **31** of file **motor\_ports.h**.

Referenced by **set\_side\_speed()**.

### 7.33.2.9 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 **24** of file **motor\_ports.h**.

Referenced by **set\_side\_speed()**.



#### 7.33.2.10 MOTOR\_LIFT

```
#define MOTOR_LIFT 9
```

Definition at line **60** of file **motor\_ports.h**.

Referenced by **set\_main\_lifter\_motors()**.

#### 7.33.2.11 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 **45** of file **motor\_ports.h**.

Referenced by **set\_side\_speed()**.

#### 7.33.2.12 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 **38** of file **motor\_ports.h**.

Referenced by **set\_side\_speed()**.

### 7.33.2.13 MOTOR\_SECONDARY\_LIFTER

```
#define MOTOR_SECONDARY_LIFTER 1
```

Definition at line **63** of file **motor\_ports.h**.

Referenced by **set\_secondary\_lifter\_motors()**.

## 7.34 motor\_ports.h

```
00001
00006 #ifndef _MOTOT_PORTS_H_
00007 #define _MOTOR_PORTS_H_
00008
00012 #define MAX_SPEED 127
00013
00017 #define MIN_SPEED -128
00018
00024 #define MOTOR_FRONT_RIGHT 2
00025
00031 #define MOTOR_FRONT_LEFT 7
00032
00038 #define MOTOR_MIDDLE_RIGHT 3
00039
00045 #define MOTOR_MIDDLE_LEFT 6
00046
00052 #define MOTOR_BACK_RIGHT 4
00053
00058 #define MOTOR_BACK_LEFT 5
00059
00060 #define MOTOR_LIFT 9
00061
00062 #define CLAW_MOTOR 10
00063 #define MOTOR_SECONDARY_LIFTER 1
00064 #define INTAKE_MOTOR 8
00065
00066 #endif
```

## 7.35 include/partner.h File Reference

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

### 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.*

### 7.35.1 Enumeration Type Documentation

#### 7.35.1.1 CONTROLL\_MODE

```
enum CONTROLL_MODE
```

## Enumerator

|                         |  |
|-------------------------|--|
| MAIN_CONTROLLER_MODE    |  |
| PARTNER_CONTROLLER_MODE |  |

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

## 7.35.2 Function Documentation

## 7.35.2.1 get\_mode()

```
enum CONTROLL_MODE get_mode ( )
```

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

References **mode**.

Referenced by **update\_drive\_motors()**.

## 7.35.2.2 update\_control()

```
void update_control ( )
```

Updates the controller mode between Driver and Partner modes.

## Author

Chris Jerrett

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

References **MAIN\_CONTROLLER\_MODE**, **mode**, **PARTNER**, and **PARTNER\_CONTROLLER\_MODE**.

## 7.36 partner.h

```
00001 #ifndef _PARTNER_H_
00002 #define _PARTNER_H_
00003
00004 #include "API.h"
00005 #include "controller.h"
00006
00007 enum CONTROLL_MODE { MAIN_CONTROLLER_MODE, PARTNER_CONTROLLER_MODE };
00013 void update_control();
00014
00015 enum CONTROLL_MODE get_mode();
00016
00017 #endif
```

## 7.37 include/potentiometer.h File Reference

### Macros

- `#define DEG_MAX 250.0`
- `#define TICK_MAX 4095.0`

#### 7.37.1 Macro Definition Documentation

##### 7.37.1.1 DEG\_MAX

```
#define DEG_MAX 250.0
```

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

Referenced by **lifterPotentiometerToDegree()**.

##### 7.37.1.2 TICK\_MAX

```
#define TICK_MAX 4095.0
```

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

Referenced by **lifterPotentiometerToDegree()**.

## 7.38 potentiometer.h

```
00001 #ifndef _POTENTIOMETER_H_
00002 #define _POTENTIOMETER_H_
00003
00004 #define TICK_MAX 4095.0
00005 #define DEG_MAX 250.0
00006
00007 #endif
```

## 7.39 include/routines.h File Reference

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

### Data Structures

- struct **routine\_t**

## Typedefs

- typedef struct **routine\_t** **routine\_t**

## Functions

- void **deinit\_routines** ()
- void **init\_routine** ()
- void **register\_routine** (void(\*routine)(), **button\_t** on\_buttons, **button\_t** \*prohibited\_buttons)
- void **routine\_task** ()

### 7.39.1 Typedef Documentation

#### 7.39.1.1 routine\_t

```
typedef struct routine_t routine_t
```

### 7.39.2 Function Documentation

#### 7.39.2.1 deinit\_routines()

```
void deinit_routines ( )
```

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

References **list\_destroy()**.

#### 7.39.2.2 init\_routine()

```
void init_routine ( )
```

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

References **list\_new()**, **routine\_task()**, and **routine\_task\_var**.

Referenced by **operatorControl()**.

### 7.39.2.3 register\_routine()

```
void register_routine (
    void(*) () routine,
    button_t on_buttons,
    button_t * prohibited_buttons )
```

Definition at line 35 of file **routines.c**.

References **routine\_t::blocked\_buttons**, **list\_node\_new()**, **list\_rpush()**, **routine\_t::on\_button**, **routine\_t::routine**, and **list\_node::val**.

### 7.39.2.4 routine\_task()

```
void routine_task ( )
```

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

References **buttonIsNewPress()**, **LIST\_HEAD**, **list\_iterator\_destroy()**, **list\_iterator\_new()**, **list\_iterator\_next()**, **routine\_t::on\_button**, **routine\_t::routine**, and **list\_node::val**.

Referenced by **init\_routine()**.

## 7.40 routines.h

```
00001 #include "controller.h"
00002
00003 typedef struct routine_t {
00004     button_t on_button;
00005     button_t* blocked_buttons;
00006     void(*routine) ();
00007 }routine_t;
00008
00009 void init_routine();
00010
00011 void routine_task();
00012
00013 void deinit_routines();
00014
00015 void register_routine(void(*routine) (), button_t on_buttons, button_t* prohibited_buttons);
```

## 7.41 include/sensor\_ports.h File Reference

### 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**

### 7.41.1 Macro Definition Documentation

#### 7.41.1.1 CLAW\_POT

```
#define CLAW_POT 1
```

Definition at line 20 of file **sensor\_ports.h**.

#### 7.41.1.2 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. 146)

Author

Christian Desimone

Date

9/7/2017

Definition at line 18 of file **sensor\_ports.h**.

#### 7.41.1.3 LIFTER

```
#define LIFTER 2
```

Definition at line 19 of file **sensor\_ports.h**.

Referenced by **getLifterTicks()**.

## 7.42 sensor\_ports.h

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

### 7.43 include/slew.h File Reference

Contains the slew rate controller wrapper for the motors.

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

#### 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.*

#### 7.43.1 Detailed Description

Contains the slew rate controller wrapper for the motors.

#### Author

Chris Jerrett

#### Date

9/14/17

Definition in file **slew.h**.



### 7.43.2 Macro Definition Documentation

#### 7.43.2.1 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**.

#### 7.43.2.2 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 **35** of file **slew.h**.

#### 7.43.2.3 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**.

### 7.43.3 Function Documentation

#### 7.43.3.1 deinitSlew()

```
void deinitSlew ( )
```

Deinitializes the slew rate controller and frees memory.

**Author**

Chris Jerrett

**Date**

9/14/17

Definition at line **59** of file **slew.c**.

References **initialized**, **motors\_curr\_speeds**, **motors\_set\_speeds**, and **slew**.

Referenced by **autonomous()**.

#### 7.43.3.2 init\_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

**Author**

Chris Jerrett, Christian DeSimone

**Date**

9/14/17

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

References **initialized**, **motors\_curr\_speeds**, **motors\_set\_speeds**, **slew**, **speeds\_mutex**, **updateMotors()**, and **warning()**.

Referenced by **autonomous()**, **operatorControl()**, **set\_motor\_immediate()**, and **set\_motor\_slew()**.

#### 7.43.3.3 set\_motor\_immediate()

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

Sets the motor speed ignoring the slew controller.

## Parameters

|              |  |
|--------------|--|
| <i>motor</i> | the motor port to use                  |
| <i>speed</i> | the speed to use, between -127 and 127 |

## Author

Chris Jerrett

## Date

9/14/17

Definition at line 90 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()**, and **set\_secondary\_lifter\_motors()**.

## 7.43.3.4 set\_motor\_slew()

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

Sets motor speed wrapped inside the slew rate controller.

## Parameters

|              |  |
|--------------|--|
| <i>motor</i> | the motor port to use                  |
| <i>speed</i> | the speed to use, between -127 and 127 |

## Author

Chris Jerrett

## Date

9/14/17

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

References **debug()**, **init\_slew()**, **initialized**, **motors\_set\_speeds**, and **speeds\_mutex**.

Referenced by **set\_main\_lifter\_motors()**, and **set\_side\_speed()**.

### 7.43.3.5 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 19 of file **slew.c**.

References **motors\_curr\_speeds**, **motors\_set\_speeds**, and **speeds\_mutex**.

Referenced by **init\_slew()**.

## 7.44 slew.h

```
00001
00008 #ifndef _SLEW_H_
00009 #define _SLEW_H_
00010
00011 #include <API.h>
00012 #include <math.h>
00013 #include <vlib.h>
00014
00020 #define UPDATE_PERIOD_MS 25
00021
00027 #define MOTOR_PORTS 12
00028
00035 #define RAMP_PROPORTION 1
00036
00043 void updateMotors();
00044
00050 void deinitSlew();
00051
00057 void init_slew();
00058
00066 void set_motor_slew(int motor, int speed);
00067
00075 void set_motor_immediate(int motor, int speed);
00076
00077 #endif
```

## 7.45 include/toggle.h File Reference

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

## 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.*

### 7.45.1 Function Documentation

#### 7.45.1.1 buttonGetState()

```
bool buttonGetState (  
    button_t )
```

Returns the current status of a button (pressed or not pressed)

#### Parameters

|               |  |
|---------------|--|
| <i>button</i> | The button to detect from the Buttons enumeration. |
|---------------|--|

#### Returns

true (pressed) or false (not pressed)

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

References **LCD\_CENT**, **LCD\_LEFT**, and **LCD\_RIGHT**.

Referenced by **buttonIsNewPress()**.

#### 7.45.1.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons.

Initializes the buttons.

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

References **buttonPressed**.

Referenced by **operatorControl()**.

#### 7.45.1.3 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

|               |  |
|---------------|--|
| <i>button</i> | 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

|               |  |
|---------------|--|
| <i>button</i> | 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 136 of file **toggle.c**.

References **buttonGetState()**, and **buttonPressed**.

Referenced by **routine\_task()**.

## 7.46 toggle.h

```
00001
00013 #ifndef BUTTONS_H_
00014 #define BUTTONS_H_
00015
00016 #include <API.h>
00017 #include "controller.h"
00018
00022 void buttonInit();
00023
00034 bool buttonIsNewPress(button_t);
00035
00044 bool buttonGetState(button_t);
00045
00046 #endif
```

## 7.47 include/vlib.h File Reference

Contains misc helpful functions.

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

## Functions

- void \* **calloc\_real** (size\_t elements, size\_t size)
- void **ftoaa** (float a, char \*buffer, int precision)  
*converts a float to string.*
- int **itaaa** (int a, char \*buffer, int digits)  
*converts a int to string.*
- void **reverse** (char \*str, int len)  
*reverses a string 'str' of length 'len'*

### 7.47.1 Detailed Description

Contains misc helpful functions.

#### Author

Chris Jerrett

#### Date

9/9/2017

Definition in file **vlib.h**.

### 7.47.2 Function Documentation

#### 7.47.2.1 calloc\_real()

```
void* calloc_real (
    size_t elements,
    size_t size )
```

#### 7.47.2.2 ftoaa()

```
void ftoaa (
    float a,
    char * buffer,
    int precision )
```

converts a float to string.



## Parameters

|                  |  |
|------------------|--|
| <i>a</i>         | the float                                |
| <i>buffer</i>    | the string the float will be written to. |
| <i>precision</i> | 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()`.

7.47.2.3 `itoaa()`

```
int itoaa (  
    int a,  
    char * buffer,  
    int digits )
```

converts a int to string.

## Parameters

|               |  |
|---------------|--|
| <i>a</i>      | the integer                            |
| <i>buffer</i> | the string the int will be written to. |
| <i>digits</i> | 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()**.**7.47.2.4 reverse()**

```
void reverse (  
    char * str,  
    int len )
```

reverses a string '*str*' of length '*len*'**Author**

Chris Jerrett

**Date**

9/9/2017

**Parameters**

|            |                       |
|------------|-----------------------|
| <i>str</i> | the string to reverse |
| <i>len</i> | the length            |

Definition at line **10** of file **vlib.c**.Referenced by **itoaa()**.**7.48 vlib.h**

```
00001  
00008 #ifndef _VLIB_H_  
00009 #define _VLIB_H_  
00010  
00011 #include <API.h>  
00012 #include <math.h>  
00013 #include <string.h>  
00014  
00022 void reverse(char *str, int len);  
00023  
00034 int itoaa(int a, char *buffer, int digits);  
00035  
00045 void ftoaa(float a, char *buffer, int precision);  
00046  
00047 void *calloc_real(size_t elements, size_t size);  
00048  
00049 #endif
```

## 7.49 include/vmath.h File Reference

Vex Specific Math Functions, includes: Cartesian to polar coordinates.

```
#include <math.h>
```

### 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 coordinated 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*

#### 7.49.1 Detailed Description

Vex Specific Math Functions, includes: Cartesian to polar coordinates.

### Author

Christian Desimone  
Chris Jerrett

### Date

9/9/2017

Definition in file **vmath.h**.

## 7.49.2 Macro Definition Documentation

### 7.49.2.1 M\_PI

```
#define M_PI 3.14159265358979323846
```

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

Referenced by **calculate\_encoder\_odometry()**, and **sind()**.

