

Vex Team A

1.1.0

Generated by Doxygen 1.8.13

Contents

Chapter 1

InTheZoneA

Team A code for In The Zone

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

[testMath](#) ??

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

_matrix	??
accelerometer_odometry	??
cord	A struct that contains cartesian coordinates	??
encoder_odometry	??
lcd_buttons	State of the lcd buttons	??
location	??
menu_t	Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via deinit_menu	??
polar_cord	A struct that contains polar coordinates	??

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

testMath.py	??
include/API.h	
Provides the high-level user functionality intended for use by typical VEX Cortex programmers	??
include/auto.h	??
include/battery.h	??
include/claw.h	??
include/controller.h	
Controller definitions, macros	??
include/drive.h	
Drive base definitions and enumerations	??
include/encoders.h	
Wrapper around encoder functions	??
include/lcd.h	
LCD wrapper functions and macros	??
include/lifter.h	??
include/localization.h	??
include/log.h	
Contains logging functions	??
include/main.h	
Header file for global functions	??
None of the matrix operations below change the input matrices if an input is required. They all return a new matrix with the new changes. Because memory issues are so prevalent, be sure to use the function to reclaim some of that memory ??	
include/menu.h	
Contains menu functionality and abstraction	??
include/mobile_goal_intake.h	??
include/motor_ports.h	??
include/partner.h	??
include/potentiometer.h	??
include/sensor_ports.h	??
include/slew.h	
Contains the slew rate controller wrapper for the motors	??
include/toggle.h	??
include/vlib.h	
Contains misc helpful functions	??

include/vmath.h		
	Vex Specific Math Functions, includes: Cartesian to polar coordinates	??
src/auto.c	File for autonomous code	??
src/battery.c		??
src/claw.c		??
src/controller.c		??
src/drive.c		??
src/encoders.c		??
src/init.c	File for initialization code	??
src/lcd.c		??
src/lifter.c		??
src/localization.c		??
src/log.c		??
src/matrix.c		??
src/menu.c		??
src/mobile_goal_intake.c		??
src/opcontrol.c	File for operator control code	??
src/partner.c		??
src/slew.c		??
src/toggle.c		??
src/vlib.c		??
src/vmath.c		??

Chapter 5

Namespace Documentation

5.1 testMath Namespace Reference

Functions

- def `test (l1, l2)`

5.1.1 Function Documentation

5.1.1.1 `test()`

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

Definition at line 3 of file testMath.py.

References `print()`.

```
3 def test(l1, l2):
4     print(l1, l2)
5     print("\n")
6     theta = l1-l2
7     x = ((l2)/(theta) + .5) - ((l1)/(theta) + .5) * cos(theta)
8     y = ((l2)/(theta) + .5) * sin(theta)
9     print(x)
10    print(y)
11    print(degrees(theta))
12    print("\n")
13    print("\n")
14
15 test(1.0, .5)
16 test(.5, 1.0)
17 test(2.0, .5)
18 test(.5, 2.0)
19 test(1.0, 3.5)
20 test(3.5, 1.0)
21 test(5, .3)
22 test(.3, 5)
23 test(1.0, 0)
24
```

Here is the call graph for this function:



Chapter 6

Data Structure Documentation

6.1 _matrix Struct Reference

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

Data Fields

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

6.1.1 Detailed Description

Definition at line 11 of file matrix.h.

6.1.2 Field Documentation

6.1.2.1 data

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

Definition at line 14 of file matrix.h.

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

6.1.2.2 height

```
int _matrix::height
```

Definition at line 12 of file matrix.h.

Referenced by covarianceMatrix(), dotDiagonalMatrix(), dotProductMatrix(), makeMatrix(), meanMatrix(), multiply←Matrix(), printMatrix(), rowSwap(), scaleMatrix(), traceMatrix(), and transposeMatrix().

6.1.2.3 width

```
int _matrix::width
```

Definition at line 13 of file matrix.h.

Referenced by covarianceMatrix(), dotDiagonalMatrix(), dotProductMatrix(), makeMatrix(), meanMatrix(), multiply←Matrix(), printMatrix(), rowSwap(), scaleMatrix(), traceMatrix(), and transposeMatrix().

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

- [include/matrix.h](#)

6.2 accelerometer_odometry Struct Reference

Data Fields

- double x
- double y

6.2.1 Detailed Description

Definition at line 17 of file localization.c.

6.2.2 Field Documentation

6.2.2.1 x

```
double accelerometer_odometry::x
```

Definition at line 18 of file localization.c.

6.2.2.2 y

```
double accelerometer_odometry::y
```

Definition at line 19 of file localization.c.

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

- [src/localization.c](#)

6.3 cord Struct Reference

A struct that contains cartesian coordinates.

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

Data Fields

- float [x](#)
- float [y](#)

6.3.1 Detailed Description

A struct that contains cartesian coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line 32 of file vmath.h.

6.3.2 Field Documentation

6.3.2.1 x

```
float cord::x
```

the x coordinate

Definition at line 34 of file vmath.h.

Referenced by [get_joystick_cord\(\)](#), and [update_drive_motors\(\)](#).

6.3.2.2 y

```
float cord::y
```

the y coordinate

Definition at line 36 of file vmath.h.

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

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

- [include/vmath.h](#)

6.4 encoder_odemtry Struct Reference

Data Fields

- `double theta`
- `double x`
- `double y`

6.4.1 Detailed Description

Definition at line 11 of file localization.c.

6.4.2 Field Documentation

6.4.2.1 theta

```
double encoder_odemtry::theta
```

Definition at line 14 of file localization.c.

Referenced by `integrate_gyro_w()`.

6.4.2.2 x

```
double encoder_odemtry::x
```

Definition at line 12 of file localization.c.

6.4.2.3 y

```
double encoder_odemtry::y
```

Definition at line 13 of file localization.c.

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

- [src/localization.c](#)

6.5 lcd_buttons Struct Reference

represents the state of the lcd buttons

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

Data Fields

- [button_state left](#)
- [button_state middle](#)
- [button_state right](#)

6.5.1 Detailed Description

represents the state of the lcd buttons

Author

Chris Jerrett

Date

9/9/2017

Definition at line 48 of file lcd.h.

6.5.2 Field Documentation

6.5.2.1 left

```
button_state lcd_buttons::left
```

Definition at line 49 of file lcd.h.

Referenced by [lcd_get_pressed_buttons\(\)](#).

6.5.2.2 middle

```
button_state lcd_buttons::middle
```

Definition at line 50 of file lcd.h.

Referenced by `lcd_get_pressed_buttons()`.

6.5.2.3 right

```
button_state lcd_buttons::right
```

Definition at line 51 of file lcd.h.

Referenced by `lcd_get_pressed_buttons()`.

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

- [include/lcd.h](#)

6.6 location Struct Reference

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

Data Fields

- int `theta`
- int `x`
- int `y`

6.6.1 Detailed Description

Definition at line 11 of file localization.h.

6.6.2 Field Documentation

6.6.2.1 theta

```
int location::theta
```

Definition at line 14 of file localization.h.

6.6.2.2 x

```
int location::x
```

Definition at line 12 of file localization.h.

6.6.2.3 y

```
int location::y
```

Definition at line 13 of file localization.h.

