Vex Team A

1.5.8

Generated by Doxygen 1.8.13

Mon Jan 15 2018 11:02:31

Contents

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:

4 File Index

4.1 File List

Here is a list of all files with brief descriptions:

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 of 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_V←OLTAGE**.

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

```

#ifndef _BATTERY_H_
#define _BATTERY_H_

#include <API.h>

#define MIN_MAIN_VOLTAGE 7.8

#define MIN_BACKUP_VOLTAGE 7.8

double main_battery_voltage();

double backup_battery_voltage();

bool battery_level_acceptable();

#endif
```

7.5 include/claw.h File Reference

Code for controlling the claw that grabs the cones.

```
#include "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

```c

#ifndef _CLAW_H_
#define _CLAW_H_

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

#define MAX_CLAW_SPEED 127

#define MIN_CLAW_SPEED -127

#define CLAW_CLOSE MASTER, 5, JOY_UP

#define CLAW_OPEN MASTER, 5, JOY_DOWN

#define CLAW_CLOSE_VAL 3000

#define CLAW_OPEN_VAL 1500

void update_claw();

void set_claw_motor(const int v);

unsigned int getClawTicks();

void open_claw();

void close_claw();

enum claw_state { CLAW_OPEN_STATE, CLAW_CLOSE_STATE, CLAW_NEUTRAL_STATE };

#endif
```

7.7 include/controller.h File Reference

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

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

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 usig the vex controllers.

Author

Chris Jerrett, Christian Desimone

Date

9/9/2017

Definition in file **controller.h**.

7.7.2 Macro Definition Documentation

7.7.2.1 LEFT_BUMPERS

```
#define LEFT_BUMPERS 6
```

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

7.7.2.2 LEFT_BUTTONS

```
#define LEFT_BUTTONS 7
```

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

7.7.2.3 LEFT_JOY_X

```
#define LEFT_JOY_X 4
```

the left x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line **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,
  JOY2_6D = 14,
  JOY2_6U = 15,
  JOY2_7U = 16,
  JOY2_7L = 17,
  JOY2_7R = 18,
  JOY2_7D = 19,
  JOY2_8U = 20,
  JOY2_8L = 21,
  JOY2_8R = 22,
  JOY2_8D = 23,

  LCD_LEFT = 24,
  LCD_CENT = 25,
  LCD_RIGHT = 26
} button_t;

#define MASTER 1

#define PARTNER 2

#define RIGHT_JOY_X 1

#define RIGHT_JOY_Y 2

#define LEFT_JOY_X 4

#define LEFT_JOY_Y 3

enum joystick {
  RIGHT_JOY,
  LEFT_JOY,
};

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

#endif
```

7.9 include/drive.h File Reference

Drive base definitions and enumerations.

```
#include "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 threshhold on the drive.
- void **update_drive_motors** ()

  Updates the drive motors during teleop.

7.9.1 Detailed Description

Drive base definitions and enumerations.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **drive.h**.

7.9.2 Macro Definition Documentation

7.9.2.1 THRESHOLD

```
#define THRESHOLD 10
```

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

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

Referenced by **joystickExp()**.

7.9.3 Typedef Documentation

7.9.3.1 side_t

```
typedef enum side side_t
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

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

7.9.4 Enumeration Type Documentation

7.9.4.1 side

```
enum side
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

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

Enumerator

LEFT	
BOTH	
RIGHT	

Definition at line **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

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

Definition at line **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 threshhold 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

```
#ifndef _DRIVE_H_
#define _DRIVE_H_

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

#define THRESHOLD 10

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

void set_side_speed(side_t side, int speed);

void setThresh(int t);

void update_drive_motors();

#endif
```

7.11 include/encoders.h File Reference

wrapper around encoder functions

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

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_odemetry()**.

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

```
#ifndef _ENCODERS_H_
#define _ENCODERS_H_

#include <API.h>

#define IME_NUMBER 2

bool init_encoders();

int get_encoder_ticks(unsigned char address);

int get_encoder_velocity(unsigned char address);

#endif
```

7.13 include/gyro.h File Reference

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

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

```
#ifndef _GYRO_H_
#define _GYRO_H_

#include "API.h"

#define GYRO_PORT 1
#define GYRO_MULTIPLIER 0

bool init_main_gyro();
float get_main_gyro_angluar_velocity();

#endif
```

7.15 include/lcd.h File Reference

LCD wrapper functions and macros.

`#include <API.h>`

Data Structures

- struct **lcd_buttons**

  represents the state of the lcd buttons

Macros

- #define **BOTTOM_ROW** 2

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

  The top row on the lcd screen.

Enumerations

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

  Represents the state of a button.

Functions

- void **init_main_lcd** (FILE ∗lcd)

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

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

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

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

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

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

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

7.15.1 Detailed Description

LCD wrapper functions and macros.

Author

Chris Jerrett

Date

9/9/2017

Definition in file **lcd.h**.

7.15.2 Macro Definition Documentation

7.15.2.1 BOTTOM_ROW

```
#define BOTTOM_ROW 2
```

The bottom row on the lcd screen.

Author

Chris Jerrett

Date

9/9/2017

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

Referenced by **log_info()**.

7.15.2.2 TOP_ROW

```
#define TOP_ROW 1
```

The top row on the lcd screen.

Author

Chris Jerrett

Date

9/9/2017

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

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

7.15.3 Enumeration Type Documentation

7.15.3.1 button_state

```
enum button_state
```

Represents the state of a button.

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

Author

Chris Jerrett

Date

9/9/2017

Enumerator

RELEASED	A released button
PRESSED	A pressed button

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

7.15.4 Function Documentation

7.15.4.1 init_main_lcd()

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

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

Parameters

lcd	the urart port of the lcd screen

See also

uart1
uart2

Author

Chris Jerrett

Date

9/9/2017

Definition at line **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 **promt_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

line	the line to print on
str	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 **promt_confirmation()**.

7.15.4.5 lcd_printf()

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

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

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

Definition at line **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

state	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 promt_confirmation()

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

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

Parameters

confirm_text	the text for the user to confirm.

Author

Chris Jerrett

Date

9/9/2017

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

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

7.16 lcd.h

```

#ifndef _LCD_H_
#define _LCD_H_

#include <API.h>

#define TOP_ROW 1

#define BOTTOM_ROW 2

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

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

lcd_buttons lcd_get_pressed_buttons();

void lcd_clear();

void init_main_lcd(FILE *lcd);

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

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

void lcd_set_backlight(bool state);

void promt_confirmation(const char *confirm_text);

#endif
```

7.17 include/lifter.h File Reference

Declarations and macros for controlling and manipulating the lifter.

```
#include "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 signficant speed for the lifter.