## 7.49.3 Function Documentation

### 7.49.3.1 cartesian\_cord\_to\_polar()

```
struct polar_cord cartesian_cord_to_polar (  
    struct cord cords )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

#### Author

Christian Desimone

#### Date

9/8/2017

#### Parameters

|              |                     |
|--------------|---------------------|
| <i>cords</i> | the cartesian cords |
|--------------|---------------------|

#### Returns

a struct containing the angle and magnitude.

#### See also

**polar\_cord** (p. 20)

**cord** (p. 7)

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

References **cartesian\_to\_polar()**.

### 7.49.3.2 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

|          |  |
|----------|--|
| <i>x</i> | float value of the x cartesian coordinate. |
| <i>y</i> | float value of the y cartesian coordinate. |

#### Returns

a struct containing the angle and magnitude.

#### See also

**polar\_cord** (p. 20)

Definition at line 15 of file **vmath.c**.

References **polar\_cord::angle**, and **polar\_cord::magnitue**.

Referenced by **cartesian\_cord\_to\_polar()**.

### 7.49.3.3 max()

```
int max (  
    int a,  
    int b )
```

the min of two values

#### Parameters

|          |            |
|----------|------------|
| <i>a</i> | the first  |
| <i>b</i> | the second |

**Returns**

the smaller of a and b

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

Referenced by **calculate\_current\_display()**, **init\_menu\_float()**, and **init\_menu\_int()**.

**7.49.3.4 min()**

```
int min (
    int a,
    int b )
```

the min of two values

**Parameters**

|          |            |
|----------|------------|
| <i>a</i> | the first  |
| <i>b</i> | the second |

**Returns**

the smaller of a and b

Definition at line **71** of file **vmath.c**.

Referenced by **calculate\_current\_display()**, **init\_menu\_float()**, and **init\_menu\_int()**.

**7.49.3.5 sind()**

```
double sind (
    double angle )
```

sine of a angle in degrees

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

References **M\_PI**.

## 7.50 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
00051 struct polar_cord cartesian_to_polar(float x, float y);
00052
00065 struct polar_cord cartesian_cord_to_polar(struct cord cords);
00066
00073 int min(int a, int b);
00074
00081 int max(int a, int b);
00082
00086 double sind(double angle);
00087 #endif

```

## 7.51 README.md File Reference

## 7.52 README.md

```

00001 # InTheZoneA
00002 Team A code for In The Zone

```

## 7.53 src/auto.c File Reference

File for autonomous code.

```

#include "auto.h"
#include "main.h"

```

### Functions

- void **autonomous** ()

### 7.53.1 Detailed Description

File for autonomous code.

This file should contain the user **autonomous()** (p. 134) 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**.

### 7.53.2 Function Documentation

#### 7.53.2.1 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.88) which should never exit. If it does so, the robot will await a switch to another mode or disable/enable cycle.

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

References **BOTH**, **calculate\_encoder\_angle()**, **close\_claw()**, **deinit\_slew()**, **DEPLOY\_HEIGHT**, **HALF\_ROTATE**, **init\_slew()**, **LEFT**, **lower\_intake()**, **LOWEST\_HEIGHT**, **MAIN\_LIFTER\_POT**, **MAX\_HEIGHT**, **MAX\_SPEED**, **MID\_LEFT\_DRIVE**, **MID\_RIGHT\_DRIVE**, **MIN\_SPEED**, **MOBILE\_GOAL\_DISTANCE**, **MOBILE\_GOAL\_HEIGHT**, **open\_claw()**, **raise\_intake()**, **RIGHT**, **SECONDARY\_LIFTER\_POT\_PORT**, **set\_claw\_motor()**, **set\_intake\_motor()**, **set\_main\_lifter\_motors()**, **set\_secondary\_lifter\_motors()**, **set\_side\_speed()**, and **ZONE\_DISTANCE**.

### 7.54 auto.c

```
00001
00014 #include "auto.h"
00015 #include "main.h"
00016
00017 /*
00018  * Runs the user autonomous code. This function will be started in its own task
00019  * with the default priority and stack size whenever the robot is enabled via
00020  * the Field Management System or the VEX Competition Switch in the autonomous
00021  * mode. If the robot is disabled or communications is lost, the autonomous
00022  * task will be stopped by the kernel. Re-enabling the robot will restart the
00023  * task, not re-start it from where it left off.
00024  *
00025  * Code running in the autonomous task cannot access information from the VEX
00026  * Joystick. However, the autonomous function can be invoked from another task
00027  * if a VEX Competition Switch is not available, and it can access joystick
00028  * information if called in this way.
00029  *
00030  * The autonomous task may exit, unlike operatorControl() which should never
00031  * exit. If it does so, the robot will await a switch to another mode or
00032  * disable/enable cycle.
00033  */
00034 void autonomous() {
00035     init_slew();
00036
00037     delay(10);
00038     printf("auto\n");
00039     // How far the left wheels have gone
00040     int counts_drive_left;
00041     // How far the right wheels have gone
00042     int counts_drive_right;
00043     // The average distance traveled forward
00044     int counts_drive;
```



```
00045
00046 // Reset the integrated motor controllers
00047 imeReset(MID_LEFT_DRIVE);
00048 imeReset(MID_RIGHT_DRIVE);
00049 // Set initial values for how far the wheels have gone
00050 imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00051 imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00052 counts_drive = counts_drive_left + counts_drive_right;
00053 counts_drive /= 2;
00054
00055 // Deploy claw
00056 while (analogRead(SECONDARY_LIFTER_POT_PORT) < DEPLOY_HEIGHT) {
00057     set_secondary_lifter_motors(MAX_SPEED);
00058 }
00059 set_secondary_lifter_motors(0);
00060
00061 while (analogRead(SECONDARY_LIFTER_POT_PORT) > LOWEST_HEIGHT) {
00062     set_secondary_lifter_motors(MIN_SPEED);
00063 }
00064 set_secondary_lifter_motors(0);
00065
00066 // Grab pre-load cone
00067 close_claw();
00068 delay(300);
00069 set_claw_motor(0);
00070
00071 while (analogRead(SECONDARY_LIFTER_POT_PORT) < MAX_HEIGHT) {
00072     set_secondary_lifter_motors(MAX_SPEED);
00073 }
00074 set_secondary_lifter_motors(0);
00075 // Raise the lifter
00076 while (analogRead(MAIN_LIFTER_POT) < MOBILE_GOAL_HEIGHT) {
00077     set_main_lifter_motors(MAX_SPEED);
00078 }
00079 set_main_lifter_motors(0);
00080 // Drive towards the goal
00081
00082 lower_intake();
00083 delay(300);
00084 set_intake_motor(0);
00085
00086 while (counts_drive < MOBILE_GOAL_DISTANCE) {
00087     set_side_speed(BOTH, 127);
00088     // Reestablish the distance traveled
00089     imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00090     imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00091     counts_drive = counts_drive_left + counts_drive_right;
00092     counts_drive /= 2;
00093 }
00094 // Stop moving
00095 set_side_speed(BOTH, 0);
00096 delay(1000);
00097
00098 raise_intake();
00099 delay(300);
00100 set_intake_motor(0);
00101
00102 // Drop the cone on the goal
00103 open_claw();
00104 delay(1000);
00105
00106 int ang = 0;
00107 while (ang < HALF_ROTATE) {
00108     ang += calculate_encoder_angle();
00109     set_side_speed(LEFT, MAX_SPEED);
00110     set_side_speed(RIGHT, MIN_SPEED);
00111 }
00112 set_side_speed(BOTH, 0);
00113
00114 counts_drive = 0;
00115
00116 while (counts_drive < MOBILE_GOAL_DISTANCE + ZONE_DISTANCE) {
00117     set_side_speed(BOTH, 127);
00118     // Reestablish the distance traveled
00119     imeGet(MID_LEFT_DRIVE, &counts_drive_left);
00120     imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
00121     counts_drive = counts_drive_left + counts_drive_right;
00122     counts_drive /= 2;
00123 }
00124
00125
```

```
00126   lower_intake();
00127   delay(300);
00128   set_intake_motor(0);
00129
00130   set_side_speed(BOTH, MIN_SPEED);
00131   delay(1000);
00132   set_side_speed(BOTH, 0);
00133
00134   deinit_slew();
00135 }
```

## 7.55 src/battery.c File Reference

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

### 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*

### 7.55.1 Function Documentation

#### 7.55.1.1 backup\_battery\_voltage()

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

#### Author

Chris Jerrett

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

Referenced by **battery\_level\_acceptable()**.

## 7.55.1.2 battery\_level\_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

See also

**MIN\_MAIN\_VOLTAGE** (p. 27)

**MIN\_BACKUP\_VOLTAGE** (p. 27)

Author

Chris Jerrett

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

References **backup\_battery\_voltage()**, **main\_battery\_voltage()**, **MIN\_BACKUP\_VOLTAGE**, and **MIN\_MAIN\_VOLTAGE**.

Referenced by **initialize()**.

## 7.55.1.3 main\_battery\_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

Author

Chris Jerrett

Definition at line 8 of file **battery.c**.

Referenced by **battery\_level\_acceptable()**.

## 7.56 battery.c

```
00001 #include "battery.h"
00002 #include <API.h>
00003
00008 double main_battery_voltage() { return powerLevelMain() / 1000.0; }
00009
00014 double backup_battery_voltage() { return powerLevelBackup() / 1000.0; }
00015
00023 bool battery_level_acceptable() {
00024     if (main_battery_voltage() < MIN_MAIN_VOLTAGE)
00025         return false;
00026     if (backup_battery_voltage() < MIN_BACKUP_VOLTAGE)
00027         return false;
00028     return true;
00029 }
```

## 7.57 src/claw.c File Reference

```
#include "claw.h"  
#include "log.h"  
#include "toggle.h"
```

### 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**

## 7.57.1 Function Documentation

### 7.57.1.1 close\_claw()

```
void close_claw ( )
```

Drives the motors to close the claw.

#### Author

Chris Jerrett

Definition at line **44** of file **claw.c**.

References **CLAW\_MOTOR**, **MIN\_CLAW\_SPEED**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**.

### 7.57.1.2 open\_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

#### Author

Chris Jerrett

Definition at line **38** of file **claw.c**.

References **CLAW\_MOTOR**, **MAX\_CLAW\_SPEED**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**.

### 7.57.1.3 set\_claw\_motor()

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

sets the claw motor speed

#### Author

Chris Jerrett

Definition at line **32** of file **claw.c**.

References **CLAW\_MOTOR**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**, and **update\_claw()**.

### 7.57.1.4 update\_claw()

```
void update_claw ( )
```

Updates the claw motor values.

#### Author

Chris Jerrett

Definition at line **10** 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()**.

## 7.57.2 Variable Documentation

### 7.57.2.1 state

```
enum claw_state state = CLAW_NEUTRAL_STATE [static]
```

Definition at line 4 of file **claw.c**.

Referenced by **update\_claw()**.

## 7.58 claw.c

```
00001 #include "claw.h"
00002 #include "log.h"
00003 #include "toggle.h"
00004 static enum claw_state state = CLAW_NEUTRAL_STATE;
00005
00010 void update_claw() {
00011     if (joystickGetDigital(CLAW_CLOSE)) {
00012         state = CLAW_CLOSE_STATE;
00013     } else if (joystickGetDigital(CLAW_OPEN)) {
00014         state = CLAW_OPEN_STATE;
00015     } else {
00016         state = CLAW_NEUTRAL_STATE;
00017     }
00018
00019     if (state == CLAW_CLOSE_STATE) {
00020         set_claw_motor(MAX_CLAW_SPEED);
00021     } else if (state == CLAW_OPEN_STATE) {
00022         set_claw_motor(MIN_CLAW_SPEED);
00023     } else {
00024         set_claw_motor(0);
00025     }
00026 }
00027
00032 void set_claw_motor(const int v) { set_motor_immediate(CLAW_MOTOR, v); }
00033
00038 void open_claw() { set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED); }
00039
00044 void close_claw() { set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED); }
```

## 7.59 src/controller.c File Reference

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

### Functions

- struct **cord** **get\_joystick\_cord** (enum **joystick side**, int controller)  
*Gets the location of a joystick on the controller.*

### 7.59.1 Function Documentation

## 7.59.1.1 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**.

## 7.60 controller.c

```
00001 #include "controller.h"
00002
00007 struct cord get_joystick_cord(enum joystick side, int controller) {
00008     int x;
00009     int y;
00010     // Get the joystick value for either the right or left,
00011     // depending on the mode
00012     if (side == RIGHT_JOY) {
00013         y = joystickGetAnalog(controller, RIGHT_JOY_X);
00014         x = joystickGetAnalog(controller, RIGHT_JOY_Y);
00015     } else {
00016         y = joystickGetAnalog(controller, LEFT_JOY_X);
00017         x = joystickGetAnalog(controller, LEFT_JOY_Y);
00018     }
00019     // Define a coordinate for the joystick value
00020     struct cord c;
00021     c.x = x;
00022     c.y = y;
00023     return c;
00024 }
```

## 7.61 src/drive.c File Reference

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

## Functions

- int **getThresh** ()  
*Gets the deadzone threshold 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 threshold on the drive.*
- void **update\_drive\_motors** ()  
*Updates the drive motors during teleop.*

## Variables

- static int **thresh** = 30

### 7.61.1 Function Documentation

#### 7.61.1.1 getThresh()

```
int getThresh ( )
```

Gets the deadzone threshold on the drive.

#### Author

Christian Desimone

Definition at line 15 of file **drive.c**.

References **thresh**.

#### 7.61.1.2 joystickExp()

```
static float joystickExp (  
    int joystickVal ) [static]
```

Applies exponential scale to a joystick value.

#### Author

Christian DeSimone, Chris Jerrett



## Parameters

|                    |                                    |
|--------------------|------------------------------------|
| <i>joystickVal</i> | the analog value from the joystick |
|--------------------|------------------------------------|

## Date

9/21/2017

Definition at line 81 of file **drive.c**.References **THRESHOLD**.

## 7.61.1.3 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

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

Definition at line 62 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()**.

## 7.61.1.4 setThresh()

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

Sets the deadzone threshold on the drive.

**Author**

Christian Desimone

Definition at line **21** of file **drive.c**.

References **thresh**.

**7.61.1.5 update\_drive\_motors()**

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

**Author**

Christian Desimone

**Date**

9/5/17

Definition at line **28** 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()**.

**7.61.2 Variable Documentation****7.61.2.1 thresh**

```
int thresh = 30 [static]
```

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

Referenced by **getThresh()**, **setThresh()**, and **update\_drive\_motors()**.

## 7.62 drive.c

```

00001 #include "drive.h"
00002 #include "controller.h"
00003 #include "log.h"
00004 #include "motor_ports.h"
00005 #include "slew.h"
00006 #include "vmath.h"
00007 #include <API.h>
00008
00009 static int thresh = 30;
00010
00015 int getThresh() { return thresh; }
00016
00021 void setThresh(int t) { thresh = t; }
00022
00028 void update_drive_motors() {
00029     // Get the joystick values from the controller
00030     int x = 0;
00031     int y = 0;
00032     if (get_mode() == PARTNER_CONTROLLER_MODE) {
00033         x = (joystickGetAnalog(PARTNER, 3));
00034         y = (joystickGetAnalog(PARTNER, 1));
00035     } else {
00036         x = -(joystickGetAnalog(MASTER, 3));
00037         y = (joystickGetAnalog(MASTER, 1));
00038     }
00039     // Make sure the joystick values are significant enough to change the motors
00040     if (x < thresh && x > -thresh) {
00041         x = 0;
00042     }
00043     if (y < thresh && y > -thresh) {
00044         y = 0;
00045     }
00046     // Create motor values for the left and right from the x and y of the joystick
00047     int r = (x + y);
00048     int l = -(x - y);
00049
00050     // Set the drive motors
00051     set_side_speed(LEFT, l);
00052     set_side_speed(RIGHT, -r);
00053 }
00054
00062 void set_side_speed(side_t side, int speed) {
00063     if (side == RIGHT || side == BOTH) {
00064         set_motor_slew(MOTOR_BACK_RIGHT, -speed);
00065         set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
00066         set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
00067     }
00068     if (side == LEFT || side == BOTH) {
00069         set_motor_slew(MOTOR_BACK_LEFT, speed);
00070         set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
00071         set_motor_slew(MOTOR_FRONT_LEFT, speed);
00072     }
00073 }
00074
00081 static float joystickExp(int joystickVal) {
00082     // make the offset negative if moving backwards
00083     if (abs(joystickVal) < THRESHOLD) {
00084         return 0;
00085     }
00086
00087     int offset;
00088     // Use the threshold to ensure the joystick values are significant
00089     if (joystickVal < 0) {
00090         offset = -(THRESHOLD);
00091     } else {
00092         offset = THRESHOLD;
00093     }
00094     // Apply the function (((x/10)^3)/18) + offset) * 0.8 to the joystick value
00095     return (pow(joystickVal / 10, 3) / 18 + offset) * 0.8;
00096 }

```

## 7.63 src/encoders.c File Reference

```

#include "encoders.h"
#include "log.h"

```

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

## 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.*

### 7.63.1 Function Documentation

#### 7.63.1.1 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 30 of file **encoders.c**.

Referenced by **calculate\_encoder\_angle()**, and **calculate\_encoder\_odometry()**.