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

- [include/localization.h](#)

6.7 menu_t Struct Reference

Represents a specific instance of a menu. Will cause a memory leak if not deinitialized via `deinit_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 or 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 or 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 or decrease with each press. Defaults to 1.0f
- enum **menu_type** type
contains the type of menu.

6.7.1 Detailed Description

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

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#)
[menu_t](#)
[create_menu](#)
[init_menu](#)
[display_menu](#)
[menu_type](#)
[denint_menu](#)

Definition at line 65 of file `menu.h`.

6.7.2 Field Documentation

6.7.2.1 current

`int menu_t::current`

contains the current index of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line 139 of file `menu.h`.

Referenced by `calculate_current_display()`, and `display_menu()`.

6.7.2.2 length

`unsigned int menu_t::length`

contains the length of options `char**`.

Author

Chris Jerrett

Date

9/8/17

Definition at line 85 of file `menu.h`.

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

6.7.2.3 max

`int menu_t::max`

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 101 of file `menu.h`.

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

6.7.2.4 max_f

`float menu_t::max_f`

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 125 of file `menu.h`.

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

6.7.2.5 min

int menu_t::min

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 93 of file menu.h.

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

6.7.2.6 min_f

float menu_t::min_f

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 117 of file menu.h.

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

6.7.2.7 options

char** menu_t::options

contains the array of string options.

Author

Chris Jerrett

Date

9/8/17

Definition at line 78 of file menu.h.

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

6.7.2.8 prompt

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

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 146 of file menu.h.

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

6.7.2.9 step

```
int menu_t::step
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 109 of file menu.h.

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

6.7.2.10 step_f

```
float menu_t::step_f
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 133 of file menu.h.

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

6.7.2.11 type

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

contains the type of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line 71 of file menu.h.

Referenced by calculate_current_display(), and create_menu().

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

- [include/menu.h](#)

6.8 polar_cord Struct Reference

A struct that contains polar coordinates.

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

Data Fields

- float [angle](#)
- float [magnitue](#)

6.8.1 Detailed Description

A struct that contains polar coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line 20 of file vmath.h.

6.8.2 Field Documentation

6.8.2.1 angle

```
float polar_cord::angle
```

the angle of the vector

Definition at line 22 of file vmath.h.

Referenced by cartesian_to_polar().

6.8.2.2 magnitue

```
float polar_cord::magnitue
```

the magnitude of the vector

Definition at line 24 of file vmath.h.

Referenced by cartesian_to_polar().

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

- include/vmath.h

Chapter 7

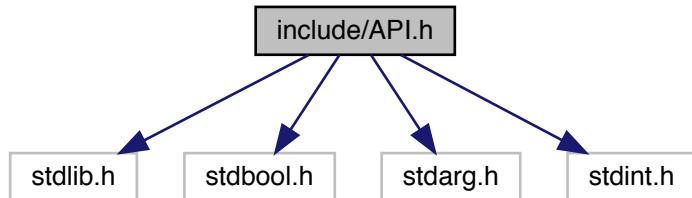
File Documentation

7.1 include/API.h File Reference

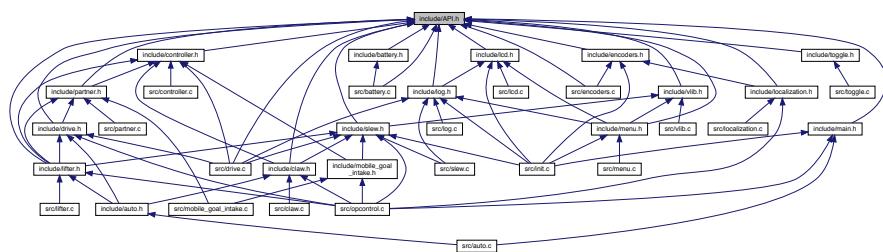
Provides the high-level user functionality intended for use by typical VEX Cortex programmers.