Functions

- double **getLifterHeight** ()
  Gets the height of the lifter in inches.
- int **getLifterTicks** ()
  Gets the value of the lifter pot.
- float **lifterPotentiometerToDegree** (int x)

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

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

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

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

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

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

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

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

  Updates the lifter in teleop.

7.17.1 Detailed Description

Declarations and macros for controlling and manipulating the lifter.

Author

Chris Jerrett, Christian Desimone

Date

8/27/2017

Definition in file **lifter.h**.

7.17.2 Macro Definition Documentation

7.17.2.1 HEIGHT

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

The integral constant for the lifter PID.

Definition at line **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 signficant 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

pos	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

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()**.

7.17.3.10 set_secondary_lifter_motors()

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

Sets the secondary lifter motors to the given value.

Parameters

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

Author

Chris Jerrett

Date

1/6/2018

Definition at line **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

```

#ifndef _LIFTER_H_
#define _LIFTER_H_

#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>

#define INIT_ROTATION 680

#define SECONDARY_LIFTER_P .05

#define SECONDARY_LIFTER_D 0

#define SECONDARY_LIFTER_I 0.000

#define MAIN_LIFTER_P 0

#define MAIN_LIFTER_D 0

#define MAIN_LIFTER_I 0.0000001

#define THRESHOLD 10

#define HEIGHT 19.1 - 3.8

#define LIFTER_UP MASTER, 6, JOY_UP

#define LIFTER_DOWN MASTER, 6, JOY_DOWN

#define SECONDARY_LIFTER_UP MASTER, 8, JOY_UP

#define SECONDARY_LIFTER_DOWN MASTER, 8, JOY_DOWN

#define LIFTER_DRIVER_LOAD MASTER, RIGHT_BUTTONS, JOY_RIGHT

#define LIFTER_UP_PARTNER PARTNER, 5, JOY_UP

#define LIFTER_DOWN_PARTNER PARTNER, 5, JOY_DOWN

#define SECONDARY_LIFTER_POT_PORT 2

#define SECONDARY_LIFTER_MAX_HEIGHT 3120

#define SECONDARY_LIFTER_MIN_HEIGHT 2000

#define MAIN_LIFTER_POT 1

#define MAIN_LIFTER_MIN_HEIGHT 1700

void set_secondary_lifter_motors(const int v);

void set_main_lifter_motors(const int v);

void set_lifter_pos(int pos);

void raise_main_lifter();

void lower_main_lifter();

void raise_secondary_lifter();

void lower_secondary_lifter();

void update_lifter();

float lifterPotentiometerToDegree(int x);

int getLifterTicks();

double getLifterHeight();

#endif
```

7.19 include/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

```

//
// list.h
//
// Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
//

#ifndef LIST_H
#define LIST_H

#ifdef __cplusplus
extern "C" {
#endif

#include <stdlib.h>

// Memory management macros

#ifndef LIST_MALLOC
#define LIST_MALLOC malloc
#endif

#ifndef LIST_FREE
#define LIST_FREE free
#endif

/*
 * list_t iterator direction.
 */

typedef enum {
    LIST_HEAD
  , LIST_TAIL
} list_direction_t;

/*
 * list_t node struct.
 */

typedef struct list_node {
  struct list_node *prev;
  struct list_node *next;
  void *val;
} list_node_t;

/*
 * list_t struct.
 */

typedef struct {
  list_node_t *head;
  list_node_t *tail;
  unsigned int len;
  void (*free)(void *val);
  int (*match)(void *a, void *b);
} list_t;

/*
 * list_t iterator struct.
 */

typedef struct {
  list_node_t *next;
  list_direction_t direction;
} list_iterator_t;

// Node prototypes.

list_node_t *
list_node_new(void *val);

// list_t prototypes.

list_t *
list_new();

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

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

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

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

list_node_t *
list_rpop(list_t *self);

list_node_t *
list_lpop(list_t *self);

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

void
list_destroy(list_t *self);

// list_t iterator prototypes.

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);

void
list_iterator_destroy(list_iterator_t *self);

#ifdef __cplusplus
}
#endif

#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 posituion of the robot.
- bool **init_localization** (const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int start_theta)
  Starts the localization process.
- void **update_position** ()
  Updates the position from the localization.

7.21.1 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_anglular_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 posituion of the robot.

Parameters

gyro1	The first gyro

Returns

the loacation 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

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

Definition at line **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_odemetry()**, and **last_call**.

Referenced by **init_localization()**.

7.22 localization.h

```c

#ifndef _LOCALIZATION_H_
#define _LOCALIZATION_H_

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

#define LOCALIZATION_UPDATE_FREQUENCY 0.500

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

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

struct location get_position();

void update_position();

int calculate_encoder_angle();

#endif
```

7.23 include/log.h File Reference

Contains logging functions.

```c
#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

debug_message	the message

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

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

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

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

error_message	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

info_message	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

use_lcd	whether to use the lcd
lcd	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

warning_message	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

```
#ifndef _LOG_H_
#define _LOG_H_

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

#define NONE 0

#define ERROR 1

#define WARNING 2

#define INFO 3

#define DEBUG 4

void init_error(bool use_lcd, FILE *lcd);

void error(const char *error_message);

void warning(const char *warning_message);

void info(const char *info_message);

void debug(const char *debug_message);

#endif
```

7.25 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.

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()**, **deinitslew()**, **DEPLOY_HEIGHT**, **HALF_ROTA↩ TE**, **init_slew()**, **LEFT**, **lower_intake()**, **LOWEST_HEIGHT**, **MAIN_LIFTER_POT**, **MAX_HEIGHT**, **MAX_SPE↩ ED**, **MID_LEFT_DRIVE**, **MID_RIGHT_DRIVE**, **MIN_SPEED**, **MOBILE_GOAL_DISTANCE**, **MOBILE_GOAL_HEI↩ GHT**, **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

```
#ifndef MAIN_H_

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

#include <API.h>

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

//#define AUTO_DEBUG

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

// Prototypes for initialization, operator control and autonomous

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

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

#endif
```

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

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

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

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

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

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

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

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

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

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

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

  returns the transpose matrix.