#### 7.63.1.2 get\_encoder\_velocity()

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

Gets the encoder reads.

#### Author

Chris Jerrett

#### Date

9/15/2017

Definition at line 41 of file **encoders.c**.

### 7.63.1.3 init\_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

**Author**

Chris Jerrett

**Date**

9/9/2017

**See also**

**IME\_NUMBER** (p. 46)

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

References **error()**, and **IME\_NUMBER**.

Referenced by **initialize()**.

## 7.64 encoders.c

```
00001 #include "encoders.h"
00002 #include "log.h"
00003 #include <API.h>
00004
00011 bool init_encoders() {
00012     #ifdef IME_NUMBER
00013         int count = imeInitializeAll();
00014         if (count != IME_NUMBER) {
00015             printf("detected only %d\n", count);
00016             error("Wrong Number of IMEs Connected");
00017             return false;
00018         }
00019         return true;
00020     #else
00021         return imeInitializeAll();
00022     #endif
00023 }
00024
00030 int get_encoder_ticks(unsigned char address) {
00031     int i = 0;
00032     imeGet(address, &i);
00033     return i;
00034 }
00035
00041 int get_encoder_velocity(unsigned char address) {
00042     int i = 0;
00043     imeGetVelocity(address, &i);
00044     return i;
00045 }
```

## 7.65 src/gyro.c File Reference

```
#include "gyro.h"
```

## Functions

- float **get\_main\_gyro\_angluar\_velocity** ()
- bool **init\_main\_gyro** ()

## Variables

- static Gyro **main\_gyro**

### 7.65.1 Function Documentation

#### 7.65.1.1 **get\_main\_gyro\_angluar\_velocity**()

```
float get_main_gyro_angluar_velocity ( )
```

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

References **GYRO\_PORT**.

#### 7.65.1.2 **init\_main\_gyro**()

```
bool init_main_gyro ( )
```

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

References **GYRO\_MULTIPLIER**, **GYRO\_PORT**, and **main\_gyro**.

### 7.65.2 Variable Documentation

#### 7.65.2.1 **main\_gyro**

```
Gyro main_gyro [static]
```

Definition at line **3** of file **gyro.c**.

Referenced by **init\_main\_gyro**()

## 7.66 gyro.c

```

00001 #include "gyro.h"
00002
00003 static Gyro main_gyro;
00004
00005 bool init_main_gyro() {
00006     main_gyro = gyroInit(GYRO_PORT, GYRO_MULTIPLIER);
00007     return main_gyro != NULL;
00008 }
00009
00010 float get_main_gyro_angluar_velocity() {
00011     uint32_t port = GYRO_PORT;
00012     int32_t reading = (int32_t)analogReadCalibratedHR(port + 1);
00013     return 0;
00014 }

```

## 7.67 src/init.c File Reference

File for initialization code.

```

#include "battery.h"
#include "encoders.h"
#include "lcd.h"
#include "lifter.h"
#include "log.h"
#include "main.h"
#include "menu.h"
#include "slew.h"

```

## Functions

- void **initialize** ()
- void **initializeIO** ()

## Variables

- Ultrasonic **lifter\_ultrasonic**

## 7.67.1 Detailed Description

File for initialization code.

This file should contain the user **initialize()** (p. 150) 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**.

## 7.67.2 Function Documentation

### 7.67.2.1 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. 88) and **autonomous()** (p. 86) 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 **50** of file **init.c**.

References **battery\_level\_acceptable()**, **error()**, **info()**, **init\_encoders()**, **init\_error()**, **init\_main\_lcd()**, **init\_menu\_var()**, **lifter\_ultrasonic**, and **STRING\_TYPE**.

### 7.67.2.2 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 (`uartOpen()`) but cannot set up an LCD (`lcdInit()`).

Definition at line **35** of file **init.c**.

## 7.67.3 Variable Documentation

### 7.67.3.1 lifter\_ultrasonic

```
Ultrasonic lifter_ultrasonic
```

Definition at line **4** of file **lifter.c**.

Referenced by **autostack\_routine()**, **initialize()**, and **main\_lifter\_update()**.



## 7.68 init.c

```

00001
00013 #include "battery.h"
00014 #include "encoders.h"
00015 #include "lcd.h"
00016 #include "lifter.h"
00017 #include "log.h"
00018 #include "main.h"
00019 #include "menu.h"
00020 #include "slew.h"
00021
00022 extern Ultrasonic lifter_ultrasonic;
00023
00024 /*
00025  * Runs pre-initialization code. This function will be started in kernel mode
00026  * one time while the VEX Cortex is starting up. As the scheduler is still
00027  * paused, most API functions will fail.
00028  *
00029  * The purpose of this function is solely to set the default pin modes
00030  * (pinMode()) and port states (digitalWrite()) of limit switches, push buttons,
00031  * and solenoids. It can also safely configure a UART port (usartOpen()) but
00032  * cannot set up an LCD (lcdInit()).
00033  *
00034  */
00035 void initializeIO() { watchdogInit(); }
00036
00037 /*
00038  * Runs user initialization code. This function will be started in its own task
00039  * with the default priority and stack size once when the robot is starting up.
00040  * It is possible that the VEXnet communication link may not be fully
00041  * established at this time, so reading from the VEX Joystick may fail.
00042  *
00043  * This function should initialize most sensors (gyro, encoders, ultrasonics),
00044  * LCDs, global variables, and IMEs.
00045  *
00046  * This function must exit relatively promptly, or the operatorControl() and
00047  * autonomous() tasks will not start. An autonomous mode selection menu like the
00048  * pre_auton() in other environments can be implemented in this task if desired.
00049  */
00050 void initialize() {
00051     init_main_lcd(uart1);
00052     info("LCD Init");
00053     if (!battery_level_acceptable())
00054         error("Bad main/backup bat");
00055     menu_t *t =
00056         init_menu_var(StringType, "TEST Menu", 5, "1", "2", "3", "4", "5");
00057     init_error(true, uart2);
00058     setTeamName("9228A");
00059     init_encoders();
00060     lifter_ultrasonic = ultrasonicInit(4, 5);
00061 }

```

## 7.69 src/lcd.c File Reference

```
#include "lcd.h"
```

## 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 formatted string to a line on the lcd. Smilar to printf*
- void **lcd\_set\_backlight** (bool state)  
*sets the backlight of the lcd*
- void **prompt\_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

### 7.69.1 Function Documentation

#### 7.69.1.1 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

|            |                                 |
|------------|---------------------------------|
| <i>lcd</i> | the uart port of the lcd screen |
|------------|---------------------------------|

#### See also

uart1  
uart2

#### Author

Chris Jerrett

#### Date

9/9/2017

Definition at line 62 of file **lcd.c**.

References **lcd\_clear()**, and **lcd\_port**.

Referenced by **initialize()**.

### 7.69.1.2 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 **prompt\_confirmation()**.

### 7.69.1.3 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()**.

#### 7.69.1.4 `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. 9)

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 **prompt\_confirmation()**.

#### 7.69.1.5 `lcd_print()`

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

prints a string to a line on the lcd

##### Parameters

|             |                      |
|-------------|----------------------|
| <i>line</i> | the line to print on |
| <i>str</i>  | string to print      |

##### Author

Chris Jerrett

## Date

9/9/2017

Definition at line 75 of file **lcd.c**.References **lcd\_assert()**, and **lcd\_port**.Referenced by **display\_menu()**, and **prompt\_confirmation()**.

## 7.69.1.6 lcd\_printf()

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

prints a formatted string to a line on the lcd. Similar to printf

## Parameters

|                   |                               |
|-------------------|-------------------------------|
| <i>line</i>       | the line to print on          |
| <i>format_str</i> | format string string to print |

## Author

Chris Jerrett

## Date

9/9/2017

Definition at line 87 of file **lcd.c**.References **lcd\_assert()**, and **lcd\_port**.

## 7.69.1.7 lcd\_set\_backlight()

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

sets the backlight of the lcd

**Parameters**

|              |  |
|--------------|--|
| <i>state</i> | a boolean representing the state of the backlight. true = on, false = off. |
|--------------|--|

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line **99** of file **lcd.c**.

References **lcd\_assert()**, and **lcd\_port**.

**7.69.1.8 prompt\_confirmation()**

```
void prompt_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**

|                     |                                   |
|---------------------|-----------------------------------|
| <i>confirm_text</i> | the text for the user to confirm. |
|---------------------|-----------------------------------|

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line **113** of file **lcd.c**.

References **lcd\_assert()**, **lcd\_get\_pressed\_buttons()**, **lcd\_print()**, and **PRESSED**.

**7.69.2 Variable Documentation**

## 7.69.2.1 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()**.

## 7.70 lcd.c

```
00001 #include "lcd.h"
00002
00004 static FILE *lcd_port = NULL;
00005
00013 static bool lcd_assert() {
00014     if (lcd_port == NULL) {
00015         printf("LCD NULL!");
00016         return false;
00017     }
00018     return true;
00019 }
00020
00028 lcd_buttons lcd_get_pressed_buttons() {
00029     lcd_assert();
00030     unsigned int btn_binary = lcdReadButtons(lcd_port);
00031     bool left = btn_binary & 0x1; // 0001
00032     bool middle = btn_binary & 0x2; // 0010
00033     bool right = btn_binary & 0x4; // 0100
00034     lcd_buttons btns;
00035     btns.left = left ? PRESSED : RELEASED;
00036     btns.middle = middle ? PRESSED : RELEASED;
00037     btns.right = right ? PRESSED : RELEASED;
00038
00039     return btns;
00040 }
00041
00047 void lcd_clear() {
00048     lcd_assert();
00049     lcdClear(lcd_port);
00050 }
00051
00062 void init_main_lcd(FILE *lcd) {
00063     lcd_port = lcd;
00064     lcdInit(lcd);
00065     lcd_clear();
00066 }
00067
00075 void lcd_print(unsigned int line, const char *str) {
00076     lcd_assert();
00077     lcdSetText(lcd_port, line, str);
00078 }
00079
00087 void lcd_printf(unsigned int line, const char *format_str, ...) {
00088     lcd_assert();
00089     lcdPrint(lcd_port, line, format_str);
00090 }
00091
00099 void lcd_set_backlight(bool state) {
00100     lcd_assert();
00101     lcdSetBacklight(lcd_port, state);
00102 }
00103
00113 void prompt_confirmation(const char *confirm_text) {
00114     lcd_assert();
00115     lcd_print(1, confirm_text);
00116     while (lcd_get_pressed_buttons().middle != PRESSED) {
00117         delay(200);
00118     }
00119 }
```

## 7.71 src/lifter.c File Reference

```
#include "lifter.h"  
#include "log.h"
```

### Functions

- void **autostack\_routine** ()
- 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.*
- static void **main\_lifter\_update** ()
- void **raise\_main\_lifter** ()  
*Raises the main lifter.*
- void **raise\_secondary\_lifter** ()  
*Raises the main lifter.*
- static void **secondary\_lifter\_update** ()
- 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.*

### Variables

- static bool **lifter\_autostack\_routine\_interrupt** = false
- static bool **lifter\_autostack\_running** = false
- Ultrasonic **lifter\_ultrasonic**
- static bool **secondary\_override** = false

#### 7.71.1 Function Documentation



#### 7.71.1.1 autostack\_routine()

```
void autostack_routine ( )
```

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

References **lifter\_ultrasonic**, **raise\_main\_lifter()**, and **set\_main\_lifter\_motors()**.

#### 7.71.1.2 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 **216** of file **lifter.c**.

References **getLifterTicks()**.

#### 7.71.1.3 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 **207** of file **lifter.c**.

References **LIFTER**.

Referenced by **getLifterHeight()**.

#### 7.71.1.4 lifterPotentiometerToDegree()

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

height of the lifter in degrees from 0 height

**Parameters**

|                |               |
|----------------|---------------|
| <code>x</code> | the pot value |
|----------------|---------------|

**Returns**

the positions in degrees

**Author**

Chris Jerrett

**Date**

10/13/2017

Definition at line **196** of file **lifter.c**.

References **DEG\_MAX**, **INIT\_ROTATION**, and **TICK\_MAX**.

**7.71.1.5 lower\_main\_lifter()**

```
void lower_main_lifter ( )
```

Lowers the main lifter.

**Author**

Christian DeSimone

**Date**

9/12/2017

Definition at line **70** of file **lifter.c**.

References **MAX\_SPEED**, and **set\_main\_lifter\_motors()**.

#### 7.71.1.6 lower\_secondary\_lifter()

```
void lower_secondary_lifter ( )
```

Lowers the secondary lifter.

##### Author

Christian DeSimone

##### Date

9/12/2017

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

References **MAX\_SPEED**, and **set\_secondary\_lifter\_motors()**.

#### 7.71.1.7 main\_lifter\_update()

```
static void main_lifter_update ( ) [static]
```

Definition at line **90** of file **lifter.c**.

References **lifter\_autostack\_running**, **LIFTER\_DOWN**, **lifter\_ultrasonic**, **LIFTER\_UP**, **MAIN\_LIFTER\_POT**, **MAX\_SPEED**, **MIN\_SPEED**, **secondary\_override**, and **set\_main\_lifter\_motors()**.

Referenced by **update\_lifter()**.

#### 7.71.1.8 raise\_main\_lifter()

```
void raise_main_lifter ( )
```

Raises the main lifter.

##### Author

Christian DeSimone

##### Date

9/12/2017

Definition at line **62** of file **lifter.c**.

References **MAX\_SPEED**, and **set\_main\_lifter\_motors()**.

Referenced by **autostack\_routine()**.

### 7.71.1.9 `raise_secondary_lifter()`

```
void raise_secondary_lifter ( )
```

Raises the main lifter.

#### Author

Christian DeSimone

#### Date

9/12/2017

Definition at line **78** of file **lifter.c**.

References **MIN\_SPEED**, and **set\_secondary\_lifter\_motors()**.

### 7.71.1.10 `secondary_lifter_update()`