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdarg.h>
#include <stdint.h>
```

Include dependency graph for API.h:



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



Macros

- #define ACCEL_X 5
- #define ACCEL_Y 6
- #define BOARD_NR_ADC_PINS 8
- #define BOARD_NR_GPIO_PINS 27
- #define EOF ((int)-1)
- #define FILE_PROS_FILE
- #define HIGH 1
- #define IME_ADDR_MAX 0x1F
- #define INPUT 0x0A
- #define INPUT_ANALOG 0x00
- #define INPUT_FLOATING 0x04
- #define INTERRUPT_EDGE_BOTH 3
- #define INTERRUPT_EDGE_FALLING 2
- #define INTERRUPT_EDGE_RISING 1
- #define JOY_DOWN 1
- #define JOY_LEFT 2
- #define JOY_RIGHT 8
- #define JOY_UP 4
- #define LCD_BTN_CENTER 2
- #define LCD_BTN_LEFT 1
- #define LCD_BTN_RIGHT 4
- #define LOW 0
- #define OUTPUT 0x01
- #define OUTPUT_OD 0x05
- #define SEEK_CUR 1
- #define SEEK_END 2
- #define SEEK_SET 0
- #define SERIAL_8N1 0x0000
- #define SERIAL_DATABITS_8 0x0000
- #define SERIAL_DATABITS_9 0x1000
- #define SERIAL_PARITY_EVEN 0x0400
- #define SERIAL_PARITY_NONE 0x0000
- #define SERIAL_PARITY_ODD 0x0600
- #define SERIAL_STOPBITS_1 0x0000
- #define SERIAL_STOPBITS_2 0x2000
- #define stdin ((PROS_FILE *)3)
- #define stdout ((PROS_FILE *)3)
- #define TASK_DEAD 0
- #define TASK_DEFAULT_STACK_SIZE 512
- #define TASK_MAX 16
- #define TASK_MAX_PRIORITIES 6
- #define TASK_MINIMAL_STACK_SIZE 64
- #define TASK_PRIORITY_DEFAULT 2
- #define TASK_PRIORITY_HIGHEST (TASK_MAX_PRIORITIES - 1)
- #define TASK_PRIORITY_LOWEST 0
- #define TASK_RUNNABLE 2
- #define TASK_RUNNING 1
- #define TASK_SLEEPING 3
- #define TASK_SUSPENDED 4
- #define uart1 ((PROS_FILE *)1)
- #define uart2 ((PROS_FILE *)2)
- #define ULTRA_BAD_RESPONSE -1

Typedefs

- `typedef void * Encoder`
- `typedef void * Gyro`
- `typedef void(* InterruptHandler) (unsigned char pin)`
- `typedef void * Mutex`
- `typedef int PROS_FILE`
- `typedef void * Semaphore`
- `typedef void(* TaskCode) (void *)`
- `typedef void * TaskHandle`
- `typedef void * Ultrasonic`

Functions

- `void __attribute__ ((format(printf, 3, 4))) lcdPrint(PROS_FILE *lcdPort`
- `int analogCalibrate (unsigned char channel)`
- `int analogRead (unsigned char channel)`
- `int analogReadCalibrated (unsigned char channel)`
- `int analogReadCalibratedHR (unsigned char channel)`
- `void delay (const unsigned long time)`
- `void delayMicroseconds (const unsigned long us)`
- `bool digitalRead (unsigned char pin)`
- `void digitalWrite (unsigned char pin, bool value)`
- `int encoderGet (Encoder enc)`
- `Encoder encoderInit (unsigned char portTop, unsigned char portBottom, bool reverse)`
- `void encoderReset (Encoder enc)`
- `void encoderShutdown (Encoder enc)`
- `void fclose (PROS_FILE *stream)`
- `int fcount (PROS_FILE *stream)`
- `int fdelete (const char *file)`
- `int feof (PROS_FILE *stream)`
- `int fflush (PROS_FILE *stream)`
- `int fgetc (PROS_FILE *stream)`
- `char * fgets (char *str, int num, PROS_FILE *stream)`
- `PROS_FILE * fopen (const char *file, const char *mode)`
- `void fprintf (const char *string, PROS_FILE *stream)`
- `int fprintf (PROS_FILE *stream, const char *formatString,...)`
- `int fputc (int value, PROS_FILE *stream)`
- `int fputs (const char *string, PROS_FILE *stream)`
- `size_t fread (void *ptr, size_t size, size_t count, PROS_FILE *stream)`
- `int fseek (PROS_FILE *stream, long int offset, int origin)`
- `long int ftell (PROS_FILE *stream)`
- `size_t fwrite (const void *ptr, size_t size, size_t count, PROS_FILE *stream)`
- `int getchar ()`
- `int gyroGet (Gyro gyro)`
- `Gyro gyroInit (unsigned char port, unsigned short multiplier)`
- `void gyroReset (Gyro gyro)`
- `void gyroShutdown (Gyro gyro)`
- `bool i2cRead (uint8_t addr, uint8_t *data, uint16_t count)`
- `bool i2cReadRegister (uint8_t addr, uint8_t reg, uint8_t *value, uint16_t count)`
- `bool i2cWrite (uint8_t addr, uint8_t *data, uint16_t count)`
- `bool i2cWriteRegister (uint8_t addr, uint8_t reg, uint16_t value)`
- `bool imeGet (unsigned char address, int *value)`
- `bool imeGetVelocity (unsigned char address, int *value)`

- `unsigned int imeInitializeAll ()`
- `bool imeReset (unsigned char address)`
- `void imeShutdown ()`
- `void ioClearInterrupt (unsigned char pin)`
- `void ioSetInterrupt (unsigned char pin, unsigned char edges, InterruptHandler handler)`
- `bool isAutonomous ()`
- `bool isEnabled ()`
- `bool isJoystickConnected (unsigned char joystick)`
- `bool isOnline ()`
- `int joystickGetAnalog (unsigned char joystick, unsigned char axis)`
- `bool joystickGetDigital (unsigned char joystick, unsigned char buttonGroup, unsigned char button)`
- `void lcdClear (PROS_FILE *lcdPort)`
- `void lcdInit (PROS_FILE *lcdPort)`
- `void unsigned char const char unsigned int lcdReadButtons (PROS_FILE *lcdPort)`
- `void lcdSetBacklight (PROS_FILE *lcdPort, bool backlight)`
- `void lcdSetText (PROS_FILE *lcdPort, unsigned char line, const char *buffer)`
- `void lcdShutdown (PROS_FILE *lcdPort)`
- `unsigned long micros ()`
- `unsigned long millis ()`
- `int motorGet (unsigned char channel)`
- `void motorSet (unsigned char channel, int speed)`
- `void motorStop (unsigned char channel)`
- `void motorStopAll ()`
- `Mutex mutexCreate ()`
- `void mutexDelete (Mutex mutex)`
- `bool mutexGive (Mutex mutex)`
- `bool mutexTake (Mutex mutex, const unsigned long blockTime)`
- `void pinMode (unsigned char pin, unsigned char mode)`
- `unsigned int powerLevelBackup ()`
- `unsigned int powerLevelMain ()`
- `void print (const char *string)`
- `int printf (const char *formatString,...)`
- `int putchar (int value)`
- `int puts (const char *string)`
- `Semaphore semaphoreCreate ()`
- `void semaphoreDelete (Semaphore semaphore)`
- `bool semaphoreGive (Semaphore semaphore)`
- `bool semaphoreTake (Semaphore semaphore, const unsigned long blockTime)`
- `void setTeamName (const char *name)`
- `int sprintf (char *buffer, size_t limit, const char *formatString,...)`
- `void speakerInit ()`
- `void speakerPlayArray (const char **songs)`
- `void speakerPlayRttl (const char *song)`
- `void speakerShutdown ()`
- `int sprint (char *buffer, const char *formatString,...)`
- `void standaloneModeEnable ()`
- `TaskHandle taskCreate (TaskCode taskCode, const unsigned int stackDepth, void *parameters, const unsigned int priority)`
- `void taskDelay (const unsigned long msToDelay)`
- `void taskDelayUntil (unsigned long *previousWakeTime, const unsigned long cycleTime)`
- `void taskDelete (TaskHandle taskToDelete)`
- `unsigned int taskGetCount ()`
- `unsigned int taskGetState (TaskHandle task)`
- `unsigned int taskPriorityGet (const TaskHandle task)`
- `void taskPrioritySet (TaskHandle task, const unsigned int newPriority)`

- void `taskResume` (`TaskHandle` taskToResume)
- `TaskHandle` `taskRunLoop` (`void(*fn)(void)`, const unsigned long increment)
- void `taskSuspend` (`TaskHandle` taskToSuspend)
- int `ultrasonicGet` (`Ultrasonic ult`)
- `Ultrasonic ultrasonicInit` (unsigned char portEcho, unsigned char portPing)
- void `ultrasonicShutdown` (`Ultrasonic ult`)
- void `uartInit` (`PROS_FILE *uart`, unsigned int baud, unsigned int flags)
- void `uartShutdown` (`PROS_FILE *uart`)
- void `wait` (const unsigned long time)
- void `waitFor` (unsigned long *previousWakeTime, const unsigned long time)
- void `watchdogInit` ()

Variables

- void unsigned char const char * `formatString`
- void unsigned char `line`

7.1.1 Detailed Description

Provides the high-level user functionality intended for use by typical VEX Cortex programmers.

This file should be included for you in the predefined stubs in each new VEX Cortex PROS project through the inclusion of "main.h". In any new C source file, it is advisable to include `main.h` instead of referencing `API.h` by name, to better handle any nomenclature changes to this file or its contents.

Copyright (c) 2011-2016, Purdue University ACM SIGBots. All rights reserved.

This Source Code Form is subject to the terms of the Mozilla Public License, v. 2.0. If a copy of the MPL was not distributed with this file, You can obtain one at <http://mozilla.org/MPL/2.0/>.

PROS contains FreeRTOS (<http://www.freertos.org>) whose source code may be obtained from <http://sourceforge.net/projects/freertos/files/> or on request.

7.1.2 Macro Definition Documentation

7.1.2.1 ACCEL_X

```
#define ACCEL_X 5
```

Analog axis for the X acceleration from the VEX Joystick.

Definition at line 56 of file API.h.

7.1.2.2 ACCEL_Y

```
#define ACCEL_Y 6
```

Analog axis for the Y acceleration from the VEX Joystick.

Definition at line 60 of file API.h.

7.1.2.3 BOARD_NR_ADC_PINS

```
#define BOARD_NR_ADC_PINS 8
```

There are 8 available analog I/O on the Cortex.

Definition at line 141 of file API.h.

7.1.2.4 BOARD_NR_GPIO_PINS