7.27.1 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 prevelant, 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

m	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

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

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

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

Parameters

a	the first matrix
b	the second matrix

Returns

the matrix result

Definition at line **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

a	the first matrix
the	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

the	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

n	the input matrix. parameter.
n	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

width	The width of the matrix
height	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

a	the first matrix
b	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

the	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

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

m	the matrix to scale
the	value to scale by

Returns

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

Definition at line **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

the	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

```
#ifndef _MATRIX_H_
#define _MATRIX_H_

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

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

matrix *makeMatrix(int width, int height);

matrix *copyMatrix(matrix *m);

void freeMatrix(matrix *m);

void printMatrix(matrix *m);

matrix *identityMatrix(int n);

double traceMatrix(matrix *m);

matrix *transposeMatrix(matrix *m);

matrix *meanMatrix(matrix *m);

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

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

matrix *covarianceMatrix(matrix *m);

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

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

#endif
```

7.29 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

menu	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

menu	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

type	the type of menu

See also

menu_type (p. 100)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **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

type	the type of menu

See also

menu_type (p. 100)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **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

type	the type of menu

See also

menu_type (p. 100)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

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

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

Referenced by **initialize()**.

7.30 menu.h

```c

#ifndef _MENU_H_
#define _MENU_H_

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

enum menu_type {
  INT_TYPE,
  FLOAT_TYPE,
  STRING_TYPE
};

typedef struct menu_t {
  enum menu_type type;

  char **options;

  unsigned int length;

  int min;

  int max;

  int step;

  float min_f;

  float max_f;

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

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

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

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

int display_menu(menu_t *menu);

void denint_menu(menu_t *menu);

#endif
```

7.31 include/mobile_goal_intake.h File Reference

```c
#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

```
#ifndef _MOBLE_GOAL_INTAKE_
#define _MOBLE_GOAL_INTAKE_

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

void update_intake();

void set_intake_motor(int n);

void raise_intake();

void lower_intake();


#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

```
#ifndef _PARTNER_H_
#define _PARTNER_H_

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

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

enum CONTROLL_MODE get_mode();

#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

```
#include "controller.h"

typedef struct routine_t {
  button_t on_button;
  button_t* blocked_buttons;
  void(*routine)();
}routine_t;

void init_routine();

void routine_task();

void deinit_routines();

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

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

Author

Chris Jerrett

Date

9/14/17

Definition at line **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

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

Author

Chris Jerrett

Date

9/14/17

Definition at line **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

```

#ifndef _SLEW_H_
#define _SLEW_H_

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

#define UPDATE_PERIOD_MS 25

#define MOTOR_PORTS 12

#define RAMP_PROPORTION 1

void updateMotors();

void deinitslew();

void init_slew();

void set_motor_slew(int motor, int speed);

void set_motor_immediate(int motor, int speed);

#endif
```

7.45 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

button	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

button	The button to detect from the Buttons enumeration (see include/buttons.h).

Returns

true or false depending on if there was a change in button state.

Parameters

button	The button to detect from the Buttons enumeration (see include/buttons.h).

Returns

true or false depending on if there was a change in button state.

Example code:

```
...
if(buttonIsNewPress(JOY1_8D))
    digitalWrite(1, !digitalRead(1));
...
```

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

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

Referenced by **routine_task()**.

7.46 toggle.h

```

#ifndef BUTTONS_H_
#define BUTTONS_H_

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

void buttonInit();

bool buttonIsNewPress(button_t);

bool buttonGetState(button_t);

#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 **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'

7.47.1 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

a	the float
buffer	the string the float will be written to.
precision	digits after the decimal to write

Author

Christian DeSimone

Date

9/26/2017

Definition at line **55** of file **vlib.c**.

References **itoaa()**.

Referenced by **calculate_current_display()**.

7.47.2.3 itoaa()

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

converts a int to string.

Parameters

a	the integer
buffer	the string the int will be written to.
digits	the number of digits to be written

Returns

the digits

Author

Chris Jerrett, Christian DeSimone

Date

9/9/2017

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

References **reverse()**.

Referenced by **ftoaa()**.

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

str	the string to reverse
len	the length

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

Referenced by **itoaa()**.

7.48 vlib.h

```

#ifndef _VLIB_H_
#define _VLIB_H_

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

void reverse(char *str, int len);

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

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

void *calloc_real(size_t elements, size_t size);

#endif
```

7.49 include/vmath.h File Reference

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

```
#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 cordinated to polar coordinates.
- struct **polar_cord** **cartesian_to_polar** (float x, float y)

  Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- int **max** (int a, int b)

  the min of two values
- int **min** (int a, int b)

  the min of two values
- double **sind** (double angle)

  sine of a angle in degrees

7.49.1 Detailed Description

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

Author

Christian Desimone
Chris Jerrett

Date

9/9/2017

Definition in file **vmath.h**.

7.49.2 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_odemetry()**, 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 cordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

cords	the cartesian cords

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 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

x	float value of the x cartesian coordinate.
y	float value of the y cartesian coordinate.

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 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

a	the first
b	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

a	the first
b	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

```

#ifndef _VMATH_H_
#define _VMATH_H_

#include <math.h>
#define M_PI 3.14159265358979323846

struct polar_cord {
  float angle;
  float magnitue;
};

struct cord {
  float x;
  float y;
};

struct polar_cord cartesian_to_polar(float x, float y);

struct polar_cord cartesian_cord_to_polar(struct cord cords);

int min(int a, int b);

int max(int a, int b);

double sind(double angle);
#endif
```

7.51 README.md File Reference

7.52 README.md

```
# InTheZoneA
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()**, **deinitslew()**, **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;
```

```

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

  // Deploy claw
  while (analogRead(SECONDARY_LIFTER_POT_PORT) < DEPLOY_HEIGHT) {

    set_secondary_lifter_motors(MAX_SPEED);
  }
  set_secondary_lifter_motors(0);

  while (analogRead(SECONDARY_LIFTER_POT_PORT) > LOWEST_HEIGHT) {
    set_secondary_lifter_motors(MIN_SPEED);
  }
  set_secondary_lifter_motors(0);

  // Grab pre-load cone
  close_claw();
  delay(300);
  set_claw_motor(0);

  while (analogRead(SECONDARY_LIFTER_POT_PORT) < MAX_HEIGHT) {
    set_secondary_lifter_motors(MAX_SPEED);
  }
  set_secondary_lifter_motors(0);
  // Raise the lifter
  while (analogRead(MAIN_LIFTER_POT) < MOBILE_GOAL_HEIGHT) {
    set_main_lifter_motors(MAX_SPEED);
  }
  set_main_lifter_motors(0);
  // Drive towards the goal

  lower_intake();
  delay(300);
  set_intake_motor(0);

  while (counts_drive < MOBILE_GOAL_DISTANCE) {
    set_side_speed(BOTH, 127);
    // Restablish the distance traveled
    imeGet(MID_LEFT_DRIVE, &counts_drive_left);
    imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
    counts_drive = counts_drive_left + counts_drive_right;
    counts_drive /= 2;
  }
  // Stop moving
  set_side_speed(BOTH, 0);
  delay(1000);

  raise_intake();
  delay(300);
  set_intake_motor(0);

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

  int ang = 0;
  while (ang < HALF_ROTATE) {
    ang += calculate_encoder_angle();
    set_side_speed(LEFT, MAX_SPEED);
    set_side_speed(RIGHT, MIN_SPEED);
  }
  set_side_speed(BOTH, 0);

  counts_drive = 0;

  while (counts_drive < MOBILE_GOAL_DISTANCE + ZONE_DISTANCE) {
    set_side_speed(BOTH, 127);
    // Restablish the distance traveled
    imeGet(MID_LEFT_DRIVE, &counts_drive_left);
    imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
    counts_drive = counts_drive_left + counts_drive_right;
    counts_drive /= 2;
  }

```

```
    lower_intake();
    delay(300);
    set_intake_motor(0);

    set_side_speed(BOTH, MIN_SPEED);
    delay(1000);
    set_side_speed(BOTH, 0);

    deinitslew();
}
```

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_V↩ OLTAGE**.