```
static void secondary_lifter_update ( ) [static]
```

Definition at line **130** of file **lifter.c**.

References **lifter\_autostack\_running**, **MAX\_SPEED**, **MIN\_SPEED**, **SECONDARY\_LIFTER\_D**, **SECONDARY\_LIFTER\_DOWN**, **SECONDARY\_LIFTER\_I**, **SECONDARY\_LIFTER\_P**, **SECONDARY\_LIFTER\_POT\_PORT**, **SECONDARY\_LIFTER\_UP**, and **set\_secondary\_lifter\_motors()**.

Referenced by **update\_lifter()**.

### 7.71.1.11 `set_lifter_pos()`

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

Sets the lifter positions to the given value.

#### Parameters

|            |                      |
|------------|----------------------|
| <i>pos</i> | The height in inches |
|------------|----------------------|

#### Author

Chris Jerrett

## Date

9/12/2017

Definition at line 54 of file **lifter.c**.

## 7.71.1.12 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 45 of file **lifter.c**.References **MOTOR\_LIFT**, and **set\_motor\_slew()**.Referenced by **autonomous()**, **autostack\_routine()**, **lower\_main\_lifter()**, **main\_lifter\_update()**, and **raise\_main\_lifter()**.

## 7.71.1.13 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 **33** of file **lifter.c**.

References **MOTOR\_SECONDARY\_LIFTER**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**, **lower\_secondary\_lifter()**, **raise\_secondary\_lifter()**, and **secondary\_lifter\_↔update()**.

**7.71.1.14 update\_lifter()**

```
void update_lifter ( )
```

Updates the lifter in teleop.

**Author**

Chris Jerrett

**Date**

9/9/2017

Definition at line **183** of file **lifter.c**.

References **main\_lifter\_update()**, **secondary\_lifter\_update()**, and **secondary\_override**.

Referenced by **operatorControl()**.

**7.71.2 Variable Documentation****7.71.2.1 lifter\_autostack\_routine\_interrupt**

```
bool lifter_autostack_routine_interrupt = false [static]
```

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

## 7.71.2.2 lifter\_autostack\_running

```
bool lifter_autostack_running = false [static]
```

Definition at line 6 of file **lifter.c**.

Referenced by **main\_lifter\_update()**, and **secondary\_lifter\_update()**.

## 7.71.2.3 lifter\_ultrasonic

```
Ultrasonic lifter_ultrasonic
```

Definition at line 4 of file **lifter.c**.

Referenced by **autostack\_routine()**, **initialize()**, and **main\_lifter\_update()**.

## 7.71.2.4 secondary\_override

```
bool secondary_override = false [static]
```

Definition at line 88 of file **lifter.c**.

Referenced by **main\_lifter\_update()**, and **update\_lifter()**.

## 7.72 lifter.c

```
00001 #include "lifter.h"
00002 #include "log.h"
00003
00004 Ultrasonic lifter_ultrasonic;
00005
00006 static bool lifter_autostack_running = false;
00007 static bool lifter_autostack_routine_interrupt = false;
00008
00009 void autostack_routine() {
00010     int instruction_couter = 0;
00011     bool routine_complete = false;
00012     while (true) {
00013         if (instruction_couter == 0) {
00014             int dist = ultrasonicGet(lifter_ultrasonic);
00015             if (dist > 11 || dist == -ULTRA_BAD_RESPONSE) {
00016                 raise_main_lifter();
00017             } else {
00018                 set_main_lifter_motors(0);
00019                 instruction_couter = 1;
00020             }
00021         }
00022     }
00023 }
00024
00033 void set_secondary_lifter_motors(const int v) {
00034     set_motor_immediate(MOTOR_SECONDARY_LIFTER, v);
00035 }
00036
00045 void set_main_lifter_motors(const int v) { set_motor_slew(MOTOR_LIFT, v); }
00046
00054 void set_lifter_pos(int pos) {}
```

```

00055
00062 void raise_main_lifter() { set_main_lifter_motors(MAX_SPEED); }
00063
00070 void lower_main_lifter() { set_main_lifter_motors(MAX_SPEED); }
00071
00078 void raise_secondary_lifter() { set_secondary_lifter_motors(MIN_SPEED / 1.5); }
00079
00086 void lower_secondary_lifter() { set_secondary_lifter_motors(MAX_SPEED); }
00087
00088 static bool secondary_override = false;
00089
00090 static void main_lifter_update() {
00091     if (lifter_autostack_running)
00092         return;
00093     static int count = 0;
00094     static bool pid_on = false;
00095     static int main_target = 0;
00096     int main_motor_speed = 0;
00097     static long long main_i = 0;
00098     if (count == 20) {
00099         main_target = analogRead(MAIN_LIFTER_POT);
00100     }
00101     if (pid_on && count > 20) {
00102         int curr = analogRead(MAIN_LIFTER_POT);
00103         static int main_last_p = 0;
00104         int main_p = curr - main_target;
00105         main_i += main_p;
00106         int main_d = main_last_p - main_p;
00107         // main_motor_speed = MAIN_LIFTER_P * main_p + MAIN_LIFTER_I * main_i +
00108         // MAIN_LIFTER_D * main_d;
00109         main_last_p = main_p;
00110     } else {
00111         main_i = 0;
00112         count++;
00113     }
00114
00115     if (joystickGetDigital(LIFTER_UP)) {
00116         int ultra = ultrasonicGet(lifter_ultrasonic);
00117         main_motor_speed = MAX_SPEED;
00118         count = 0;
00119     } else if (joystickGetDigital(LIFTER_DOWN)) {
00120         main_motor_speed = MIN_SPEED;
00121         count = 0;
00122         secondary_override = false;
00123     } else {
00124         secondary_override = false;
00125     }
00126     set_main_lifter_motors(main_motor_speed);
00127     pid_on = true;
00128 }
00129
00130 static void secondary_lifter_update() {
00131     if (lifter_autostack_running)
00132         return;
00133     static int count = 0;
00134     // static bool pid_on = false;
00135     static int second_target = 0;
00136     int second_motor_speed = 0;
00137     static long long second_i = 0;
00138
00139     if (count < 10) {
00140         second_target = analogRead(SECONDARY_LIFTER_POT_PORT);
00141         count++;
00142     }
00143
00144     int curr = analogRead(SECONDARY_LIFTER_POT_PORT);
00145     static int second_last_p = 0;
00146     int second_p = curr - second_target;
00147     second_i += second_p;
00148     int second_d = second_last_p - second_p;
00149     second_motor_speed = SECONDARY_LIFTER_P * second_p +
00150         SECONDARY_LIFTER_I * second_i +
00151         SECONDARY_LIFTER_D * second_d;
00152     second_last_p = second_p;
00153
00154     if (joystickGetDigital(SECONDARY_LIFTER_DOWN)) {
00155         second_motor_speed = MAX_SPEED;
00156         count = 0;
00157         second_i = 0;
00158         second_target = analogRead(SECONDARY_LIFTER_POT_PORT);
00159     } else if (joystickGetDigital(SECONDARY_LIFTER_UP)) {

```



```

00160     second_motor_speed = MIN_SPEED;
00161     count = 0;
00162     second_i = 0;
00163     second_target =
00164         second_target > 3000 ? 4095 : analogRead(SECONDARY_LIFTER_POT_PORT);
00165     ;
00166 } else {
00167     second_target = second_target > 3000 ? 4095 : second_target;
00168 }
00169 second_motor_speed = abs(second_motor_speed) < 20 ? 0 : second_motor_speed;
00170 /*printf("Motor %d \n", second_motor_speed);
00171 printf("P %d \n", second_p);
00172 printf("I %ld \n", second_i);
00173 printf("D %d \n", second_d);*/
00174 set_secondary_lifter_motors(second_motor_speed);
00175 }
00176
00183 void update_lifter() {
00184     main_lifter_update();
00185     if (!secondary_override)
00186         secondary_lifter_update();
00187 }
00196 float lifterPotentiometerToDegree(int x) {
00197     return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
00198 }
00199
00207 int getLifterTicks() { return analogRead(LIFTER); }
00208
00216 double getLifterHeight() {
00217     unsigned int ticks = getLifterTicks();
00218     return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) +
00219         0.0198 * ticks + 2.3033);
00220 }

```

## 7.73 src/list.c File Reference

```
#include "list.h"
```

### Functions

- **list\_node\_t \* list\_at** ( list\_t \*self, int index)
- void **list\_destroy** ( list\_t \*self)
- **list\_node\_t \* list\_find** ( list\_t \*self, void \*val)
- **list\_node\_t \* list\_lpop** ( list\_t \*self)
- **list\_node\_t \* list\_lpush** ( list\_t \*self, list\_node\_t \*node)
- **list\_t \* list\_new** ()
- void **list\_remove** ( list\_t \*self, list\_node\_t \*node)
- **list\_node\_t \* list\_rpop** ( list\_t \*self)
- **list\_node\_t \* list\_rpush** ( list\_t \*self, list\_node\_t \*node)

#### 7.73.1 Function Documentation

#### 7.73.1.1 list\_at()

```
list_node_t* list_at (
    list_t * self,
    int index )
```

Definition at line **162** of file **list.c**.

References **LIST\_HEAD**, **list\_iterator\_destroy()**, **list\_iterator\_new()**, **list\_iterator\_next()**, and **LIST\_TAIL**.

#### 7.73.1.2 list\_destroy()

```
void list_destroy (
    list_t * self )
```

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

References **LIST\_FREE**, **list\_node::next**, and **list\_node::val**.

Referenced by **deinit\_routines()**.

#### 7.73.1.3 list\_find()

```
list_node_t* list_find (
    list_t * self,
    void * val )
```

Definition at line **136** of file **list.c**.

References **LIST\_HEAD**, **list\_iterator\_destroy()**, **list\_iterator\_new()**, **list\_iterator\_next()**, and **list\_node::val**.

#### 7.73.1.4 list\_lpop()

```
list_node_t* list_lpop (
    list_t * self )
```

Definition at line **93** of file **list.c**.

References **list\_node::next**, and **list\_node::prev**.

## 7.73.1.5 list\_lpush()

```
list_node_t* list_lpush (
    list_t * self,
    list_node_t * node )
```

Definition at line 114 of file **list.c**.

References **list\_node::next**, and **list\_node::prev**.

## 7.73.1.6 list\_new()

```
list_t* list_new ( )
```

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

References **list\_t::head**, and **LIST\_MALLOC**.

Referenced by **init\_routine()**.

## 7.73.1.7 list\_remove()

```
void list_remove (
    list_t * self,
    list_node_t * node )
```

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

References **LIST\_FREE**, **list\_node::next**, **list\_node::prev**, and **list\_node::val**.

## 7.73.1.8 list\_rpop()

```
list_node_t* list_rpop (
    list_t * self )
```

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

References **list\_node::next**, and **list\_node::prev**.

## 7.73.1.9 list\_rpush()

```
list_node_t* list_rpush (
    list_t * self,
    list_node_t * node )
```

Definition at line 51 of file list.c.

References `list_node::next`, and `list_node::prev`.

Referenced by `register_routine()`.

## 7.74 list.c

```
00001
00002 //
00003 // list.c
00004 //
00005 // Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
00006 //
00007
00008 #include "list.h"
00009
00010 /*
00011  * Allocate a new list_t. NULL on failure.
00012  */
00013
00014 list_t *list_new() {
00015     list_t *self;
00016     if (!(self = LIST_MALLOC(sizeof(list_t))))
00017         return NULL;
00018     self->head = NULL;
00019     self->tail = NULL;
00020     self->free = NULL;
00021     self->match = NULL;
00022     self->len = 0;
00023     return self;
00024 }
00025
00026 /*
00027  * Free the list.
00028  */
00029
00030 void list_destroy(list_t *self) {
00031     unsigned int len = self->len;
00032     list_node_t *next;
00033     list_node_t *curr = self->head;
00034
00035     while (len--) {
00036         next = curr->next;
00037         if (self->free)
00038             self->free(curr->val);
00039         LIST_FREE(curr);
00040         curr = next;
00041     }
00042
00043     LIST_FREE(self);
00044 }
00045
00046 /*
00047  * Append the given node to the list
00048  * and return the node, NULL on failure.
00049  */
00050
00051 list_node_t *list_rpush(list_t *self, list_node_t *node) {
00052     if (!node)
00053         return NULL;
00054
00055     if (self->len) {
00056         node->prev = self->tail;
00057         node->next = NULL;
```

```

00058     self->tail->next = node;
00059     self->tail = node;
00060 } else {
00061     self->head = self->tail = node;
00062     node->prev = node->next = NULL;
00063 }
00064
00065 ++self->len;
00066 return node;
00067 }
00068
00069 /*
00070  * Return / detach the last node in the list, or NULL.
00071  */
00072
00073 list_node_t *list_rpop(list_t *self) {
00074     if (!self->len)
00075         return NULL;
00076
00077     list_node_t *node = self->tail;
00078
00079     if (--self->len) {
00080         (self->tail = node->prev)->next = NULL;
00081     } else {
00082         self->tail = self->head = NULL;
00083     }
00084
00085     node->next = node->prev = NULL;
00086     return node;
00087 }
00088
00089 /*
00090  * Return / detach the first node in the list, or NULL.
00091  */
00092
00093 list_node_t *list_lpop(list_t *self) {
00094     if (!self->len)
00095         return NULL;
00096
00097     list_node_t *node = self->head;
00098
00099     if (--self->len) {
00100         (self->head = node->next)->prev = NULL;
00101     } else {
00102         self->head = self->tail = NULL;
00103     }
00104
00105     node->next = node->prev = NULL;
00106     return node;
00107 }
00108
00109 /*
00110  * Prepend the given node to the list
00111  * and return the node, NULL on failure.
00112  */
00113
00114 list_node_t *list_lpush(list_t *self, list_node_t *node) {
00115     if (!node)
00116         return NULL;
00117
00118     if (self->len) {
00119         node->next = self->head;
00120         node->prev = NULL;
00121         self->head->prev = node;
00122         self->head = node;
00123     } else {
00124         self->head = self->tail = node;
00125         node->prev = node->next = NULL;
00126     }
00127
00128     ++self->len;
00129     return node;
00130 }
00131
00132 /*
00133  * Return the node associated to val or NULL.
00134  */
00135
00136 list_node_t *list_find(list_t *self, void *val) {
00137     list_iterator_t *it = list_iterator_new(self, LIST_HEAD);
00138     list_node_t *node;

```