```
#define BOARD_NR_GPIO_PINS 27
```

There are 27 available I/O on the Cortex that can be used for digital communication.

This excludes the crystal ports but includes the Communications, Speaker, and Analog ports.

The motor ports are not on the Cortex and are thus excluded from this count. Pin 0 is the Speaker port, pins 1-12 are the standard Digital I/O, 13-20 are the Analog I/O, 21+22 are UART1, 23+24 are UART2, and 25+26 are the I2C port.

Definition at line 151 of file API.h.

7.1.2.5 EOF

```
#define EOF ((int)-1)
```

EOF is a value evaluating to -1.

Definition at line 846 of file API.h.

7.1.2.6 FILE

```
#define FILE PROS_FILE
```

For convenience, FILE is defined as PROS_FILE if it wasn't already defined. This provides backwards compatibility with PROS, but also allows libraries such as newlib to be incorporated into PROS projects. If you're not using C++/newlib, you can disregard this and just use FILE.

Definition at line 759 of file API.h.

7.1.2.7 HIGH

```
#define HIGH 1
```

Used for [digitalWrite\(\)](#) to specify a logic HIGH state to output.

In reality, using any non-zero expression or "true" will work to set a pin to HIGH.

Definition at line 157 of file API.h.

7.1.2.8 IME_ADDR_MAX

```
#define IME_ADDR_MAX 0x1F
```

IME addresses end at 0x1F. Actually using more than 10 (address 0x1A) encoders will cause unreliable communications.

Definition at line 458 of file API.h.

7.1.2.9 INPUT

```
#define INPUT 0x0A
```

[pinMode\(\)](#) state for digital input, with pullup.

This is the default state for the 12 Digital pins. The pullup causes the input to read as "HIGH" when unplugged, but is fairly weak and can safely be driven by most sources. Many VEX digital sensors rely on this behavior and cannot be used with INPUT_FLOATING.

Definition at line 172 of file API.h.

7.1.2.10 INPUT_ANALOG

```
#define INPUT_ANALOG 0x00
```

[pinMode\(\)](#) state for analog inputs.

This is the default state for the 8 Analog pins and the Speaker port. This only works on pins with analog input capabilities; use anywhere else results in undefined behavior.

Definition at line 179 of file API.h.

7.1.2.11 INPUT_FLOATING

```
#define INPUT_FLOATING 0x04
```

[pinMode\(\)](#) state for digital input, without pullup.

Beware of power consumption, as digital inputs left "floating" may switch back and forth and cause spurious interrupts.

Definition at line 186 of file API.h.

7.1.2.12 INTERRUPT_EDGE_BOTH

```
#define INTERRUPT_EDGE_BOTH 3
```

When used in [ioSetInterrupt\(\)](#), triggers an interrupt on both rising and falling edges (LOW to HIGH or HIGH to LOW).

Definition at line 327 of file API.h.

7.1.2.13 INTERRUPT_EDGE_FALLING

```
#define INTERRUPT_EDGE_FALLING 2
```

When used in [ioSetInterrupt\(\)](#), triggers an interrupt on falling edges (HIGH to LOW).

Definition at line 322 of file API.h.

7.1.2.14 INTERRUPT_EDGE_RISING

```
#define INTERRUPT_EDGE_RISING 1
```

When used in [ioSetInterrupt\(\)](#), triggers an interrupt on rising edges (LOW to HIGH).

Definition at line 318 of file API.h.

7.1.2.15 JOY_DOWN

```
#define JOY_DOWN 1
```

DOWN button (valid on channels 5, 6, 7, 8)

Definition at line 40 of file API.h.

Referenced by buttonGetState(), and updateIntake().

7.1.2.16 JOY_LEFT

```
#define JOY_LEFT 2
```

LEFT button (valid on channels 7, 8)

Definition at line 44 of file API.h.

Referenced by buttonGetState(), and update_control().

7.1.2.17 JOY_RIGHT

```
#define JOY_RIGHT 8
```

RIGHT button (valid on channels 7, 8)

Definition at line 52 of file API.h.

Referenced by buttonGetState(), and update_control().

7.1.2.18 JOY_UP

```
#define JOY_UP 4
```

UP button (valid on channels 5, 6, 7, 8)

Definition at line 48 of file API.h.

Referenced by buttonGetState(), and updateIntake().

7.1.2.19 LCD_BTN_CENTER

```
#define LCD_BTN_CENTER 2
```

CENTER button on LCD for use with [LcdReadButtons\(\)](#)

Definition at line 1144 of file API.h.

Referenced by [buttonGetState\(\)](#).

7.1.2.20 LCD_BTN_LEFT

```
#define LCD_BTN_LEFT 1
```

LEFT button on LCD for use with [LcdReadButtons\(\)](#)

Definition at line 1140 of file API.h.

Referenced by [buttonGetState\(\)](#).

7.1.2.21 LCD_BTN_RIGHT

```
#define LCD_BTN_RIGHT 4
```

RIGHT button on LCD for use with [LcdReadButtons\(\)](#)

Definition at line 1148 of file API.h.

Referenced by [buttonGetState\(\)](#).

7.1.2.22 LOW

```
#define LOW 0
```

Used for [digitalWrite\(\)](#) to specify a logic LOW state to output.

In reality, using a zero expression or "false" will work to set a pin to LOW.

Definition at line 163 of file API.h.

7.1.2.23 OUTPUT

```
#define OUTPUT 0x01
```

[pinMode\(\)](#) state for digital output, push-pull.

This is the mode which should be used to output a digital HIGH or LOW value from the Cortex. This mode is useful for pneumatic solenoid valves and VEX LEDs.

Definition at line 193 of file API.h.

7.1.2.24 OUTPUT_OD

```
#define OUTPUT_OD 0x05
```

[pinMode\(\)](#) state for open-drain outputs.

This is useful in a few cases for external electronics and should not be used for the VEX solenoid or LEDs.

Definition at line 200 of file API.h.

7.1.2.25 SEEK_CUR

```
#define SEEK_CUR 1
```

[fseek\(\)](#) is used in [fseek\(\)](#) to denote an relative position in bytes from the current file location.

Definition at line 861 of file API.h.

7.1.2.26 SEEK_END

```
#define SEEK_END 2
```

[fseek\(\)](#) is used in [fseek\(\)](#) to denote an absolute position in bytes from the end of the file. The offset will most likely be negative in this case.

Definition at line 868 of file API.h.

7.1.2.27 SEEK_SET

```
#define SEEK_SET 0
```

[fseek\(\)](#) is used in [fseek\(\)](#) to denote an absolute position in bytes from the start of the file.

Definition at line 854 of file API.h.

7.1.2.28 SERIAL_8N1

```
#define SERIAL_8N1 0x0000
```

Specifies the default serial settings when used in [uartInit\(\)](#)

Definition at line 793 of file API.h.

7.1.2.29 SERIAL_DATABITS_8

```
#define SERIAL_DATABITS_8 0x0000
```

Bit mask for [uartInit\(\)](#) for 8 data bits (typical)

Definition at line 765 of file API.h.

7.1.2.30 SERIAL_DATABITS_9

```
#define SERIAL_DATABITS_9 0x1000
```

Bit mask for [uartInit\(\)](#) for 9 data bits

Definition at line 769 of file API.h.

7.1.2.31 SERIAL_PARITY_EVEN