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_OPE↩ N_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

```
#include "claw.h"
#include "log.h"
#include "toggle.h"
static enum claw_state state = CLAW_NEUTRAL_STATE;

void update_claw() {
  if (joystickGetDigital(CLAW_CLOSE)) {
    state = CLAW_CLOSE_STATE;
  } else if (joystickGetDigital(CLAW_OPEN)) {
    state = CLAW_OPEN_STATE;
  } else {
    state = CLAW_NEUTRAL_STATE;
  }

  if (state == CLAW_CLOSE_STATE) {
    set_claw_motor(MAX_CLAW_SPEED);
  } else if (state == CLAW_OPEN_STATE) {
    set_claw_motor(MIN_CLAW_SPEED);
  } else {
    set_claw_motor(0);
  }
}

void set_claw_motor(const int v) { set_motor_immediate(CLAW_MOTOR, v); }

void open_claw() { set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED); }

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 threshhold on the drive.
- static float **joystickExp** (int joystickVal)
  Applies exponential scale to a joystick value.
- void **set_side_speed** (**side_t side**, int speed)
  sets the speed of one side of the robot.
- void **setThresh** (int t)
  Sets the deadzone threshhold on the drive.
- void **update_drive_motors** ()
  Updates the drive motors during teleop.

Variables

- static int **thresh** = 30

7.61.1 Function Documentation

7.61.1.1 getThresh()

```
int getThresh ( )
```

Gets the deadzone threshhold 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

joystickVal	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

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

Definition at line **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 threshhold 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

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

static int thresh = 30;

int getThresh() { return thresh; }

void setThresh(int t) { thresh = t; }

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

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

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

static float joystickExp(int joystickVal) {
  // make the offset negative if moving backwards
  if (abs(joystickVal) < THRESHOLD) {
    return 0;
  }

  int offset;
  // Use the threshold to ensure the joystick values are significant
  if (joystickVal < 0) {
    offset = -(THRESHOLD);
  } else {
    offset = THRESHOLD;
  }
  // Apply the function ((((x/10)^3)/18) + offset) * 0.8 to the joystick value
  return (pow(joystickVal / 10, 3) / 18 + offset) * 0.8;
}
```

7.63 src/encoders.c File Reference

```c
#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_odemetry()**.

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

```
#include "encoders.h"
#include "log.h"
#include <API.h>

bool init_encoders() {
#ifdef IME_NUMBER
  int count = imeInitializeAll();
  if (count != IME_NUMBER) {
    printf("detected only %d\n", count);
    error("Wrong Number of IMEs Connected");
    return false;
  }
  return true;
#else
  return imeInitializeAll();
#endif
}

int get_encoder_ticks(unsigned char address) {
  int i = 0;
  imeGet(address, &i);
  return i;
}

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

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

```
#include "gyro.h"

static Gyro main_gyro;

bool init_main_gyro() {
  main_gyro = gyroInit(GYRO_PORT, GYRO_MULTIPLIER);
  return main_gyro != NULL;
}

float get_main_gyro_angluar_velocity() {
  uint32_t port = GYRO_PORT;
  int32_t reading = (int32_t)analogReadCalibratedHR(port + 1);
  return 0;
}
```

7.67 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 (usartOpen()) 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

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

extern Ultrasonic lifter_ultrasonic;

/*
 * 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()).
 *
 */
void initializeIO() { watchdogInit(); }

/*
 * Runs user initialization code. This function will be started in its own task
 * with the default priority and stack size once when the robot is starting up.
 * It is possible that the VEXnet communication link may not be fully
 * established at this time, so reading from the VEX Joystick may fail.
 *
 * This function should initialize most sensors (gyro, encoders, ultrasonics),
 * LCDs, global variables, and IMEs.
 *
 * This function must exit relatively promptly, or the operatorControl() and
 * autonomous() tasks will not start. An autonomous mode selection menu like the
 * pre_auton() in other environments can be implemented in this task if desired.
 */
void initialize() {
  init_main_lcd(uart1);
  info("LCD Init");
  if (!battery_level_acceptable())
    error("Bad main/backup bat");
  menu_t *t =
      init_menu_var(STRING_TYPE, "TEST Menu", 5, "1", "2", "3", "4", "5");
  init_error(true, uart2);
  setTeamName("9228A");
  init_encoders();
  lifter_ultrasonic = ultrasonicInit(4, 5);
}
```

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

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

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

Variables

- static FILE ∗ **lcd_port** = NULL

7.69.1 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

lcd	the urart 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 **promt_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 **promt_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

line	the line to print on
str	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 **promt_confirmation()**.

7.69.1.6 lcd_printf()

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

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

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

Definition at line **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

state	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 promt_confirmation()

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

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

Parameters

confirm_text	the text for the user to confirm.

Author

Chris Jerrett

Date

9/9/2017

Definition at line **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 promt_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_interupt** = 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

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**.

7.71.1.5 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**, **M↩
AX_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**, **SE↩CONDARY_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

pos	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_interupt

```
bool lifter_autostack_routine_interupt = 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_interupt = 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) {}
```