```

00139
00140 while ((node = list_iterator_next(it))) {
00141     if (self->match) {
00142         if (self->match(val, node->val)) {
00143             list_iterator_destroy(it);
00144             return node;
00145         }
00146     } else {
00147         if (val == node->val) {
00148             list_iterator_destroy(it);
00149             return node;
00150         }
00151     }
00152 }
00153
00154 list_iterator_destroy(it);
00155 return NULL;
00156 }
00157
00158 /*
00159  * Return the node at the given index or NULL.
00160  */
00161
00162 list_node_t *list_at(list_t *self, int index) {
00163     list_direction_t direction = LIST_HEAD;
00164
00165     if (index < 0) {
00166         direction = LIST_TAIL;
00167         index = ~index;
00168     }
00169
00170     if ((unsigned)index < self->len) {
00171         list_iterator_t *it = list_iterator_new(self, direction);
00172         list_node_t *node = list_iterator_next(it);
00173         while (index--)
00174             node = list_iterator_next(it);
00175         list_iterator_destroy(it);
00176         return node;
00177     }
00178
00179     return NULL;
00180 }
00181
00182 /*
00183  * Remove the given node from the list, freeing it and it's value.
00184  */
00185
00186 void list_remove(list_t *self, list_node_t *node) {
00187     node->prev ? (node->prev->next = node->next) : (self->head = node->next);
00188     node->next ? (node->next->prev = node->prev) : (self->tail = node->prev);
00189
00190     if (self->free)
00191         self->free(node->val);
00192
00193     LIST_FREE(node);
00194     --self->len;
00195 }
00196 }

```

## 7.75 src/list\_iterator.c File Reference

```
#include "list.h"
```

### Functions

- void **list\_iterator\_destroy** ( list\_iterator\_t \*self)
- list\_iterator\_t \* **list\_iterator\_new** ( list\_t \*list, list\_direction\_t direction)
- list\_iterator\_t \* **list\_iterator\_new\_from\_node** ( list\_node\_t \*node, list\_direction\_t direction)
- list\_node\_t \* **list\_iterator\_next** ( list\_iterator\_t \*self)

### 7.75.1 Function Documentation

#### 7.75.1.1 list\_iterator\_destroy()

```
void list_iterator_destroy (  
    list_iterator_t * self )
```

Definition at line 52 of file `list_iterator.c`.

References `LIST_FREE`.

Referenced by `list_at()`, `list_find()`, and `routine_task()`.

#### 7.75.1.2 list\_iterator\_new()

```
list_iterator_t* list_iterator_new (  
    list_t * list,  
    list_direction_t direction )
```

Definition at line 15 of file `list_iterator.c`.

References `list_t::head`, `LIST_HEAD`, `list_iterator_new_from_node()`, and `list_t::tail`.

Referenced by `list_at()`, `list_find()`, and `routine_task()`.

#### 7.75.1.3 list\_iterator\_new\_from\_node()

```
list_iterator_t* list_iterator_new_from_node (  
    list_node_t * node,  
    list_direction_t direction )
```

Definition at line 25 of file `list_iterator.c`.

References `LIST_MALLOC`, and `list_iterator_t::next`.

Referenced by `list_iterator_new()`.

#### 7.75.1.4 list\_iterator\_next()

```
list_node_t* list_iterator_next (  
    list_iterator_t * self )
```

Definition at line 40 of file `list_iterator.c`.

References `LIST_HEAD`, `list_node::next`, and `list_node::prev`.

Referenced by `list_at()`, `list_find()`, and `routine_task()`.

## 7.76 list\_iterator.c

```

00001
00002 //
00003 // iterator.c
00004 //
00005 // Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
00006 //
00007
00008 #include "list.h"
00009
00010 /*
00011  * Allocate a new list_iterator_t. NULL on failure.
00012  * Accepts a direction, which may be LIST_HEAD or LIST_TAIL.
00013  */
00014
00015 list_iterator_t *list_iterator_new(list_t *list, list_direction_t direction) {
00016     list_node_t *node = direction == LIST_HEAD ? list->head : list->tail;
00017     return list_iterator_new_from_node(node, direction);
00018 }
00019
00020 /*
00021  * Allocate a new list_iterator_t with the given start
00022  * node. NULL on failure.
00023  */
00024
00025 list_iterator_t *list_iterator_new_from_node(list_node_t *node,
00026                                             list_direction_t direction) {
00027     list_iterator_t *self;
00028     if (!(self = LIST_MALLOC(sizeof(list_iterator_t))))
00029         return NULL;
00030     self->next = node;
00031     self->direction = direction;
00032     return self;
00033 }
00034
00035 /*
00036  * Return the next list_node_t or NULL when no more
00037  * nodes remain in the list.
00038  */
00039
00040 list_node_t *list_iterator_next(list_iterator_t *self) {
00041     list_node_t *curr = self->next;
00042     if (curr) {
00043         self->next = self->direction == LIST_HEAD ? curr->next : curr->prev;
00044     }
00045     return curr;
00046 }
00047
00048 /*
00049  * Free the list iterator.
00050  */
00051
00052 void list_iterator_destroy(list_iterator_t *self) {
00053     LIST_FREE(self);
00054     self = NULL;
00055 }

```

## 7.77 src/list\_node.c File Reference

```
#include "list.h"
```

### Functions

- `list_node_t * list_node_new (void *val)`



## 7.77.1 Function Documentation

## 7.77.1.1 list\_node\_new()

```
list_node_t* list_node_new (
    void * val )
```

Definition at line 14 of file **list\_node.c**.

References **LIST\_MALLOC**, **list\_node::next**, **list\_node::prev**, and **list\_node::val**.

Referenced by **register\_routine()**.

## 7.78 list\_node.c

```
00001
00002 //
00003 // node.c
00004 //
00005 // Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
00006 //
00007
00008 #include "list.h"
00009
00010 /*
00011  * Allocates a new list_node_t. NULL on failure.
00012  */
00013
00014 list_node_t *list_node_new(void *val) {
00015     list_node_t *self;
00016     if (!(self = LIST_MALLOC(sizeof(list_node_t))))
00017         return NULL;
00018     self->prev = NULL;
00019     self->next = NULL;
00020     self->val = val;
00021     return self;
00022 }
```

## 7.79 src/localization.c File Reference

```
#include "localization.h"
#include "vmath.h"
#include <inttypes.h>
```

## Data Structures

- struct **accelerometer\_odometry**
- struct **encoder\_odometry**

## Macros

- `#define CPR 392.0`
- `#define CPR 392.0`
- `#define WHEEL_RADIUS 2`
- `#define WHEEL_RADIUS 2`
- `#define WIDTH 13.5`
- `#define WIDTH 13.5`

## Functions

- static struct **accelerometer\_odometry** **calculate\_accelerometer\_odometry** ()
- static double **calculate\_angle** ()
- int **calculate\_encoder\_angle** ()
- static void **calculate\_encoder\_odometry** ()
- static double **calculate\_gyro\_angular\_velocity** ()
- struct **location** **get\_position** ()  
*Gets the current position 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**

### 7.79.1 Macro Definition Documentation

#### 7.79.1.1 CPR [1/2]

```
#define CPR 392.0
```

Referenced by **calculate\_encoder\_angle()**, and **calculate\_encoder\_odometry()**.

#### 7.79.1.2 CPR [2/2]

```
#define CPR 392.0
```

### 7.79.1.3 WHEEL\_RADIUS [1/2]

```
#define WHEEL_RADIUS 2
```

### 7.79.1.4 WHEEL\_RADIUS [2/2]

```
#define WHEEL_RADIUS 2
```

### 7.79.1.5 WIDTH [1/2]

```
#define WIDTH 13.5
```

Referenced by **calculate\_encoder\_angle()**, and **calculate\_encoder\_odometry()**.

### 7.79.1.6 WIDTH [2/2]

```
#define WIDTH 13.5
```

## 7.79.2 Function Documentation

### 7.79.2.1 calculate\_accelerometer\_odometry()

```
static struct accelerometer_odometry calculate_accelerometer_odometry ( ) [static]
```

Definition at line **59** of file **localization.c**.

References **last\_call**.

Referenced by **update\_position()**.

### 7.79.2.2 calculate\_angle()

```
static double calculate_angle ( ) [static]
```

### 7.79.2.3 calculate\_encoder\_angle()

```
int calculate_encoder_angle ( )
```

Definition at line 101 of file **localization.c**.

References **CPR**, **get\_encoder\_ticks()**, and **WIDTH**.

Referenced by **autonomous()**.

### 7.79.2.4 calculate\_encoder\_odemetry()

```
static void calculate_encoder_odemetry ( ) [static]
```

Definition at line 110 of file **localization.c**.

References **CPR**, **get\_encoder\_ticks()**, **M\_PI**, **encoder\_odemtry::theta**, and **WIDTH**.

### 7.79.2.5 calculate\_gryo\_anglular\_velocity()

```
static double calculate_gryo_anglular_velocity ( ) [static]
```

Definition at line 93 of file **localization.c**.

References **g1**, and **LOCALIZATION\_UPDATE\_FREQUENCY**.

### 7.79.2.6 get\_position()

```
struct location get_position ( )
```

Gets the current positiuon of the robot.

#### Parameters

|              |                |
|--------------|----------------|
| <i>gyro1</i> | The first gyro |
|--------------|----------------|

#### Returns

the loacation of the robot as a struct.

Definition at line 32 of file **localization.c**.

## 7.79.2.7 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

|              |   |
|--------------|---|
| <i>gyro1</i> | 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 123 of file **localization.c**.

References **g1**, **last\_call**, **localization\_task**, **LOCALIZATION\_UPDATE\_FREQUENCY**, **makeMatrix()**, and **update\_position()**.

## 7.79.2.8 integrate\_gyro\_w()

```
static double integrate_gyro_w (
    int new_w ) [static]
```

Definition at line 87 of file **localization.c**.

References **LOCALIZATION\_UPDATE\_FREQUENCY**, and **encoder\_odemtry::theta**.

## 7.79.2.9 update\_position()

```
void update_position ( )
```

Updates the position from the localization.

## Author

Chris Jerrett

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

References **calculate\_accelerometer\_odometry()**, and **last\_call**.

Referenced by **init\_localization()**.

### 7.79.3 Variable Documentation

#### 7.79.3.1 g1

```
Gyro g1 [static]
```

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

Referenced by **calculate\_gryo\_angular\_velocity()**, and **init\_localization()**.

#### 7.79.3.2 last\_call

```
int last_call = 0 [static]
```

Definition at line **8** of file **localization.c**.

Referenced by **calculate\_accelerometer\_odometry()**, **init\_localization()**, and **update\_position()**.

#### 7.79.3.3 localization\_task

```
TaskHandle localization_task [static]
```

Definition at line **6** of file **localization.c**.

Referenced by **init\_localization()**.

#### 7.79.3.4 state\_matrix

```
matrix* state_matrix
```

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

## 7.80 localization.c

```

00001 #include "localization.h"
00002 #include "vmath.h"
00003 #include <inttypes.h>
00004
00005 static Gyro g1;
00006 static TaskHandle localization_task;
00007
00008 static int last_call = 0;
00009
00010 matrix *state_matrix;
00011
00012 struct encoder_odometry {
00013     double x;
00014     double y;
00015     double theta;
00016 };
00017
00018 struct accelerometer_odometry {
00019     double x;
00020     double y;
00021 };
00022
00023 static double calculate_angle();
00024 static struct accelerometer_odometry calculate_accelerometer_odometry();
00025
00032 struct location get_position() {}
00033
00039 void update_position() {
00040     // int curr_theta = calculate_angle();
00041
00042     struct accelerometer_odometry oddem = calculate_accelerometer_odometry();
00043     // printf("x: %d y: %d T: %d\n", a.x, a.y, 0);
00044
00045     /*int l = 1;
00046     int vr = get_encoder_velocity(1);
00047     int vl = get_encoder_velocity(2);
00048     int theta_dot = (vr - vl) / l;
00049     int curr_theta = theta + theta_dot;
00050     double dt = LOCALIZATION_UPDATE_FREQUENCY;
00051     double v_tot = (vr+vl)/2.0;
00052     int x_curr = x - v_tot*dt*sin(curr_theta);
00053     int y_curr = y + v_tot*dt*cos(curr_theta);
00054     x = x_curr;
00055     y = y_curr;*/
00056     last_call = millis();
00057 }
00058
00059 static struct accelerometer_odometry calculate_accelerometer_odometry() {
00060     static double vel_acumm_x = 0;
00061     static double vel_acumm_y = 0;
00062
00063     int32_t accel_x_rel = (int32_t)analogReadCalibratedHR(2);
00064     int32_t accel_y_rel = (int32_t)analogReadCalibratedHR(3);
00065
00066     // Ignore atom format string errors
00067     printf("x: %" PRId32 " y: %" PRId32 "\n", accel_x_rel, accel_y_rel);
00068
00069     double delta_time = (millis() - last_call) / 1000.0;
00070     // double accel_x_abs = (accel_x_rel * cos(theta) + accel_y_rel * sin(theta))
00071     // * delta_time; double accel_y_abs = (accel_y_rel * cos(theta) +
00072     // accel_x_rel
00073     // * sin(theta)) * delta_time;
00074
00075     // vel_acumm_x += accel_x_abs;
00076     // vel_acumm_y += accel_y_abs;
00077
00078     // double new_x = x + vel_acumm_x * delta_time;
00079     // double new_y = y + vel_acumm_y * delta_time;
00080
00081     struct accelerometer_odometry od;
00082     // od.x = new_x;
00083     // od.y = new_y;
00084     return od;
00085 }
00086
00087 static double integrate_gyro_w(int new_w) {
00088     static double theta = 0;
00089     double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;

```

```

00090  theta += delta_theta;
00091  }
00092
00093  static double calculate_gryo_anglular_velocity() {
00094      static int last_gyro = 0;
00095      int current = gyroGet(g1);
00096      // Calculate w (anglular velocity in degrees per second)
00097      double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY / 1000.0);
00098      return w;
00099  }
00100
00101  int calculate_encoder_angle() {
00102      #define WIDTH 13.5
00103      #define CPR 392.0
00104      #define WHEEL_RADIUS 2
00105      int dist_r = get_encoder_ticks(0) / CPR;
00106      int dist_l = get_encoder_ticks(1) / CPR;
00107      return ((dist_r - dist_l) / WIDTH);
00108  }
00109
00110  static void calculate_encoder_odometry() {
00111      #define WIDTH 13.5
00112      #define CPR 392.0
00113      #define WHEEL_RADIUS 2
00114
00115      int dist_r = get_encoder_ticks(0) / CPR;
00116      int dist_l = get_encoder_ticks(1) / CPR;
00117      printf("dist_r: %d dist_l: %d\n", dist_r, dist_l);
00118      int theta = (dist_l - dist_r) / WIDTH;
00119      printf("theta: %d\n", theta);
00120      int arc_length = ((M_PI * theta) * (WIDTH * WIDTH) / (8));
00121  }
00122
00123  bool init_localization(const unsigned char gyro1, unsigned short multiplier,
00124                        int start_x, int start_y, int start_theta) {
00125      g1 = gyroInit(gyro1, multiplier);
00126      // init state matrix
00127
00128      // one dimensional vector with x, y, theta, acceleration in x and y
00129      state_matrix = makeMatrix(1, 5);
00130      localization_task =
00131          taskRunLoop(update_position, LOCALIZATION_UPDATE_FREQUENCY * 1000);
00132      last_call = millis();
00133      return true;
00134  }

```

## 7.81 src/log.c File Reference

```
#include "log.h"
```

### 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**

### 7.81.1 Function Documentation

#### 7.81.1.1 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. 186)

#### Parameters

|                      |             |
|----------------------|-------------|
| <i>debug_message</i> | the message |
|----------------------|-------------|

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

References **ERROR**, and **log\_level**.

Referenced by **set\_motor\_immediate()**, and **set\_motor\_slew()**.

#### 7.81.1.2 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. 186)

#### Author

Chris Jerrett

#### Date

9/10/17

**Parameters**

|                      |             |
|----------------------|-------------|
| <i>error_message</i> | the message |
|----------------------|-------------|

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

References **log\_info()**, **log\_level**, and **NONE**.

Referenced by **assert()**, **create\_menu()**, **init\_encoders()**, and **initialize()**.

**7.81.1.3 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. 186)

**Parameters**

|                     |             |
|---------------------|-------------|
| <i>info_message</i> | the message |
|---------------------|-------------|

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

References **ERROR**, **log\_info()**, and **log\_level**.

Referenced by **initialize()**.

**7.81.1.4 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

|                |                        |
|----------------|------------------------|
| <i>use_lcd</i> | whether to use the lcd |
| <i>lcd</i>     | the lcd                |

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

References **log\_lcd**.

Referenced by **initialize()**.

## 7.81.1.5 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()**.

## 7.81.1.6 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. 186)

## Author

Chris Jerrett

## Date

9/10/17

**Parameters**

|                        |             |
|------------------------|-------------|
| <i>warning_message</i> | the message |
|------------------------|-------------|

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

References **log\_info()**, **log\_level**, and **WARNING**.

Referenced by **init\_slew()**.

**7.81.2 Variable Documentation****7.81.2.1 log\_lcd**

```
FILE* log_lcd = NULL [static]
```

Definition at line 4 of file **log.c**.

Referenced by **init\_error()**, and **log\_info()**.

**7.81.2.2 log\_level**

```
unsigned int log_level = INFO
```

Definition at line 3 of file **log.c**.

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

**7.82 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 }

```

## 7.83 src/matrix.c File Reference

```

#include "matrix.h"
#include "log.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

```

### 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 diagonal 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.*

### 7.83.1 Function Documentation

#### 7.83.1.1 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 **15** of file **matrix.c**.

References **error()**.