```
#define SERIAL_PARITY_EVEN 0x0400
```

Bit mask for [uartInit\(\)](#) for Even parity

Definition at line 785 of file API.h.

7.1.2.32 SERIAL_PARITY_NONE

```
#define SERIAL_PARITY_NONE 0x0000
```

Bit mask for [uartInit\(\)](#) for No parity (typical)

Definition at line 781 of file API.h.

7.1.2.33 SERIAL_PARITY_ODD

```
#define SERIAL_PARITY_ODD 0x0600
```

Bit mask for [uartInit\(\)](#) for Odd parity

Definition at line 789 of file API.h.

7.1.2.34 SERIAL_STOPBITS_1

```
#define SERIAL_STOPBITS_1 0x0000
```

Bit mask for [uartInit\(\)](#) for 1 stop bit (typical)

Definition at line 773 of file API.h.

7.1.2.35 SERIAL_STOPBITS_2

```
#define SERIAL_STOPBITS_2 0x2000
```

Bit mask for [uartInit\(\)](#) for 2 stop bits

Definition at line 777 of file API.h.

7.1.2.36 stdin

```
#define stdin ((PROS_FILE *)3)
```

The standard input stream uses the PC debug terminal.

Definition at line 832 of file API.h.

7.1.2.37 stdout

```
#define stdout ((PROS_FILE *)3)
```

The standard output stream uses the PC debug terminal.

Definition at line 828 of file API.h.

7.1.2.38 TASK_DEAD

```
#define TASK_DEAD 0
```

Constant returned from [taskGetState\(\)](#) when the task is dead or nonexistent.

Definition at line 1275 of file API.h.

7.1.2.39 TASK_DEFAULT_STACK_SIZE

```
#define TASK_DEFAULT_STACK_SIZE 512
```

The recommended stack size for a new task that does an average amount of work. This stack size is used for default tasks such as [autonomous\(\)](#).

This is probably OK for 4-5 levels of function calls and the use of [printf\(\)](#) with several arguments. Tasks requiring deep recursion or large local buffers will need a bigger stack.

Definition at line 1262 of file API.h.

7.1.2.40 TASK_MAX

```
#define TASK_MAX 16
```

Only this many tasks can exist at once. Attempts to create further tasks will not succeed until tasks end or are destroyed, AND the idle task cleans them up.

Changing this value will not change the limit without a kernel recompile. The idle task and VEX daemon task count against the limit. The user [autonomous\(\)](#) or teleop() also counts against the limit, so 12 tasks usually remain for other uses.

Definition at line 1232 of file API.h.

7.1.2.41 TASK_MAX_PRIORITIES

```
#define TASK_MAX_PRIORITIES 6
```

The maximum number of available task priorities, which run from 0 to 5.

Changing this value will not change the priority count without a kernel recompile.

Definition at line 1238 of file API.h.

7.1.2.42 TASK_MINIMAL_STACK_SIZE

```
#define TASK_MINIMAL_STACK_SIZE 64
```

The minimum stack depth for a task. Scheduler state is stored on the stack, so even if the task never uses the stack, at least this much space must be allocated.

Function calls and other seemingly innocent constructs may place information on the stack. Err on the side of a larger stack when possible.

Definition at line 1270 of file API.h.

7.1.2.43 TASK_PRIORITY_DEFAULT

```
#define TASK_PRIORITY_DEFAULT 2
```

The default task priority, which should be used for most tasks.

Default tasks such as [autonomous\(\)](#) inherit this priority.

Definition at line 1249 of file API.h.

7.1.2.44 TASK_PRIORITY_HIGHEST

```
#define TASK_PRIORITY_HIGHEST (TASK_MAX_PRIORITIES - 1)
```

The highest priority that can be assigned to a task. Unlike the lowest priority, this priority can be safely used without hampering interrupts. Beware of deadlock.

Definition at line 1254 of file API.h.

7.1.2.45 TASK_PRIORITY_LOWEST

```
#define TASK_PRIORITY_LOWEST 0
```

The lowest priority that can be assigned to a task, which puts it on a level with the idle task. This may cause severe performance problems and is generally not recommended.

Definition at line 1243 of file API.h.

7.1.2.46 TASK_RUNNABLE

```
#define TASK_RUNNABLE 2
```

Constant returned from [taskGetState\(\)](#) when the task is exists and is available to run, but not currently running.

Definition at line 1284 of file API.h.

7.1.2.47 TASK_RUNNING

```
#define TASK_RUNNING 1
```

Constant returned from [taskGetState\(\)](#) when the task is actively executing.

Definition at line 1279 of file API.h.

7.1.2.48 TASK_SLEEPING

```
#define TASK_SLEEPING 3
```

Constant returned from [taskGetState\(\)](#) when the task is delayed or blocked waiting for a semaphore, mutex, or I/O operation.

Definition at line 1289 of file API.h.

7.1.2.49 TASK_SUSPENDED

```
#define TASK_SUSPENDED 4
```

Constant returned from [taskGetState\(\)](#) when the task is suspended using [taskSuspend\(\)](#).

Definition at line 1293 of file API.h.

7.1.2.50 uart1

```
#define uart1 ((PROS_FILE *)1)
```

UART 1 on the Cortex; must be opened first using [uartInit\(\)](#).

Definition at line 836 of file API.h.

Referenced by [buttonGetState\(\)](#).

7.1.2.51 uart2

```
#define uart2 ((PROS_FILE *)2)
```

UART 2 on the Cortex; must be opened first using [uartInit\(\)](#).

Definition at line 840 of file API.h.

7.1.2.52 ULTRA_BAD_RESPONSE

```
#define ULTRA_BAD_RESPONSE -1
```

This value is returned if the sensor cannot find a reasonable value to return.

Definition at line 650 of file API.h.

7.1.3 Typedef Documentation

7.1.3.1 Encoder

```
typedef void* Encoder
```

Reference type for an initialized encoder.

Encoder information is stored as an opaque pointer to a structure in memory; as this is a pointer type, it can be safely passed or stored by value.

Definition at line 605 of file API.h.

7.1.3.2 Gyro

```
typedef void* Gyro
```

Reference type for an initialized gyro.

Gyro information is stored as an opaque pointer to a structure in memory; as this is a pointer type, it can be safely passed or stored by value.

Definition at line 548 of file API.h.

7.1.3.3 InterruptHandler

```
typedef void(* InterruptHandler) (unsigned char pin)
```

Type definition for interrupt handlers. Such functions must accept one argument indicating the pin which changed.

Definition at line 332 of file API.h.

7.1.3.4 Mutex

```
typedef void* Mutex
```

Type by which mutexes are referenced.

As this is a pointer type, it can be safely passed or stored by value.

Definition at line 1306 of file API.h.

7.1.3.5 PROS_FILE

```
typedef int PROS_FILE
```

PROS_FILE is an integer referring to a stream for the standard I/O functions.

PROS_FILE * is the standard library method of referring to a file pointer, even though there is actually nothing there.

Definition at line 750 of file API.h.

7.1.3.6 Semaphore

```
typedef void* Semaphore
```

Type by which semaphores are referenced.

As this is a pointer type, it can be safely passed or stored by value.

Definition at line 1312 of file API.h.

7.1.3.7 TaskCode