```

void raise_main_lifter() { set_main_lifter_motors(MAX_SPEED); }

void lower_main_lifter() { set_main_lifter_motors(MAX_SPEED); }

void raise_secondary_lifter() { set_secondary_lifter_motors(MIN_SPEED / 1.5); }

void lower_secondary_lifter() { set_secondary_lifter_motors(MAX_SPEED); }

static bool secondary_override = false;

static void main_lifter_update() {
  if (lifter_autostack_running)
    return;
  static int count = 0;
  static bool pid_on = false;
  static int main_target = 0;
  int main_motor_speed = 0;
  static long long main_i = 0;
  if (count == 20) {
    main_target = analogRead(MAIN_LIFTER_POT);
  }
  if (pid_on && count > 20) {
    int curr = analogRead(MAIN_LIFTER_POT);
    static int main_last_p = 0;
    int main_p = curr - main_target;
    main_i += main_p;
    int main_d = main_last_p - main_p;
    // main_motor_speed = MAIN_LIFTER_P * main_p + MAIN_LIFTER_I * main_i +
    // MAIN_LIFTER_D * main_d;
    main_last_p = main_p;
  } else {
    main_i = 0;
    count++;
  }

  if (joystickGetDigital(LIFTER_UP)) {
    int ultra = ultrasonicGet(lifter_ultrasonic);
    main_motor_speed = MAX_SPEED;
    count = 0;
  } else if (joystickGetDigital(LIFTER_DOWN)) {
    main_motor_speed = MIN_SPEED;
    count = 0;
    secondary_override = false;
  } else {
    secondary_override = false;
  }
  set_main_lifter_motors(main_motor_speed);
  pid_on = true;
}

static void secondary_lifter_update() {
  if (lifter_autostack_running)
    return;
  static int count = 0;
  // static bool pid_on = false;
  static int second_target = 0;
  int second_motor_speed = 0;
  static long long second_i = 0;

  if (count < 10) {
    second_target = analogRead(SECONDARY_LIFTER_POT_PORT);
    count++;
  }

  int curr = analogRead(SECONDARY_LIFTER_POT_PORT);
  static int second_last_p = 0;
  int second_p = curr - second_target;
  second_i += second_p;
  int second_d = second_last_p - second_p;
  second_motor_speed = SECONDARY_LIFTER_P * second_p +
                       SECONDARY_LIFTER_I * second_i +
                       SECONDARY_LIFTER_D * second_d;
  second_last_p = second_p;

  if (joystickGetDigital(SECONDARY_LIFTER_DOWN)) {
    second_motor_speed = MAX_SPEED;
    count = 0;
    second_i = 0;
    second_target = analogRead(SECONDARY_LIFTER_POT_PORT);
  } else if (joystickGetDigital(SECONDARY_LIFTER_UP)) {
    second_motor_speed = MIN_SPEED;
    count = 0;
    second_i = 0;
    second_target =
        second_target > 3000 ? 4095 : analogRead(SECONDARY_LIFTER_POT_PORT);
    ;
  } else {
    second_target = second_target > 3000 ? 4095 : second_target;
  }
  second_motor_speed = abs(second_motor_speed) < 20 ? 0 : second_motor_speed;
  /*printf("Motor %d \n", second_motor_speed);
  printf("P %d \n", second_p);
  printf("I %lld \n", second_i);
  printf("D %d \n", second_d);*/
  set_secondary_lifter_motors(second_motor_speed);
}

void update_lifter() {
  main_lifter_update();
  if (!secondary_override)
    secondary_lifter_update();
}
float lifterPotentiometerToDegree(int x) {
  return (x - INIT_ROTATION) / TICK_MAX * DEG_MAX;
}

int getLifterTicks() { return analogRead(LIFTER); }

double getLifterHeight() {
  unsigned int ticks = getLifterTicks();
  return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks))) +
          0.0198 * ticks + 2.3033);
}
```

7.73 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;
```

```c
    self->tail->next = node;
    self->tail = node;
  } else {
    self->head = self->tail = node;
    node->prev = node->next = NULL;
  }

  ++self->len;
  return node;
}

/*
 * Return / detach the last node in the list, or NULL.
 */

list_node_t *list_rpop(list_t *self) {
  if (!self->len)
    return NULL;

  list_node_t *node = self->tail;

  if (--self->len) {
    (self->tail = node->prev)->next = NULL;
  } else {
    self->tail = self->head = NULL;
  }

  node->next = node->prev = NULL;
  return node;
}

/*
 * Return / detach the first node in the list, or NULL.
 */

list_node_t *list_lpop(list_t *self) {
  if (!self->len)
    return NULL;

  list_node_t *node = self->head;

  if (--self->len) {
    (self->head = node->next)->prev = NULL;
  } else {
    self->head = self->tail = NULL;
  }

  node->next = node->prev = NULL;
  return node;
}

/*
 * Prepend the given node to the list
 * and return the node, NULL on failure.
 */

list_node_t *list_lpush(list_t *self, list_node_t *node) {
  if (!node)
    return NULL;

  if (self->len) {
    node->next = self->head;
    node->prev = NULL;
    self->head->prev = node;
    self->head = node;
  } else {
    self->head = self->tail = node;
    node->prev = node->next = NULL;
  }

  ++self->len;
  return node;
}

/*
 * Return the node associated to val or NULL.
 */

list_node_t *list_find(list_t *self, void *val) {
  list_iterator_t *it = list_iterator_new(self, LIST_HEAD);
  list_node_t *node;

  while ((node = list_iterator_next(it))) {
    if (self->match) {
      if (self->match(val, node->val)) {
        list_iterator_destroy(it);
        return node;
      }
    } else {
      if (val == node->val) {
        list_iterator_destroy(it);
        return node;
      }
    }
  }

  list_iterator_destroy(it);
  return NULL;
}

/*
 * Return the node at the given index or NULL.
 */

list_node_t *list_at(list_t *self, int index) {
  list_direction_t direction = LIST_HEAD;

  if (index < 0) {
    direction = LIST_TAIL;
    index = ~index;
  }

  if ((unsigned)index < self->len) {
    list_iterator_t *it = list_iterator_new(self, direction);
    list_node_t *node = list_iterator_next(it);
    while (index--)
      node = list_iterator_next(it);
    list_iterator_destroy(it);
    return node;
  }

  return NULL;
}

/*
 * Remove the given node from the list, freeing it and it's value.
 */

void list_remove(list_t *self, list_node_t *node) {
  node->prev ? (node->prev->next = node->next) : (self->head = node->next);

  node->next ? (node->next->prev = node->prev) : (self->tail = node->prev);

  if (self->free)
    self->free(node->val);

  LIST_FREE(node);
  --self->len;
}
```

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

```

//
// iterator.c
//
// Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
//

#include "list.h"

/*
 * Allocate a new list_iterator_t. NULL on failure.
 * Accepts a direction, which may be LIST_HEAD or LIST_TAIL.
 */

list_iterator_t *list_iterator_new(list_t *list, list_direction_t direction) {
  list_node_t *node = direction == LIST_HEAD ? list->head : list->tail;
  return list_iterator_new_from_node(node, direction);
}

/*
 * Allocate a new list_iterator_t with the given start
 * node. NULL on failure.
 */

list_iterator_t *list_iterator_new_from_node(list_node_t *node,
                                             list_direction_t direction) {
  list_iterator_t *self;
  if (!(self = LIST_MALLOC(sizeof(list_iterator_t))))
    return NULL;
  self->next = node;
  self->direction = direction;
  return self;
}

/*
 * Return the next list_node_t or NULL when no more
 * nodes remain in the list.
 */

list_node_t *list_iterator_next(list_iterator_t *self) {
  list_node_t *curr = self->next;
  if (curr) {
    self->next = self->direction == LIST_HEAD ? curr->next : curr->prev;
  }
  return curr;
}

/*
 * Free the list iterator.
 */

void list_iterator_destroy(list_iterator_t *self) {
  LIST_FREE(self);
  self = NULL;
}
```