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

#### 7.83.1.2 copyMatrix()

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

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

#### Parameters

|          |                         |
|----------|-------------------------|
| <i>m</i> | a pointer to the matrix |
|----------|-------------------------|

**Returns**

a copied matrix

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

References **scaleMatrix()**.

**7.83.1.3 covarianceMatrix()**

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

returns the covariance of the matrix

**Parameters**

|            |        |
|------------|--------|
| <i>the</i> | matrix |
|------------|--------|

**Returns**

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

Definition at line 169 of file **matrix.c**.

References **assert()**, **\_matrix::data**, **freeMatrix()**, **\_matrix::height**, **makeMatrix()**, **meanMatrix()**, and **\_matrix↵::width**.

**7.83.1.4 dotDiagonalMatrix()**

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

performs a diagonal 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**

|          |                   |
|----------|-------------------|
| <i>a</i> | the first matrix  |
| <i>b</i> | the second matrix |

**Returns**

the matrix result

Definition at line **389** of file **matrix.c**.

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

**7.83.1.5 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**

|            |                  |
|------------|------------------|
| <i>a</i>   | the first matrix |
| <i>the</i> | second matrix    |

**Returns**

the result of the dot product

Definition at line **336** of file **matrix.c**.

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

**7.83.1.6 freeMatrix()**

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

Frees the resources of a matrix.

**Parameters**

|            |                |
|------------|----------------|
| <i>the</i> | matrix to free |
|------------|----------------|



Definition at line 59 of file **matrix.c**.

References **\_matrix::data**.

Referenced by **covarianceMatrix()**.

#### 7.83.1.7 identityMatrix()

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

Returns an identity matrix of size *n* by *n*.

##### Parameters

|          |                   |
|----------|-------------------|
| <i>n</i> | the input matrix. |
|----------|-------------------|

##### Returns

the identity matrix parameter.

Definition at line 93 of file **matrix.c**.

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

#### 7.83.1.8 makeMatrix()

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

Makes a matrix with a width and height parameters.

##### Parameters

|               |                          |
|---------------|--------------------------|
| <i>width</i>  | The width of the matrix  |
| <i>height</i> | the height of the matrix |

##### Returns

the new matrix

Definition at line 28 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()**.

#### 7.83.1.9 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()**.

#### 7.83.1.10 multiplyMatrix()

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

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

##### Parameters

|          |   |
|----------|---|
| <i>a</i> | the first matrix  |
| <i>b</i> | the seconf matrix return the result of the multiplication |

Definition at line **231** of file **matrix.c**.

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

#### 7.83.1.11 printMatrix()

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

Prints a matrix.

## Parameters

|            |        |
|------------|--------|
| <i>the</i> | matrix |
|------------|--------|

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

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

## 7.83.1.12 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

|            |   |
|------------|---|
| <i>the</i> | matrix to swap. This method changes the input matrix. |
| <i>the</i> | first row   |
| <i>the</i> | second row  |

Definition at line 292 of file **matrix.c**.

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

## 7.83.1.13 scaleMatrix()

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

scales a matrix.

## Parameters

|            |                     |
|------------|---------------------|
| <i>m</i>   | the matrix to scale |
| <i>the</i> | value to scale by   |

## Returns

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

Definition at line 270 of file **matrix.c**.

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

Referenced by **copyMatrix()**.

#### 7.83.1.14 **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 115 of file **matrix.c**.

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

#### 7.83.1.15 **transposeMatrix()**

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

returns the transpose matrix.

##### Parameters

|            |                      |
|------------|----------------------|
| <i>the</i> | matrix to transpose. |
|------------|----------------------|

##### Returns

the transposed matrix.

Definition at line 207 of file **matrix.c**.

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

## 7.84 matrix.c

```

00001 #include "matrix.h"
00002 #include "log.h"
00003 #include <stdio.h>
00004 #include <stdlib.h>
00005 #include <string.h>
00006
00015 void assert(int assertion, const char *message) {
00016     if (assertion == 0) {
00017         error(message);
00018         exit(1);
00019     }
00020 }
00021
00028 matrix *makeMatrix(int width, int height) {
00029     matrix *out;
00030     assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
00031     out = (matrix *)malloc(sizeof(matrix));
00032
00033     assert(out != NULL, "Out of memory.");
00034
00035     out->width = width;
00036     out->height = height;
00037     out->data = (double *)malloc(sizeof(double) * width * height);
00038
00039     assert(out->data != NULL, "Out of memory.");
00040
00041     memset(out->data, 0.0, width * height * sizeof(double));
00042
00043     return out;
00044 }
00045
00053 matrix *copyMatrix(matrix *m) { return scaleMatrix(m, 1); }
00054
00059 void freeMatrix(matrix *m) {
00060     if (m != NULL) {
00061         if (m->data != NULL) {
00062             free(m->data);
00063             m->data = NULL;
00064         }
00065         free(m);
00066     }
00067     return;
00068 }
00069
00074 void printMatrix(matrix *m) {
00075     int i, j;
00076     double *ptr = m->data;
00077     printf("%d %d\n", m->width, m->height);
00078     for (i = 0; i < m->height; i++) {
00079         for (j = 0; j < m->width; j++) {
00080             printf(" %9.6f", *(ptr++));
00081         }
00082         printf("\n");
00083     }
00084     return;
00085 }
00086
00093 matrix *identityMatrix(int n) {
00094     int i;
00095     matrix *out;
00096     double *ptr;
00097
00098     assert(n > 0, "Identity matrix must have value greater than zero.");
00099
00100     out = makeMatrix(n, n);
00101     ptr = out->data;
00102     for (i = 0; i < n; i++) {
00103         *ptr = 1.0;
00104         ptr += n + 1;
00105     }
00106
00107     return out;
00108 }
00109
00115 double traceMatrix(matrix *m) {
00116     int i;
00117     int size;
00118     double *ptr = m->data;

```

```

00119     double sum = 0.0;
00120
00121     if (m->height < m->width) {
00122         size = m->height;
00123     } else {
00124         size = m->width;
00125     }
00126
00127     for (i = 0; i < size; i++) {
00128         sum += *ptr;
00129         ptr += m->width + 1;
00130     }
00131
00132     return sum;
00133 }
00134
00142 matrix *meanMatrix(matrix *m) {
00143     int i, j;
00144     matrix *out;
00145
00146     assert(m->height > 0, "Height of matrix cannot be zero.");
00147
00148     out = makeMatrix(m->width, 1);
00149
00150     for (i = 0; i < m->width; i++) {
00151         double *ptr;
00152         out->data[i] = 0.0;
00153         ptr = &m->data[i];
00154         for (j = 0; j < m->height; j++) {
00155             out->data[i] += *ptr;
00156             ptr += out->width;
00157         }
00158         out->data[i] /= (double)m->height;
00159     }
00160     return out;
00161 }
00162
00169 matrix *covarianceMatrix(matrix *m) {
00170     int i, j, k = 0;
00171     matrix *out;
00172     matrix *mean;
00173     double *ptrA;
00174     double *ptrB;
00175     double *ptrOut;
00176
00177     assert(m->height > 1, "Height of matrix cannot be zero or one.");
00178
00179     mean = meanMatrix(m);
00180     out = makeMatrix(m->width, m->width);
00181     ptrOut = out->data;
00182
00183     for (i = 0; i < m->width; i++) {
00184         for (j = 0; j < m->width; j++) {
00185             ptrA = &m->data[i];
00186             ptrB = &m->data[j];
00187             *ptrOut = 0.0;
00188             for (k = 0; k < m->height; k++) {
00189                 *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
00190                 ptrA += m->width;
00191                 ptrB += m->width;
00192             }
00193             *ptrOut /= m->height - 1;
00194             ptrOut++;
00195         }
00196     }
00197
00198     freeMatrix(mean);
00199     return out;
00200 }
00201
00207 matrix *transposeMatrix(matrix *m) {
00208     matrix *out = makeMatrix(m->height, m->width);
00209     double *ptrM = m->data;
00210     int i, j;
00211
00212     for (i = 0; i < m->height; i++) {
00213         double *ptrOut;
00214         ptrOut = &out->data[i];
00215         for (j = 0; j < m->width; j++) {
00216             *ptrOut = *ptrM;
00217             ptrM++;

```

```
00218     ptrOut += out->width;
00219 }
00220 }
00221
00222 return out;
00223 }
00224
00231 matrix *multiplyMatrix(matrix *a, matrix *b) {
00232     int i, j, k;
00233     matrix *out;
00234     double *ptrOut;
00235     double *ptrA;
00236     double *ptrB;
00237
00238     assert(a->width == b->height,
00239          "Matrices have incorrect dimensions. a->width != b->height");
00240
00241     out = makeMatrix(b->width, a->height);
00242     ptrOut = out->data;
00243
00244     for (i = 0; i < a->height; i++) {
00245
00246         for (j = 0; j < b->width; j++) {
00247             ptrA = &a->data[i * a->width];
00248             ptrB = &b->data[j];
00249
00250             *ptrOut = 0;
00251             for (k = 0; k < a->width; k++) {
00252                 *ptrOut += *ptrA * *ptrB;
00253                 ptrA++;
00254                 ptrB += b->width;
00255             }
00256             ptrOut++;
00257         }
00258     }
00259
00260     return out;
00261 }
00262
00270 matrix *scaleMatrix(matrix *m, double value) {
00271     int i, elements = m->width * m->height;
00272     matrix *out = makeMatrix(m->width, m->height);
00273     double *ptrM = m->data;
00274     double *ptrOut = out->data;
00275
00276     for (i = 0; i < elements; i++) {
00277         *(ptrOut++) = *(ptrM++) * value;
00278     }
00279
00280     return out;
00281 }
00282
00292 void rowSwap(matrix *a, int p, int q) {
00293     int i;
00294     double temp;
00295     double *pRow;
00296     double *qRow;
00297
00298     assert(a->height > 2, "Matrix must have at least two rows to swap.");
00299     assert(p < a->height && q < a->height,
00300          "Values p and q must be less than the height of the matrix.");
00301
00302     // If p and q are equal, do nothing.
00303     if (p == q) {
00304         return;
00305     }
00306
00307     pRow = a->data + (p * a->width);
00308     qRow = a->data + (q * a->width);
00309
00310     // Swap!
00311     for (i = 0; i < a->width; i++) {
00312         temp = *pRow;
00313         *pRow = *qRow;
00314         *qRow = temp;
00315         pRow++;
00316         qRow++;
00317     }
00318
00319     return;
00320 }
```

```

00321
00336 matrix *dotProductMatrix(matrix *a, matrix *b) {
00337     matrix *out;
00338     double *ptrOut;
00339     double *ptrA;
00340     double *ptrB;
00341     int i, j, k;
00342
00343     if (b != NULL) {
00344         assert(a->width == b->width,
00345             "Matrices must be of the same dimensionality.");
00346     }
00347
00348     // Are we computing the sum of squares of the same matrix?
00349     if (a == b || b == NULL) {
00350         b = a; // May not appear safe, but we can do this without risk of losing b.
00351     }
00352
00353     out = makeMatrix(b->height, a->height);
00354     ptrOut = out->data;
00355
00356     for (i = 0; i < a->height; i++) {
00357         ptrB = b->data;
00358
00359         for (j = 0; j < b->height; j++) {
00360             ptrA = &a->data[i * a->width];
00361
00362             *ptrOut = 0;
00363             for (k = 0; k < a->width; k++) {
00364                 *ptrOut += *ptrA * *ptrB;
00365                 ptrA++;
00366                 ptrB++;
00367             }
00368             ptrOut++;
00369         }
00370     }
00371
00372     return out;
00373 }
00374
00389 matrix *dotDiagonalMatrix(matrix *a, matrix *b) {
00390     matrix *out;
00391     double *ptrOut;
00392     double *ptrA;
00393     double *ptrB;
00394     int i, j;
00395
00396     if (b != NULL) {
00397         assert(a->width == b->width && a->height == b->height,
00398             "Matrices must be of the same dimensionality.");
00399     }
00400
00401     // Are we computing the sum of squares of the same matrix?
00402     if (a == b || b == NULL) {
00403         b = a; // May not appear safe, but we can do this without risk of losing b.
00404     }
00405
00406     out = makeMatrix(1, a->height);
00407     ptrOut = out->data;
00408     ptrA = a->data;
00409     ptrB = b->data;
00410
00411     for (i = 0; i < a->height; i++) {
00412         *ptrOut = 0;
00413         for (j = 0; j < a->width; j++) {
00414             *ptrOut += *ptrA * *ptrB;
00415             ptrA++;
00416             ptrB++;
00417         }
00418         ptrOut++;
00419     }
00420
00421     return out;
00422 }

```

## 7.85 src/menu.c File Reference

```
#include "menu.h"
```



## 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 context. 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*

### 7.85.1 Function Documentation

#### 7.85.1.1 calculate\_current\_display()

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

Static function that calculates the string from menu.

#### Parameters

|             |  |
|-------------|--|
| <i>rtn</i>  | the string to be written to                    |
| <i>menu</i> | the menu for the display to be calculated from |

#### Author

Chris Jerrett

#### Date

9/8/17

Definition at line 120 of file `menu.c`.