```
typedef void(* TaskCode) (void *)
```

Type for defining task functions. Task functions must accept one parameter of type "void *"; they need not use it.

For example:

```
void MyTask(void *ignore) { while (1); }
```

Definition at line 1323 of file API.h.

7.1.3.8 TaskHandle

```
typedef void* TaskHandle
```

Type by which tasks are referenced.

As this is a pointer type, it can be safely passed or stored by value.

Definition at line 1300 of file API.h.

7.1.3.9 Ultrasonic

```
typedef void* Ultrasonic
```

Reference type for an initialized ultrasonic sensor.

Ultrasonic information is stored as an opaque pointer to a structure in memory; as this is a pointer type, it can be safely passed or stored by value.

Definition at line 658 of file API.h.

7.1.4 Function Documentation

7.1.4.1 __attribute__()

```
void __attribute__ (
    format (printf, 3, 4))
```

Prints the formatted string to the attached LCD.

The output string will be truncated as necessary to fit on the LCD screen, 16 characters wide. It is probably better to generate the string in a local buffer and use [lcdSetText\(\)](#) but this method is provided for convenience.

Parameters

<i>lcdPort</i>	the LCD to write, either uart1 or uart2
<i>line</i>	the LCD line to write, either 1 or 2
<i>formatString</i>	the format string as specified in fprintf()

7.1.4.2 analogCalibrate()

```
int analogCalibrate (
    unsigned char channel )
```

Calibrates the analog sensor on the specified channel.

This method assumes that the true sensor value is not actively changing at this time and computes an average from approximately 500 samples, 1 ms apart, for a 0.5 s period of calibration. The average value thus calculated is returned and stored for later calls to the [analogReadCalibrated\(\)](#) and [analogReadCalibratedHR\(\)](#) functions. These functions will return the difference between this value and the current sensor value when called.

Do not use this function in [initializeIO\(\)](#), or when the sensor value might be unstable (gyro rotation, accelerometer movement).

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

<i>channel</i>	the channel to calibrate from 1-8
----------------	-----------------------------------

Returns

the average sensor value computed by this function

7.1.4.3 analogRead()

```
int analogRead (
    unsigned char channel )
```

Reads an analog input channel and returns the 12-bit value.

The value returned is undefined if the analog pin has been switched to a different mode. This function is Wiring-compatible with the exception of the larger output range. The meaning of the returned value varies depending on the sensor attached.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

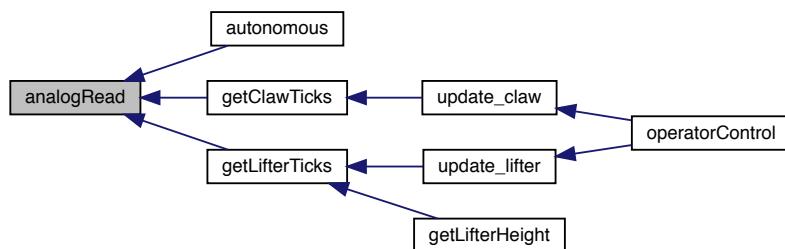
<i>channel</i>	the channel to read from 1-8
----------------	------------------------------

Returns

the analog sensor value, where a value of 0 reflects an input voltage of nearly 0 V and a value of 4095 reflects an input voltage of nearly 5 V

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

Here is the caller graph for this function:

**7.1.4.4 analogReadCalibrated()**

```
int analogReadCalibrated (
    unsigned char channel )
```

Reads the calibrated value of an analog input channel.

The [analogCalibrate\(\)](#) function must be run first on that channel. This function is inappropriate for sensor values intended for integration, as round-off error can accumulate causing drift over time. Use [analogReadCalibratedHR\(\)](#) instead.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

<i>channel</i>	the channel to read from 1-8
----------------	------------------------------

Returns

the difference of the sensor value from its calibrated default from -4095 to 4095

7.1.4.5 analogReadCalibratedHR()

```
int analogReadCalibratedHR (
    unsigned char channel )
```

Reads the calibrated value of an analog input channel 1-8 with enhanced precision.

The [analogCalibrate\(\)](#) function must be run first. This is intended for integrated sensor values such as gyros and accelerometers to reduce drift due to round-off, and should not be used on a sensor such as a line tracker or potentiometer.

The value returned actually has 16 bits of "precision", even though the ADC only reads 12 bits, so that errors induced by the average value being between two values come out in the wash when integrated over time. Think of the value as the true value times 16.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

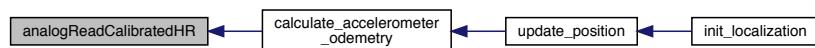
<i>channel</i>	the channel to read from 1-8
----------------	------------------------------

Returns

the difference of the sensor value from its calibrated default from -16384 to 16384

Referenced by [calculate_accelerometer_odometry\(\)](#).

Here is the caller graph for this function:



7.1.4.6 delay()

```
void delay (
    const unsigned long time )
```

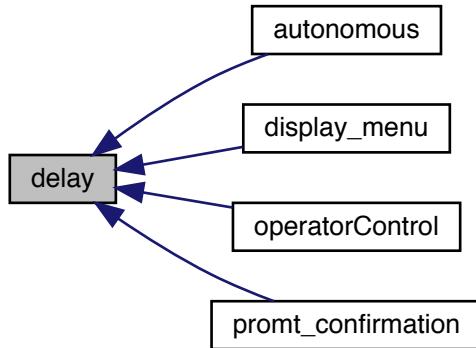
Wiring-compatible alias of [taskDelay\(\)](#).

Parameters

<i>time</i>	the duration of the delay in milliseconds (1 000 milliseconds per second)
-------------	---

Referenced by [autonomous\(\)](#), [display_menu\(\)](#), [operatorControl\(\)](#), and [prompt_confirmation\(\)](#).

Here is the caller graph for this function:



7.1.4.7 delayMicroseconds()

```
void delayMicroseconds (
    const unsigned long us )
```

Wait for approximately the given number of microseconds.

The method used for delaying this length of time may vary depending on the argument. The current task will always be delayed by at least the specified period, but possibly much more depending on CPU load. In general, this function is less reliable than [delay\(\)](#). Using this function in a loop may hog processing time from other tasks.

Parameters

<i>us</i>	the duration of the delay in microseconds (1 000 000 microseconds per second)
-----------	---

7.1.4.8 digitalRead()

```
bool digitalRead (
    unsigned char pin )
```

Gets the digital value (1 or 0) of a pin configured as a digital input.

If the pin is configured as some other mode, the digital value which reflects the current state of the pin is returned, which may or may not differ from the currently set value. The return value is undefined for pins configured as Analog inputs, or for ports in use by a Communications interface. This function is Wiring-compatible.

This function may not work properly if the VEX Cortex is tethered to a PC using the orange USB A to A cable and has no VEX 7.2V Battery connected and powered on, as the VEX Battery provides power to sensors.

Parameters

<i>pin</i>	the pin to read from 1-26
------------	---------------------------

Returns

true if the pin is HIGH, or false if it is LOW

7.1.4.9 digitalWrite()