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

```

//
// node.c
//
// Copyright (c) 2010 TJ Holowaychuk <tj@vision-media.ca>
//

#include "list.h"

/*
 * Allocates a new list_node_t. NULL on failure.
 */

list_node_t *list_node_new(void *val) {
  list_node_t *self;
  if (!(self = LIST_MALLOC(sizeof(list_node_t))))
    return NULL;
  self->prev = NULL;
  self->next = NULL;
  self->val = val;
  return self;
}
```

7.79 src/localization.c File Reference

```
#include "localization.h"
#include "vmath.h"
#include <inttypes.h>
```

Data Structures

- struct **accelerometer_odometry**
- struct **encoder_odemtry**

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_odemetry** ()
- static double **calculate_angle** ()
- int **calculate_encoder_angle** ()
- static void **calculate_encoder_odemetry** ()
- static double **calculate_gryo_anglular_velocity** ()
- struct **location get_position** ()

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

  Starts the localization process.
- static double **integrate_gyro_w** (int new_w)
- void **update_position** ()

  Updates the position from the localization.

Variables

- static Gyro **g1**
- static int **last_call** = 0
- static TaskHandle **localization_task**
- **matrix** ∗ **state_matrix**

7.79.1 Macro Definition Documentation

7.79.1.1 CPR [1/2]

```
#define CPR 392.0
```

Referenced by **calculate_encoder_angle()**, and **calculate_encoder_odemetry()**.

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_odemetry()**.

7.79.1.6 WIDTH [2/2]

```
#define WIDTH 13.5
```

7.79.2 Function Documentation

7.79.2.1 calculate_accelerometer_odemetry()

```
static struct accelerometer_odometry calculate_accelerometer_odemetry ( ) [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 posituion of the robot.

Parameters

gyro1	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

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

Definition at line **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_odemetry()**, 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_anglular_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_odemetry()**, **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

```
#include "localization.h"
#include "vmath.h"
#include <inttypes.h>

static Gyro g1;
static TaskHandle localization_task;

static int last_call = 0;

matrix *state_matrix;

struct encoder_odemtry {
  double x;
  double y;
  double theta;
};

struct accelerometer_odometry {
  double x;
  double y;
};

static double calculate_angle();
static struct accelerometer_odometry calculate_accelerometer_odemetry();

struct location get_position() {}

void update_position() {
  // int curr_theta = calculate_angle();

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

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

static struct accelerometer_odometry calculate_accelerometer_odemetry() {
  static double vel_acumm_x = 0;
  static double vel_acumm_y = 0;

  int32_t accel_x_rel = (int32_t)analogReadCalibratedHR(2);
  int32_t accel_y_rel = (int32_t)analogReadCalibratedHR(3);

  // Ignore atom format string errors
  printf("x: %+" PRId32 " y: %+" PRId32 "\n", accel_x_rel, accel_y_rel);

  double delta_time = ((millis() - last_call) / 1000.0);
  // double accel_x_abs = (accel_x_rel *  cos(theta) + accel_y_rel * sin(theta))
  // * delta_time;  double accel_y_abs = (accel_y_rel *  cos(theta) +
  // accel_x_rel
  // * sin(theta)) * delta_time;

  // vel_acumm_x += accel_x_abs;
  // vel_acumm_y += accel_y_abs;

  // double new_x = x + vel_acumm_x * delta_time;
  // double new_y = y + vel_acumm_y * delta_time;

  struct accelerometer_odometry od;
  // od.x = new_x;
  // od.y = new_y;
  return od;
}

static double integrate_gyro_w(int new_w) {
  static double theta = 0;
  double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;
  theta += delta_theta;
}

static double calculate_gryo_anglular_velocity() {
  static int last_gyro = 0;
  int current = gyroGet(g1);
  // Calculate w (angluar velocity in degrees per second)
  double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY / 1000.0);
  return w;
}

int calculate_encoder_angle() {
#define WIDTH 13.5
#define CPR 392.0
#define WHEEL_RADIUS 2
  int dist_r = get_encoder_ticks(0) / CPR;
  int dist_l = get_encoder_ticks(1) / CPR;
  return ((dist_r - dist_l) / WIDTH);
}

static void calculate_encoder_odemetry() {
#define WIDTH 13.5
#define CPR 392.0
#define WHEEL_RADIUS 2

  int dist_r = get_encoder_ticks(0) / CPR;
  int dist_l = get_encoder_ticks(1) / CPR;
  printf("dist_r: %d dist_l: %d\n", dist_r, dist_l);
  int theta = (dist_l - dist_r) / WIDTH;
  printf("theta: %d\n", theta);
  int arc_length = ((M_PI * theta) * (WIDTH * WIDTH) / (8));
}

bool init_localization(const unsigned char gyro1, unsigned short multiplier,
                       int start_x, int start_y, int start_theta) {
  g1 = gyroInit(gyro1, multiplier);
  // init state matrix

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

7.81 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

debug_message	the message

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

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

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

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

error_message	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

info_message	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

use_lcd	whether to use the lcd
lcd	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

warning_message	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

```
#include "log.h"

unsigned int log_level = INFO;
static FILE *log_lcd = NULL;

void init_error(bool use_lcd, FILE *lcd) {
  if (use_lcd) {
    lcdInit(lcd);
    log_lcd = lcd;
    lcdClear(log_lcd);
    printf("LCD Init\n");
  }
}

static void log_info(const char *s, const char *mess) {
  printf("[%s]: %s\n", s, mess);
  lcdSetBacklight(log_lcd, false);
  lcdClear(log_lcd);
  lcdPrint(log_lcd, TOP_ROW, s);
  lcdPrint(log_lcd, BOTTOM_ROW, mess);
}
```

```
void error(const char *error_message) {
  if (log_level > NONE)
    log_info("ERROR", error_message);
}

void warning(const char *warning_message) {
  if (log_level > WARNING)
    log_info("WARNING", warning_message);
}

void info(const char *info_message) {
  if (log_level > ERROR) {
    log_info("INFO", info_message);
  }
}

void debug(const char *debug_message) {
  if (log_level > ERROR) {
    printf("[INFO]: %s\n", debug_message);
  }
}
```

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

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

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

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

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

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

- **matrix** ∗ **multiplyMatrix** (**matrix** ∗a, **matrix** ∗b)

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

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

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

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

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

  returns the transpose matrix.