References `menu_t::current`, `FLOAT_TYPE`, `ftoa()`, `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()`.

#### 7.85.1.2 `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 27 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()`.

#### 7.85.1.3 `denint_menu()`

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

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

##### Parameters

|                   |                  |
|-------------------|------------------|
| <code>menu</code> | the menu to free |
|-------------------|------------------|

##### See also

`menu`

## Author

Chris Jerrett

## Date

9/8/17

Definition at line 203 of file **menu.c**.References **menu\_t::options**, and **menu\_t::prompt**.

## 7.85.1.4 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

|             |                     |
|-------------|---------------------|
| <i>menu</i> | the menu to display |
|-------------|---------------------|

## See also

**menu\_type** (p. 100)

## Author

Chris Jerrett

## Date

9/8/17

Definition at line 164 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**.

## 7.85.1.5 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

|             |                  |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

## See also

**menu\_type** (p. 100)

## Parameters

|               |                               |
|---------------|-------------------------------|
| <i>min</i>    | the minimum value             |
| <i>max</i>    | the maximum value             |
| <i>step</i>   | the step value                |
| <i>prompt</i> | the prompt to display to user |

## Author

Chris Jerrett

## Date

9/8/17

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

References **create\_menu()**, **max()**, **menu\_t::max\_f**, **min()**, **menu\_t::min\_f**, and **menu\_t::step\_f**.

7.85.1.6 **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

|             |                  |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

**menu\_type** (p. 100)

#### Parameters

|               |                               |
|---------------|-------------------------------|
| <i>min</i>    | the minimum value             |
| <i>max</i>    | the maximum value             |
| <i>step</i>   | the step value                |
| <i>prompt</i> | the prompt to display to user |

#### Author

Chris Jerrett

#### Date

9/8/17

Definition at line **88** of file **menu.c**.

References **create\_menu()**, **menu\_t::current**, **max()**, **menu\_t::max**, **min()**, **menu\_t::min**, and **menu\_t::step**.

#### 7.85.1.7 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

|             |                  |
|-------------|------------------|
| <i>type</i> | the type of menu |
|-------------|------------------|

See also

**menu\_type** (p. 100)

#### Parameters

|                |   |
|----------------|---|
| <i>nums</i>    | the number of elements passed to function |
| <i>prompt</i>  | the prompt to display to user             |
| <i>options</i> | the options to display for user           |

**Author**

Chris Jerrett

**Date**

9/8/17

Definition at line 60 of file **menu.c**.References **create\_menu()**, **menu\_t::length**, and **menu\_t::options**.Referenced by **initialize()**.**7.86 menu.c**

```

00001 #include "menu.h"
00002
00009 static menu_t *create_menu(enum menu_type type, const char *prompt);
00010
00019 static void calculate_current_display(char *rtn, menu_t *menu);
00020
00027 static menu_t *create_menu(enum menu_type type, const char *prompt) {
00028     menu_t *menu = (menu_t *)malloc(sizeof(menu_t));
00029     if (!menu) {
00030         error("Menu Malloc");
00031     }
00032     menu->type = type;
00033     // Add one for null terminator
00034     size_t strlength = strlen(prompt) + 1;
00035     menu->prompt = (char *)malloc(strlength * sizeof(char));
00036     memcpy(menu->prompt, prompt, strlength);
00037     menu->max = INT_MAX;
00038     menu->min = INT_MIN;
00039     menu->step = 1;
00040     menu->min_f = FLT_MIN;
00041     menu->max_f = FLT_MAX;
00042     menu->step_f = 1;
00043     menu->current = 0;
00044
00045     return menu;
00046 }
00047
00060 menu_t *init_menu_var(enum menu_type type, const char *prompt, int nums, ...) {
00061     menu_t *menu = create_menu(type, prompt);
00062     va_list ap;
00063     char **options_array = (char **)calloc(sizeof(char *), nums);
00064     va_start(ap, nums);
00065     for (int i = 0; i < nums; i++) {
00066         options_array[i] = (char *)va_arg(ap, char *);
00067         printf("%s\n", options_array[i]);
00068     }
00069     va_end(ap);
00070     menu->options = options_array;
00071     menu->length = nums;
00072     return menu;
00073 }
00074
00088 menu_t *init_menu_int(enum menu_type type, int min, int max, int step,
00089                     const char *prompt) {
00090     menu_t *menu = create_menu(type, prompt);
00091     menu->min = min;
00092     menu->max = max;
00093     menu->step = step;
00094     menu->current = 0;
00095     return menu;
00096 }
00097
00111 menu_t *init_menu_float(enum menu_type type, float min, float max, float step,
00112                        const char *prompt) {

```

```

00113 menu_t *menu = create_menu(type, prompt);
00114 menu->min_f = min;
00115 menu->max_f = max;
00116 menu->step_f = step;
00117 return menu;
00118 }
00119
00120 static void calculate_current_display(char *rtn, menu_t *menu) {
00121     if (menu->type == STRING_TYPE) {
00122         int index = menu->current % menu->length;
00123         sprintf(rtn, "%s", menu->options[index]);
00124         printf("%s\n", rtn);
00125         return;
00126     }
00127     if (menu->type == INT_TYPE) {
00128         int step = (menu->step);
00129         int min = (menu->min);
00130         int max = (menu->max);
00131         int value = menu->current * step;
00132         if (value < min) {
00133             value = min;
00134             menu->current++;
00135         }
00136         if (value > max) {
00137             value = max;
00138             menu->current--;
00139         }
00140         sprintf(rtn, "%d", value);
00141     }
00142     if (menu->type == FLOAT_TYPE) {
00143         float step = (menu->step_f);
00144         float min = (menu->min_f);
00145         float max = (menu->max_f);
00146         float value = menu->current * step;
00147         value = value < min ? min : value;
00148         value = value > max ? max : value;
00149         ftoa(value, rtn, 5);
00150     }
00151 }
00152 }
00153
00164 int display_menu(menu_t *menu) {
00165     lcd_print(TOP_ROW, menu->prompt);
00166     printf("printed prompt\n");
00167     // Will exit if teleop or autonomous begin. This is extremely important if
00168     // robot disconnects or resets.
00169     char val[16];
00170     while (lcd_get_pressed_buttons().middle == RELEASED) {
00171         calculate_current_display(val, menu);
00172
00173         if (lcd_get_pressed_buttons().right == PRESSED) {
00174             menu->current += 1;
00175         }
00176         if (lcd_get_pressed_buttons().left == PRESSED) {
00177             menu->current -= 1;
00178         }
00179         printf("%s\n", val);
00180         printf("%d\n", menu->current);
00181         lcd_print(2, val);
00182         delay(300);
00183     }
00184     printf("%d\n", menu->current);
00185     printf("return\n");
00186     lcd_clear();
00187     lcd_print(1, "Thk Cm Agn");
00188     lcd_print(2, val);
00189     delay(800);
00190     lcd_clear();
00191     return menu->current;
00192 }
00193
00203 void denint_menu(menu_t *menu) {
00204     free(menu->prompt);
00205     if (menu->options != NULL)
00206         free(menu->options);
00207     free(menu);
00208 }

```

## 7.87 src/mobile\_goal\_intake.c File Reference

```
#include "mobile_goal_intake.h"
#include "log.h"
#include "partner.h"
```

### Functions

- void **lower\_intake** ()  
*lowers the intake*
- void **raise\_intake** ()  
*raises the intake*
- void **set\_intake\_motor** (int n)  
*sets the intake motor*
- void **update\_intake** ()  
*updates the mobile goal intake in teleop.*

### 7.87.1 Function Documentation

#### 7.87.1.1 lower\_intake()

```
void lower_intake ( )
```

lowers the intake

Definition at line 7 of file **mobile\_goal\_intake.c**.

References **set\_intake\_motor**().

Referenced by **autonomous**(), and **update\_intake**().

#### 7.87.1.2 raise\_intake()

```
void raise_intake ( )
```

raises the intake

Definition at line 9 of file **mobile\_goal\_intake.c**.

References **set\_intake\_motor**().

Referenced by **autonomous**(), and **update\_intake**().



### 7.87.1.3 set\_intake\_motor()

```
void set_intake_motor (
    int n )
```

sets the intake motor

#### Author

Chris Jerrett

Definition at line 5 of file **mobile\_goal\_intake.c**.

References **INTAKE\_MOTOR**, and **set\_motor\_immediate()**.

Referenced by **autonomous()**, **lower\_intake()**, **raise\_intake()**, and **update\_intake()**.

### 7.87.1.4 update\_intake()

```
void update_intake ( )
```

updates the mobile goal intake in teleop.

#### Author

Chris Jerrett

Definition at line 14 of file **mobile\_goal\_intake.c**.

References **lower\_intake()**, **MASTER**, **raise\_intake()**, and **set\_intake\_motor()**.

Referenced by **operatorControl()**.

## 7.88 mobile\_goal\_intake.c

```
00001 #include "mobile_goal_intake.h"
00002 #include "log.h"
00003 #include "partner.h"
00004
00005 void set_intake_motor(int n) { set_motor_immediate(INTAKE_MOTOR, n); }
00006
00007 void lower_intake() { set_intake_motor(-100); }
00008
00009 void raise_intake() { set_intake_motor(100); }
00010
00014 void update_intake() {
00015     if (joystickGetDigital(MASTER, 7, JOY_UP)) {
00016         raise_intake();
00017     } else if (joystickGetDigital(MASTER, 7, JOY_DOWN)) {
00018         lower_intake();
00019     } else
00020         set_intake_motor(0);
00021 }
```

## 7.89 src/opcontrol.c File Reference

File for operator control code.

```
#include "drive.h"
#include "main.h"
#include "slew.h"
#include "claw.h"
#include "lifter.h"
#include "localization.h"
#include "log.h"
#include "mobile_goal_intake.h"
#include "routines.h"
#include "toggle.h"
#include "vmath.h"
```

### Functions

- void **operatorControl** ()

#### 7.89.1 Detailed Description

File for operator control code.

This file should contain the user **operatorControl()** (p. 208) 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**.

#### 7.89.2 Function Documentation

## 7.89.2.1 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 49 of file **opcontrol.c**.

References **buttonInit()**, **init\_routine()**, **init\_slew()**, **update\_claw()**, **update\_drive\_motors()**, **update\_intake()**, and **update\_lifter()**.

## 7.90 opcontrol.c

```
00001
00014 #include "drive.h"
00015 #include "main.h"
00016 #include "slew.h"
00017
00018 #include "claw.h"
00019 #include "lifter.h"
00020 #include "localization.h"
00021 #include "log.h"
00022 #include "mobile_goal_intake.h"
00023 #include "routines.h"
00024 #include "toggle.h"
00025 #include "vmath.h"
00026
00049 void operatorControl() {
00050     buttonInit();
00051     init_routine();
00052     init_slew();
00053     delay(10);
00054     while (1) {
00055         update_claw();
00056         update_intake();
00057         update_lifter();
00058         update_drive_motors();
00059         delay(10);
00060     }
00061 }
```

## 7.91 src/partner.c File Reference

```
#include "partner.h"
```

## 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**

### 7.91.1 Function Documentation

#### 7.91.1.1 **get\_mode**()

```
enum CONTROLL_MODE get_mode ( )
```

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

References **mode**.

Referenced by **update\_drive\_motors**().

#### 7.91.1.2 **update\_control**()

```
void update_control ( )
```

Updates the controller mode between Driver and Partner modes.

## Author

Chris Jerrett

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

References **MAIN\_CONTROLLER\_MODE**, **mode**, **PARTNER**, and **PARTNER\_CONTROLLER\_MODE**.

### 7.91.2 Variable Documentation

## 7.91.2.1 mode

```
enum CONTROLL_MODE mode = MAIN_CONTROLLER_MODE [static]
```

Definition at line **3** of file **partner.c**.

Referenced by **get\_mode()**, and **update\_control()**.

## 7.92 partner.c

```
00001 #include "partner.h"
00002
00003 static enum CONTROLL_MODE mode = MAIN_CONTROLLER_MODE;
00004
00005 enum CONTROLL_MODE get_mode() { return mode; }
00006
00007 void update_control() {
00008     if (joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
00009         mode = MAIN_CONTROLLER_MODE;
00010     } else if (joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
00011         mode = PARTNER_CONTROLLER_MODE;
00012     }
00013 }
```

## 7.93 src/routines.c File Reference

```
#include "routines.h"
#include "controller.h"
#include "list.h"
#include "log.h"
#include "toggle.h"
#include <API.h>
```

## Functions

- void **deinit\_routines** ()
- void **init\_routine** ()
- void **register\_routine** (void(\*routine)(), **button\_t** on\_buttons, **button\_t** \*prohibited\_buttons)
- void **routine\_task** ()

## Variables

- static **list\_t** \* **routine\_list**
- static TaskHandle **routine\_task\_var**

## 7.93.1 Function Documentation

### 7.93.1.1 deinit\_routines()

```
void deinit_routines ( )
```

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

References **list\_destroy()**.

### 7.93.1.2 init\_routine()

```
void init_routine ( )
```

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

References **list\_new()**, **routine\_task()**, and **routine\_task\_var**.

Referenced by **operatorControl()**.

### 7.93.1.3 register\_routine()

```
void register_routine (
    void(*)() routine,
    button_t on_buttons,
    button_t * prohibited_buttons )
```

Definition at line **35** of file **routines.c**.

References **routine\_t::blocked\_buttons**, **list\_node\_new()**, **list\_rpush()**, **routine\_t::on\_button**, **routine\_t::routine**, and **list\_node::val**.

### 7.93.1.4 routine\_task()

```
void routine_task ( )
```

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

References **buttonIsNewPress()**, **LIST\_HEAD**, **list\_iterator\_destroy()**, **list\_iterator\_new()**, **list\_iterator\_next()**, **routine\_t::on\_button**, **routine\_t::routine**, and **list\_node::val**.

Referenced by **init\_routine()**.

## 7.93.2 Variable Documentation

## 7.93.2.1 routine\_list

```
list_t* routine_list [static]
```

Definition at line 8 of file **routines.c**.

## 7.93.2.2 routine\_task\_var

```
TaskHandle routine_task_var [static]
```

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

Referenced by **init\_routine()**.

## 7.94 routines.c

```
00001 #include "routines.h"
00002 #include "controller.h"
00003 #include "list.h"
00004 #include "log.h"
00005 #include "toggle.h"
00006 #include <API.h>
00007
00008 static list_t *routine_list;
00009
00010 static TaskHandle routine_task_var;
00011
00012 void routine_task() {
00013     list_node_t *node;
00014     list_iterator_t *it = list_iterator_new(routine_list, LIST_HEAD);
00015     if (it != NULL) {
00016         while (node = list_iterator_next(it)) {
00017             if (node->val != NULL) {
00018                 routine_t *routine = (routine_t *) (node->val);
00019                 if (buttonIsNewPress(routine->on_button)) {
00020                     routine->routine();
00021                 }
00022             }
00023         }
00024     }
00025     list_iterator_destroy(it);
00026 }
00027
00028 void init_routine() {
00029     routine_list = list_new();
00030     routine_task_var = taskRunLoop(routine_task, 20);
00031 }
00032
00033 void deinit_routines() { list_destroy(routine_list); }
00034
00035 void register_routine(void (*routine)(), button_t on_buttons,
00036                     button_t *prohibited_buttons) {
00037     struct routine_t *r = (struct routine_t *) malloc(sizeof(routine_t));
00038     r->blocked_buttons = prohibited_buttons;
00039     r->routine = routine;
00040     r->on_button = on_buttons;
00041     list_node_t *node = list_node_new(r);
00042     node->val = r;
00043     list_rpush(routine_list, node);
00044 }
```

## 7.95 src/slew.c File Reference

```
#include "slew.h"  
#include "log.h"
```

### 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**

### 7.95.1 Function Documentation

#### 7.95.1.1 deinitslew()

```
void deinitslew ( )
```

Deinitializes the slew rate controller and frees memory.