```
void digitalWrite (
    unsigned char pin,
    bool value )
```

Sets the digital value (1 or 0) of a pin configured as a digital output.

If the pin is configured as some other mode, behavior is undefined. This function is Wiring-compatible.

Parameters

<i>pin</i>	the pin to write from 1-26
<i>value</i>	an expression evaluating to "true" or "false" to set the output to HIGH or LOW respectively, or the constants HIGH or LOW themselves

7.1.4.10 encoderGet()

```
int encoderGet (
    Encoder enc )
```

Gets the number of ticks recorded by the encoder.

There are 360 ticks in one revolution.

Parameters

<i>enc</i>	the Encoder object from encoderInit() to read
------------	---

Returns

the signed and cumulative number of counts since the last start or reset

7.1.4.11 encoderInit()

```
Encoder encoderInit (
    unsigned char portTop,
    unsigned char portBottom,
    bool reverse )
```

Initializes and enables a quadrature encoder on two digital ports.

Neither the top port nor the bottom port can be digital port 10. NULL will be returned if either port is invalid or the encoder is already in use. Initializing an encoder implicitly resets its count.

Parameters

<i>portTop</i>	the "top" wire from the encoder sensor with the removable cover side UP
<i>portBottom</i>	the "bottom" wire from the encoder sensor
<i>reverse</i>	if "true", the sensor will count in the opposite direction

Returns

an Encoder object to be stored and used for later calls to encoder functions

7.1.4.12 encoderReset()

```
void encoderReset (
    Encoder enc )
```

Resets the encoder to zero.

It is safe to use this method while an encoder is enabled. It is not necessary to call this method before stopping or starting an encoder.

Parameters

<i>enc</i>	the Encoder object from encoderInit() to reset
------------	--

7.1.4.13 encoderShutdown()

```
void encoderShutdown (
    Encoder enc )
```

Stops and disables the encoder.

Encoders use processing power, so disabling unused encoders increases code performance. The encoder's count will be retained.

Parameters

<code>enc</code>	the Encoder object from encoderInit() to stop
------------------	---

7.1.4.14 fclose()

```
void fclose (
    PROS_FILE * stream )
```

Closes the specified file descriptor. This function does not work on communication ports; use [uartShutdown\(\)](#) instead.

Parameters

<code>stream</code>	the file descriptor to close from fopen()
---------------------	---

7.1.4.15 fcount()

```
int fcount (
    PROS_FILE * stream )
```

Returns the number of characters that can be read without blocking (the number of characters available) from the specified stream. This only works for communication ports and files in Read mode; for files in Write mode, 0 is always returned.

This function may underestimate, but will not overestimate, the number of characters which meet this criterion.

Parameters

<code>stream</code>	the stream to read (stdin, uart1, uart2, or an open file in Read mode)
---------------------	--

Returns

the number of characters which meet this criterion; if this number cannot be determined, returns 0

7.1.4.16 fdelete()

```
int fdelete (
    const char * file )
```

Delete the specified file if it exists and is not currently open.

The file will actually be erased from memory on the next re-boot. A physical power cycle is required to purge deleted files and free their allocated space for new files to be written. Deleted files are still considered inaccessible to [fopen\(\)](#) in Read mode.

Parameters

<i>file</i>	the file name to erase
-------------	------------------------

Returns

0 if the file was deleted, or 1 if the file could not be found

7.1.4.17 feof()

```
int feof (
    PROS_FILE * stream )
```

Checks to see if the specified stream is at its end. This only works for communication ports and files in Read mode; for files in Write mode, 1 is always returned.

Parameters

<i>stream</i>	the channel to check (stdin, uart1, uart2, or an open file in Read mode)
---------------	--

Returns

0 if the stream is not at EOF, or 1 otherwise.

7.1.4.18 fflush()

```
int fflush (
    PROS_FILE * stream )
```

Flushes the data on the specified file channel open in Write mode. This function has no effect on a communication port or a file in Read mode, as these streams are always flushed as quickly as possible by the kernel.

Successful completion of an fflush function on a file in Write mode cannot guarantee that the file is valid until [fclose\(\)](#) is used on that file descriptor.

Parameters

<i>stream</i>	the channel to flush (an open file in Write mode)
---------------	---

Returns

0 if the data was successfully flushed, EOF otherwise

7.1.4.19 fgetc()

```
int fgetc (
    PROS_FILE * stream )
```

Reads and returns one character from the specified stream, blocking until complete.

Do not use [fgetc\(\)](#) on a VEX LCD port; deadlock may occur.

Parameters

<i>stream</i>	the stream to read (stdin, uart1, uart2, or an open file in Read mode)
---------------	--

Returns

the next character from 0 to 255, or -1 if no character can be read

7.1.4.20 fgets()

```
char* fgets (
    char * str,
    int num,
    PROS_FILE * stream )
```

Reads a string from the specified stream, storing the characters into the memory at str. Characters will be read until the specified limit is reached, a new line is found, or the end of file is reached.

If the stream is already at end of file (for files in Read mode), NULL will be returned; otherwise, at least one character will be read and stored into str.

Parameters

<i>str</i>	the location where the characters read will be stored
<i>num</i>	the maximum number of characters to store; at most (num - 1) characters will be read, with a null terminator ('\0') automatically appended
<i>stream</i>	the channel to read (stdin, uart1, uart2, or an open file in Read mode)

Returns

str, or NULL if zero characters could be read

7.1.4.21 fopen()

```
PROS_FILE* fopen (
    const char * file,
    const char * mode )
```

Opens the given file in the specified mode. The file name is truncated to eight characters. Only four files can be in use simultaneously in any given time, with at most one of those files in Write mode. This function does not work on communication ports; use [uartInit\(\)](#) instead.

mode can be "r" or "w". Due to the nature of the VEX Cortex memory, the "r+", "w+", and "a" modes are not supported by the file system.

Opening a file that does not exist in Read mode will fail and return NULL, but opening a new file in Write mode will create it if there is space. Opening a file that already exists in Write mode will destroy the contents and create a new blank file if space is available.

There are important considerations when using of the file system on the VEX Cortex. Reading from files is safe, but writing to files should only be performed when robot actuators have been stopped. PROS will attempt to continue to handle events during file writes, but most user tasks cannot execute during file writing. Powering down the VEX Cortex mid-write may cause file system corruption.

Parameters

<i>file</i>	the file name
<i>mode</i>	the file mode

Returns

a file descriptor pointing to the new file, or NULL if the file could not be opened

7.1.4.22 `fprint()`

```
void fprint (
    const char * string,
    PROS_FILE * stream )
```

Prints the simple string to the specified stream.

This method is much, much faster than [fprintf\(\)](#) and does not add a new line like [fputs\(\)](#). Do not use [fprint\(\)](#) on a VEX LCD port. Use [lcdSetText\(\)](#) instead.

Parameters

<i>string</i>	the string to write
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

7.1.4.23 `fprintf()`