7.83.1 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

m	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

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

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

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

Parameters

a	the first matrix
b	the second matrix

Returns

the matrix result

Definition at line **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

a	the first matrix
the	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

the	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

n	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

width	The width of the matrix
height	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

a	the first matrix
b	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

the	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

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

m	the matrix to scale
the	value to scale by

Returns

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

Definition at line **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

the	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

```c
#include "matrix.h"
#include "log.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void assert(int assertion, const char *message) {
  if (assertion == 0) {
    error(message);
    exit(1);
  }
}

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

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

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

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

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

  return out;
}

matrix *copyMatrix(matrix *m) { return scaleMatrix(m, 1); }

void freeMatrix(matrix *m) {
  if (m != NULL) {
    if (m->data != NULL) {
      free(m->data);
      m->data = NULL;
    }
    free(m);
  }
  return;
}

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

matrix *identityMatrix(int n) {
  int i;
  matrix *out;
  double *ptr;

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

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

  return out;
}

double traceMatrix(matrix *m) {
  int i;
  int size;
  double *ptr = m->data;
  double sum = 0.0;

  if (m->height < m->width) {
    size = m->height;
  } else {
    size = m->width;
  }

  for (i = 0; i < size; i++) {
    sum += *ptr;
    ptr += m->width + 1;
  }

  return sum;
}

matrix *meanMatrix(matrix *m) {
  int i, j;
  matrix *out;

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

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

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

matrix *covarianceMatrix(matrix *m) {
  int i, j, k = 0;
  matrix *out;
  matrix *mean;
  double *ptrA;
  double *ptrB;
  double *ptrOut;

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

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

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

  freeMatrix(mean);
  return out;
}

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

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

  return out;
}

matrix *multiplyMatrix(matrix *a, matrix *b) {
  int i, j, k;
  matrix *out;
  double *ptrOut;
  double *ptrA;
  double *ptrB;

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

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

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

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

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

  return out;
}

matrix *scaleMatrix(matrix *m, double value) {
  int i, elements = m->width * m->height;
  matrix *out = makeMatrix(m->width, m->height);
  double *ptrM = m->data;
  double *ptrOut = out->data;

  for (i = 0; i < elements; i++) {
    *(ptrOut++) = *(ptrM++) * value;
  }

  return out;
}

void rowSwap(matrix *a, int p, int q) {
  int i;
  double temp;
  double *pRow;
  double *qRow;

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

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

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

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

  return;
}
```

```

matrix *dotProductMatrix(matrix *a, matrix *b) {
  matrix *out;
  double *ptrOut;
  double *ptrA;
  double *ptrB;
  int i, j, k;

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

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

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

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

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

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

  return out;
}

matrix *dotDiagonalMatrix(matrix *a, matrix *b) {
  matrix *out;
  double *ptrOut;
  double *ptrA;
  double *ptrB;
  int i, j;

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

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

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

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

  return out;
}
```

7.85 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 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

7.85.1 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

rtn	the string to be written to
menu	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**, **ftoaa()**, **INT_TYPE**, **menu_t::length**, **max()**, **menu_t::max**, **menu_t::max_f**, **min()**, **menu_t::min**, **menu_t::min_f**, **menu_t::options**, **menu_t::step**, **menu_t::step_f**, **STRING_TYPE**, and **menu_t::type**.

Referenced by **display_menu()**.

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

menu	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

menu	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

type	the type of menu

See also

menu_type (p. 100)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **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

type	the type of menu

See also

menu_type (p. 100)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line **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

type	the type of menu

See also

menu_type (p. 100)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

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

static void calculate_current_display(char *rtn, menu_t *menu) {
  if (menu->type == STRING_TYPE) {
    int index = menu->current % menu->length;
    sprintf(rtn, "%s", menu->options[index]);
    printf("%s\n", rtn);
    return;
  }
  if (menu->type == INT_TYPE) {
    int step = (menu->step);
    int min = (menu->min);
    int max = (menu->max);
    int value = menu->current * step;
    if (value < min) {
      value = min;
      menu->current++;
    }
    if (value > max) {
      value = max;
      menu->current--;
    }
    sprintf(rtn, "%d", value);
  }
  if (menu->type == FLOAT_TYPE) {
    float step = (menu->step_f);
    float min = (menu->min_f);
    float max = (menu->max_f);
    float value = menu->current * step;
    value = value < min ? min : value;
    value = value > max ? max : value;

    ftoaa(value, rtn, 5);
  }
}

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

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

void denint_menu(menu_t *menu) {
  free(menu->prompt);
  if (menu->options != NULL)
    free(menu->options);
  free(menu);
}
```

7.87 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

```
#include "mobile_goal_intake.h"
#include "log.h"
#include "partner.h"

void set_intake_motor(int n) { set_motor_immediate(INTAKE_MOTOR, n); }

void lower_intake() { set_intake_motor(-100); }

void raise_intake() { set_intake_motor(100); }

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

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

```
#include "partner.h"

static enum CONTROLL_MODE mode = MAIN_CONTROLLER_MODE;

enum CONTROLL_MODE get_mode() { return mode; }

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

7.93 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

```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

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

Author

Chris Jerrett

Date

9/14/17

Definition at line **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

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

Author

Chris Jerrett

Date

9/14/17

Definition at line **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

```
#include "slew.h"
#include "log.h"

static Mutex speeds_mutex;

static int motors_set_speeds[10];
static int motors_curr_speeds[10];

static TaskHandle slew = NULL; // TaskHandle is of type void*

static bool initialized = false;

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

void init_slew() {
  if (initialized) {
    warning("Trying to init already init slew");
  }
  memset(motors_set_speeds, 0, sizeof(int) * 10);
  memset(motors_curr_speeds, 0, sizeof(int) * 10);
  motorStopAll();
  speeds_mutex = mutexCreate();
  slew = taskRunLoop(updateMotors, 100);
  initialized = true;
}

void deinitslew() {
  taskDelete(slew);
  memset(motors_set_speeds, 0, sizeof(int) * 10);
  memset(motors_curr_speeds, 0, sizeof(int) * 10);
  initialized = false;
}

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

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

7.97 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

button	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

button	The button to detect from the Buttons enumeration (see include/buttons.h).

Returns

true or false depending on if there was a change in button state.

Example code:

```
...
if(buttonIsNewPress(JOY1_8D))
    digitalWrite(1, !digitalRead(1));
...
```

Definition at line **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