#### Author

Chris Jerrett

#### Date

9/14/17

Definition at line 59 of file **slew.c**.

References **initialized**, **motors\_curr\_speeds**, **motors\_set\_speeds**, and **slew**.

Referenced by **autonomous()**.



## 7.95.1.2 init\_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

**Author**

Chris Jerrett, Christian DeSimone

**Date**

9/14/17

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

References **initialized**, **motors\_curr\_speeds**, **motors\_set\_speeds**, **slew**, **speeds\_mutex**, **updateMotors()**, and **warning()**.

Referenced by **autonomous()**, **operatorControl()**, **set\_motor\_immediate()**, and **set\_motor\_slew()**.

## 7.95.1.3 set\_motor\_immediate()

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

Sets the motor speed ignoring the slew controller.

**Parameters**

|              |  |
|--------------|--|
| <i>motor</i> | the motor port to use                  |
| <i>speed</i> | the speed to use, between -127 and 127 |

**Author**

Chris Jerrett

**Date**

9/14/17

Definition at line 90 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()**, and **set\_secondary\_lifter\_↵  
motors()**.

#### 7.95.1.4 set\_motor\_slew()

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

Sets motor speed wrapped inside the slew rate controller.

##### Parameters

|              |  |
|--------------|--|
| <i>motor</i> | the motor port to use                  |
| <i>speed</i> | the speed to use, between -127 and 127 |

##### Author

Chris Jerrett

##### Date

9/14/17

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

References **debug()**, **init\_slew()**, **initialized**, **motors\_set\_speeds**, and **speeds\_mutex**.

Referenced by **set\_main\_lifter\_motors()**, and **set\_side\_speed()**.

#### 7.95.1.5 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 19 of file **slew.c**.

References **motors\_curr\_speeds**, **motors\_set\_speeds**, and **speeds\_mutex**.

Referenced by **init\_slew()**.

## 7.95.2 Variable Documentation

### 7.95.2.1 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()**.

### 7.95.2.2 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()**.

### 7.95.2.3 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()**.

### 7.95.2.4 slew

```
TaskHandle slew = NULL [static]
```

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

Referenced by **deinitslew()**, and **init\_slew()**.

### 7.95.2.5 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()**.

## 7.96 slew.c

```

00001 #include "slew.h"
00002 #include "log.h"
00003
00004 static Mutex speeds_mutex;
00005
00006 static int motors_set_speeds[10];
00007 static int motors_curr_speeds[10];
00008
00009 static TaskHandle slew = NULL; // TaskHandle is of type void*
00010
00011 static bool initialized = false;
00012
00013 void updateMotors() {
00014     // Take back half approach
00015     // Not linear but equal to setSpeed(1-(1/2)^x)
00016     for (unsigned int i = 0; i < 9; i++) {
00017         if (motors_set_speeds[i] == motors_curr_speeds[i])
00018             continue;
00019         mutexTake(speeds_mutex, 10);
00020         int set_speed = (motors_set_speeds[i]);
00021         int curr_speed = (motors_curr_speeds[i]);
00022         mutexGive(speeds_mutex);
00023         int diff = set_speed - curr_speed;
00024         int offset = diff;
00025         int n = curr_speed + offset;
00026         motors_curr_speeds[i] = n;
00027         motorSet(i + 1, n);
00028     }
00029 }
00030
00031 void init_slew() {
00032     if (initialized) {
00033         warning("Trying to init already init slew");
00034     }
00035     memset(motors_set_speeds, 0, sizeof(int) * 10);
00036     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00037     motorStopAll();
00038     speeds_mutex = mutexCreate();
00039     slew = taskRunLoop(updateMotors, 100);
00040     initialized = true;
00041 }
00042
00043 void deinitSlew() {
00044     taskDelete(slew);
00045     memset(motors_set_speeds, 0, sizeof(int) * 10);
00046     memset(motors_curr_speeds, 0, sizeof(int) * 10);
00047     initialized = false;
00048 }
00049
00050 void set_motor_slew(int motor, int speed) {
00051     if (!initialized) {
00052         debug("Slew Not Initialized! Initializing");
00053         init_slew();
00054     }
00055     mutexTake(speeds_mutex, 10);
00056     motors_set_speeds[motor - 1] = speed;
00057     mutexGive(speeds_mutex);
00058 }
00059
00060 void set_motor_immediate(int motor, int speed) {
00061     if (!initialized) {
00062         debug("Slew Not Initialized! Initializing");
00063         init_slew();
00064     }
00065     motorSet(motor, speed);
00066     mutexTake(speeds_mutex, 10);
00067     motors_curr_speeds[motor - 1] = speed;
00068     motors_set_speeds[motor - 1] = speed;
00069     mutexGive(speeds_mutex);
00070 }

```

## 7.97 src/toggle.c File Reference

```
#include "toggle.h"
```

## 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]

## 7.97.1 Function Documentation

## 7.97.1.1 buttonGetState()

```
bool buttonGetState (
    button_t )
```

Returns the current status of a button (pressed or not pressed)

## Parameters

|               |  |
|---------------|--|
| <i>button</i> | The button to detect from the Buttons enumeration. |
|---------------|--|

## Returns

true (pressed) or false (not pressed)

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

References **LCD\_CENT**, **LCD\_LEFT**, and **LCD\_RIGHT**.

Referenced by **buttonIsNewPress()**.

## 7.97.1.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons array.

Initializes the buttons.

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

References **buttonPressed**.

Referenced by **operatorControl()**.

### 7.97.1.3 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

|               |  |
|---------------|--|
| <i>button</i> | 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 **136** of file **toggle.c**.

References **buttonGetState()**, and **buttonPressed**.

Referenced by **routine\_task()**.

## 7.97.2 Variable Documentation

### 7.97.2.1 buttonPressed

```
bool buttonPressed[27]
```

Represents the array of "wasPressed" for all 27 available buttons.

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

Referenced by **buttonInit()**, and **buttonIsNewPress()**.

## 7.98 toggle.c

```
00001
00012 #include "toggle.h"
00013
00017 bool buttonPressed[27];
00018
00022 void buttonInit() {
00023     for (int i = 0; i < 27; i++)
00024         buttonPressed[i] = false;
00025 }
00026
00027 bool buttonGetState(button_t button) {
00028     bool currentButton = false;
00029
00030     // Determine how to get the current button value (from what function) and
00031     // where it is, then get it.
00032     if (button < LCD_LEFT) {
00033         // button is a joystick button
00034         unsigned char joystick;
00035         unsigned char buttonGroup;
00036         unsigned char buttonLocation;
00037
00038         button_t newButton;
00039         if (button <= 11) {
00040             // button is on joystick 1
00041             joystick = 1;
00042             newButton = button;
00043         } else {
00044             // button is on joystick 2
00045             joystick = 2;
00046             // shift button down to joystick 1 buttons in order to
00047             // detect which button on joystick is queried
00048             newButton = (button_t)(button - 12);
00049         }
00050
00051         switch (newButton) {
00052             case 0:
00053                 buttonGroup = 5;
00054                 buttonLocation = JOY_DOWN;
00055                 break;
00056             case 1:
00057                 buttonGroup = 5;
00058                 buttonLocation = JOY_UP;
00059                 break;
00060             case 2:
00061                 buttonGroup = 6;
00062                 buttonLocation = JOY_DOWN;
00063                 break;
00064             case 3:
00065                 buttonGroup = 6;
00066                 buttonLocation = JOY_UP;
00067                 break;
00068             case 4:
00069                 buttonGroup = 7;
00070                 buttonLocation = JOY_UP;
00071                 break;
00072             case 5:
00073                 buttonGroup = 7;
00074                 buttonLocation = JOY_LEFT;
00075                 break;
00076             case 6:
00077                 buttonGroup = 7;
00078                 buttonLocation = JOY_RIGHT;
00079                 break;
00080             case 7:
00081                 buttonGroup = 7;
00082                 buttonLocation = JOY_DOWN;
00083                 break;
00084             case 8:
00085                 buttonGroup = 8;
00086                 buttonLocation = JOY_UP;
00087                 break;
00088             case 9:
00089                 buttonGroup = 8;
00090                 buttonLocation = JOY_LEFT;
00091                 break;
00092             case 10:
00093                 buttonGroup = 8;
00094                 buttonLocation = JOY_RIGHT;
```

```

00095         break;
00096     case 11:
00097         buttonGroup = 8;
00098         buttonLocation = JOY_DOWN;
00099         break;
00100     default:
00101         break;
00102     }
00103     currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
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 }
00117
00136 bool buttonIsNewPress(button_t button) {
00137     bool currentButton = buttonGetState(button);
00138
00139     if (!currentButton) // buttons is not currently pressed
00140         buttonPressed[button] = false;
00141
00142     if (currentButton && !buttonPressed[button]) {
00143         // button is currently pressed and was not detected as being pressed during
00144         // last check
00145         buttonPressed[button] = true;
00146         return true;
00147     } else
00148         return false; // button is not pressed or was already detected
00149 }

```

## 7.99 src/vlib.c File Reference

```
#include "vlib.h"
```

### 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'*

#### 7.99.1 Function Documentation

##### 7.99.1.1 ftoaa()

```

void ftoaa (
    float a,
    char * buffer,
    int precision )

```

converts a float to string.



## Parameters

|                  |  |
|------------------|--|
| <i>a</i>         | the float                                |
| <i>buffer</i>    | the string the float will be written to. |
| <i>precision</i> | 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()**.

## 7.99.1.2 itoaa()

```
int itoaa (  
    int a,  
    char * buffer,  
    int digits )
```

converts a int to string.

## Parameters

|               |  |
|---------------|--|
| <i>a</i>      | the integer                            |
| <i>buffer</i> | the string the int will be written to. |
| <i>digits</i> | 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()**.**7.99.1.3 reverse()**

```
void reverse (
    char * str,
    int len )
```

reverses a string 'str' of length 'len'

**Author**

Chris Jerrett

**Date**

9/9/2017

**Parameters**

|            |                       |
|------------|-----------------------|
| <i>str</i> | the string to reverse |
| <i>len</i> | the length            |

Definition at line **10** of file **vlib.c**.Referenced by **itoaa()**.**7.100 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++;
00017         j--;
00018     }
00019 }
00020
```

```

00030 int itoa(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 ftoa(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 = itoa(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         itoa((int)fpart, buffer + i + 1, precision);
00076     }
00077 }

```

## 7.101 src/vmath.c File Reference

```
#include "vmath.h"
```

### Functions

- struct **polar\_cord cartesian\_cord\_to\_polar** (struct **cord** cords)  
*Function to convert x and y 2 dimensional cartesian coordinated 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*

#### 7.101.1 Function Documentation

### 7.101.1.1 cartesian\_cord\_to\_polar()

```
struct  polar_cord cartesian_cord_to_polar (
    struct  cord cords )
```

Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.

#### Author

Christian Desimone

#### Date

9/8/2017

#### Parameters

|              |                     |
|--------------|---------------------|
| <i>cords</i> | the cartesian cords |
|--------------|---------------------|

#### Returns

a struct containing the angle and magnitude.

#### See also

**polar\_cord** (p. 20)

**cord** (p. 7)

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

References **cartesian\_to\_polar()**.

### 7.101.1.2 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

|          |  |
|----------|--|
| <i>x</i> | float value of the x cartesian coordinate. |
| <i>y</i> | float value of the y cartesian coordinate. |

#### Returns

a struct containing the angle and magnitude.

#### See also

**polar\_cord** (p. 20)

Definition at line **15** of file **vmath.c**.

References **polar\_cord::angle**, and **polar\_cord::magnitue**.

Referenced by **cartesian\_cord\_to\_polar()**.

#### 7.101.1.3 max()

```
int max (  
    int a,  
    int b )
```

the min of two values

#### Parameters

|          |            |
|----------|------------|
| <i>a</i> | the first  |
| <i>b</i> | the second |

#### Returns

the smaller of a and b

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

Referenced by **calculate\_current\_display()**, **init\_menu\_float()**, and **init\_menu\_int()**.

#### 7.101.1.4 min()

```
int min (  
    int a,  
    int b )
```

the min of two values

**Parameters**

|          |            |
|----------|------------|
| <i>a</i> | the first  |
| <i>b</i> | the second |

**Returns**

the smaller of a and b

Definition at line 71 of file **vmath.c**.

Referenced by **calculate\_current\_display()**, **init\_menu\_float()**, and **init\_menu\_int()**.

**7.101.1.5 sind()**

```
double sind (
    double angle )
```

sine of a angle in degrees

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

References **M\_PI**.

**7.102 vmath.c**

```
00001 #include "vmath.h"
00002
00015 struct polar_cord cartesian_to_polar(float x, float y) {
00016     float degree = 0;
00017     double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
00018
00019     if (x < 0) {
00020         degree += 180.0;
00021     } else if (x > 0 && y < 0) {
00022         degree += 360.0;
00023     }
00024
00025     if (x != 0 && y != 0) {
00026         degree += atan((float)y / (float)x);
00027     } else if (x == 0 && y > 0) {
00028         degree = 90.0;
00029     } else if (y == 0 && x < 0) {
00030         degree = 180.0;
00031     } else if (x == 0 && y < 0) {
00032         degree = 270.0;
00033     }
00034
00035     struct polar_cord p;
00036     p.angle = degree;
00037     p.magnitude = magnitude;
00038     return p;
00039 }
00040
00053 struct polar_cord cartesian_cord_to_polar(struct cord cords) {
00054     return cartesian_to_polar(cords.x, cords.y);
00055 }
00056
00060 double sind(double angle) {
```

```

00061 double angleradians = angle * M_PI / 180.0f;
00062 return sin(angleradians);
00063 }
00064
00071 int min(int a, int b) {
00072     if (a < b)
00073         return a;
00074     return b;
00075 }
00076
00083 int max(int a, int b) {
00084     if (a > b)
00085         return a;
00086     return b;
00087 }

```

## 7.103 test\_code/testMath.py File Reference

### Namespaces

- **testMath**

### Functions

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

## 7.104 testMath.py

```

00001 from math import *
00002
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)

```

## 7.105 testMath.py File Reference

### Namespaces

- **testMath**

## Functions

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

### 7.106 testMath.py

```
00001 from math import *
00002
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)
```