```

#include "toggle.h"

bool buttonPressed[27];

void buttonInit() {
  for (int i = 0; i < 27; i++)
    buttonPressed[i] = false;
}

bool buttonGetState(button_t button) {
  bool currentButton = false;

  // Determine how to get the current button value (from what function) and
  // where it is, then get it.
  if (button < LCD_LEFT) {
    // button is a joystick button
    unsigned char joystick;
    unsigned char buttonGroup;
    unsigned char buttonLocation;

    button_t newButton;
    if (button <= 11) {
      // button is on joystick 1
      joystick = 1;
      newButton = button;
    } else {
      // button is on joystick 2
      joystick = 2;
      // shift button down to joystick 1 buttons in order to
      // detect which button on joystick is queried
      newButton = (button_t)(button - 12);
    }

    switch (newButton) {
    case 0:
      buttonGroup = 5;
      buttonLocation = JOY_DOWN;
      break;
    case 1:
      buttonGroup = 5;
      buttonLocation = JOY_UP;
      break;
    case 2:
      buttonGroup = 6;
      buttonLocation = JOY_DOWN;
      break;
    case 3:
      buttonGroup = 6;
      buttonLocation = JOY_UP;
      break;
    case 4:
      buttonGroup = 7;
      buttonLocation = JOY_UP;
      break;
    case 5:
      buttonGroup = 7;
      buttonLocation = JOY_LEFT;
      break;
    case 6:
      buttonGroup = 7;
      buttonLocation = JOY_RIGHT;
      break;
    case 7:
      buttonGroup = 7;
      buttonLocation = JOY_DOWN;
      break;
    case 8:
      buttonGroup = 8;
      buttonLocation = JOY_UP;
      break;
    case 9:
      buttonGroup = 8;
      buttonLocation = JOY_LEFT;
      break;
    case 10:
      buttonGroup = 8;
      buttonLocation = JOY_RIGHT;
      break;
    case 11:
      buttonGroup = 8;
      buttonLocation = JOY_DOWN;
      break;
    default:
      break;
    }
    currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
  } else {
    // button is on LCD
    if (button == LCD_LEFT)
      currentButton = (lcdReadButtons(uart1) == LCD_BTN_LEFT);

    if (button == LCD_CENT)
      currentButton = (lcdReadButtons(uart1) == LCD_BTN_CENTER);

    if (button == LCD_RIGHT)
      currentButton = (lcdReadButtons(uart1) == LCD_BTN_RIGHT);
  }
  return currentButton;
}

bool buttonIsNewPress(button_t button) {
  bool currentButton = buttonGetState(button);

  if (!currentButton) // buttons is not currently pressed
    buttonPressed[button] = false;

  if (currentButton && !buttonPressed[button]) {
    // button is currently pressed and was not detected as being pressed during
    // last check
    buttonPressed[button] = true;
    return true;
  } else
    return false; // button is not pressed or was already detected
}
```

7.99 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

a	the float
buffer	the string the float will be written to.
precision	digits after the decimal to write

Author

Christian DeSimone

Date

9/26/2017

Definition at line **55** of file **vlib.c**.

References **itoaa()**.

Referenced by **calculate_current_display()**.

7.99.1.2 itoaa()

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

converts a int to string.

Parameters

a	the integer
buffer	the string the int will be written to.
digits	the number of digits to be written

Returns

the digits

Author

Chris Jerrett, Christian DeSimone

Date

9/9/2017

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

References **reverse()**.

Referenced by **ftoaa()**.

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

str	the string to reverse
len	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
```

```
int itoaa(int a, char *buffer, int digits) {
  int i = 0;
  while (a) {
    buffer[i++] = (a % 10) + '0';
    a = a / 10;
  }

  // If number of digits required is more, then
  // add 0s at the beginning
  while (i < digits)
    buffer[i++] = '0';

  reverse(buffer, i);
  buffer[i] = '\0';
  return i;
}

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

  // Extract integer part
  int ipart = (int)a;

  // Extract floating part
  float fpart = a - (float)ipart;

  // convert integer part to string
  int i = itoaa(ipart, buffer, 0);

  // check for display option after point
  if (precision != 0) {
    buffer[i] = '.'; // add dot

    // Get the value of fraction part up to given num.
    // of points after dot. The third parameter is needed
    // to handle cases like 233.007
    fpart = fpart * pow(10, precision);

    itoaa((int)fpart, buffer + i + 1, precision);
  }
}
```

7.101 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 cordinated to polar coordinates.
- struct **polar_cord** **cartesian_to_polar** (float x, float y)

  Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- int **max** (int a, int b)

  the min of two values
- int **min** (int a, int b)

  the min of two values
- double **sind** (double angle)

  sine of a angle in degrees

7.101.1 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 cordinated to polar coordinates.

Author

Christian Desimone

Date

9/8/2017

Parameters

cords	the cartesian cords

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 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

x	float value of the x cartesian coordinate.
y	float value of the y cartesian coordinate.

Returns

a struct containing the angle and magnitude.

See also

polar_cord (p. 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

a	the first
b	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

a	the first
b	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.magnitue = 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) {
  double angleradians = angle * M_PI / 180.0f;
  return sin(angleradians);
}

int min(int a, int b) {
  if (a < b)
    return a;
  return b;
}

int max(int a, int b) {
  if (a > b)
    return a;
  return b;
}
```

7.103 test_code/testMath.py File Reference

Namespaces

- **testMath**

Functions

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

7.104 testMath.py

```
from math import *

def test(l1, l2):
    print(l1, l2)
    print("\n")
    theta = l1-l2
    x = ((l2)/(theta) + .5) - ((l2)/(theta) + .5) * cos(theta)
    y = ((l2)/(theta) + .5) * sin(theta)
    print(x)
    print(y)
    print(degrees(theta))
    print("\n")
    print("\n")

test(1.0, .5)
test(.5, 1.0)
test(2.0, .5)
test(.5, 2.0)
test(1.0, 3.5)
test(3.5, 1.0)
test(5, .3)
test(.3, 5)
test(1.0, 0)
```

7.105 testMath.py File Reference

Namespaces

- **testMath**

Functions

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

7.106 testMath.py

```
from math import *

def test(l1, l2):
    print(l1, l2)
    print("\n")
    theta = l1-l2
    x = ((l2)/(theta) + .5) - ((l2)/(theta) + .5) * cos(theta)
    y = ((l2)/(theta) + .5) * sin(theta)
    print(x)
    print(y)
    print(degrees(theta))
    print("\n")
    print("\n")

test(1.0, .5)
test(.5, 1.0)
test(2.0, .5)
test(.5, 2.0)
test(1.0, 3.5)
test(3.5, 1.0)
test(5, .3)
test(.3, 5)
test(1.0, 0)
```