```
int fprintf (
    PROS_FILE * stream,
    const char * formatString,
    ... )
```

Prints the formatted string to the specified output stream.

The specifiers supported by this minimalistic [printf\(\)](#) function are:

- %d : Signed integer in base 10 (int)
- %u : Unsigned integer in base 10 (unsigned int)
- %x, %X : Integer in base 16 (unsigned int, int)
- %p : Pointer (void *, int *, ...)
- %c : Character (char)
- %s : Null-terminated string (char *)
- %%: Single literal percent sign
- %f : Floating-point number

Specifiers can be modified with:

- 0: Zero-pad, instead of space-pad
- a.b: Make the field at least "a" characters wide. If "b" is specified for "%f", changes the number of digits after the decimal point
- -: Left-align, instead of right-align
- +: Always display the sign character (displays a leading "+" for positive numbers)
- l: Ignored for compatibility

Invalid format specifiers, or mismatched parameters to specifiers, cause undefined behavior. Other characters are written out verbatim. Do not use [fprintf\(\)](#) on a VEX LCD port. Use [lcdPrint\(\)](#) instead.

Parameters

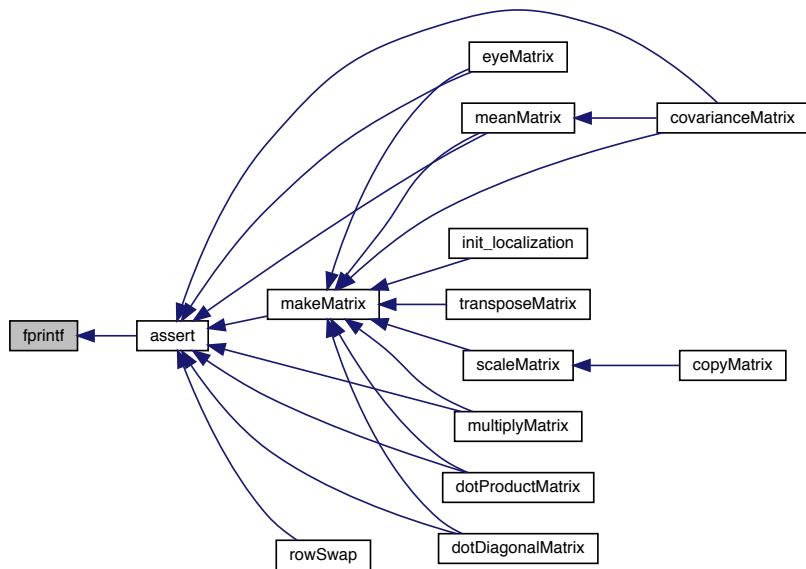
<i>stream</i>	the stream to write (stdout, uart1, or uart2)
<i>formatString</i>	the format string as specified above

Returns

the number of characters written

Referenced by assert().

Here is the caller graph for this function:

**7.1.4.24 fputc()**

```
int fputc (
    int value,
    PROS_FILE * stream )
```

Writes one character to the specified stream.

Do not use [fputc\(\)](#) on a VEX LCD port. Use [lcdSetText\(\)](#) instead.

Parameters

<code>value</code>	the character to write (a value of type "char" can be used)
<code>stream</code>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

Returns

the character written

7.1.4.25 fputs()

```
int fputs (
    const char * string,
    PROS_FILE * stream )
```

Behaves the same as the "fprint" function, and appends a trailing newline ("\n").

Do not use [fputs\(\)](#) on a VEX LCD port. Use [lcdSetText\(\)](#) instead.

Parameters

<i>string</i>	the string to write
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

Returns

the number of characters written, excluding the new line

7.1.4.26 fread()

```
size_t fread (
    void * ptr,
    size_t size,
    size_t count,
    PROS_FILE * stream )
```

Reads data from a stream into memory. Returns the number of bytes thus read.

If the memory at *ptr* cannot store (*size* * *count*) bytes, undefined behavior occurs.

Parameters

<i>ptr</i>	a pointer to where the data will be stored
<i>size</i>	the size of each data element to read in bytes
<i>count</i>	the number of data elements to read
<i>stream</i>	the stream to read (stdout, uart1, uart2, or an open file in Read mode)

Returns

the number of bytes successfully read

7.1.4.27 fseek()

```
int fseek (
    PROS_FILE * stream,
```

```
long int offset,
int origin )
```

Seeks within a file open in Read mode. This function will fail when used on a file in Write mode or on any communications port.

Parameters

<i>stream</i>	the stream to seek within
<i>offset</i>	the location within the stream to seek
<i>origin</i>	the reference location for offset: SEEK_CUR, SEEK_SET, or SEEK_END

Returns

0 if the seek was successful, or 1 otherwise

7.1.4.28 `fseek()`

```
long int fseek (
    PROS_FILE * stream )
```

Returns the current position of the stream. This function works on files in either Read or Write mode, but will fail on communications ports.

Parameters

<i>stream</i>	the stream to check
---------------	---------------------

Returns

the offset of the stream, or -1 if the offset could not be determined

7.1.4.29 `fwrite()`

```
size_t fwrite (
    const void * ptr,
    size_t size,
    size_t count,
    PROS_FILE * stream )
```

Writes data from memory to a stream. Returns the number of bytes thus written.

If the memory at *ptr* is not as long as (*size* * *count*) bytes, undefined behavior occurs.

Parameters

<i>ptr</i>	a pointer to the data to write
<i>size</i>	the size of each data element to write in bytes
<i>count</i>	the number of data elements to write
<i>stream</i>	the stream to write (stdout, uart1, uart2, or an open file in Write mode)

Returns

the number of bytes successfully written

7.1.4.30 `getchar()`

```
int getchar ( )
```

Reads and returns one character from "stdin", which is the PC debug terminal.

Returns

the next character from 0 to 255, or -1 if no character can be read

7.1.4.31 `gyroGet()`

```
int gyroGet (
    Gyro gyro )
```

Gets the current gyro angle in degrees, rounded to the nearest degree.

There are 360 degrees in a circle.

Parameters

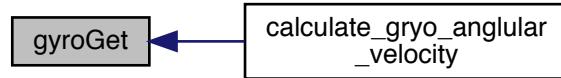
<i>gyro</i>	the Gyro object from gyroInit() to read
-------------	---

Returns

the signed and cumulative number of degrees rotated around the gyro's vertical axis since the last start or reset

Referenced by `calculate_gryo-angular_velocity()`.

Here is the caller graph for this function:



7.1.4.32 gyroInit()

```
Gyro gyroInit (
    unsigned char port,
    unsigned short multiplier )
```

Initializes and enables a gyro on an analog port.

NULL will be returned if the port is invalid or the gyro is already in use. Initializing a gyro implicitly calibrates it and resets its count. Do not move the robot while the gyro is being calibrated. It is suggested to call this function in [initialize\(\)](#) and to place the robot in its final position before powering it on.

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.

Parameters

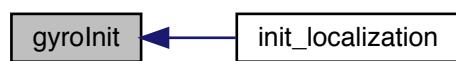
<i>port</i>	the analog port to use from 1-8
<i>multiplier</i>	an optional constant to tune the gyro readings; use 0 for the default value

Returns

a Gyro object to be stored and used for later calls to gyro functions

Referenced by [init_localization\(\)](#).

Here is the caller graph for this function:



7.1.4.33 gyroReset()

```
void gyroReset (
    Gyro gyro )
```

Resets the gyro to zero.

It is safe to use this method while a gyro is enabled. It is not necessary to call this method before stopping or starting a gyro.

Parameters

<i>gyro</i>	the Gyro object from gyroInit() to reset
-------------	--

7.1.4.34 gyroShutdown()

```
void gyroShutdown (
    Gyro gyro )
```

Stops and disables the gyro.

Gyros use processing power, so disabling unused gyros increases code performance. The gyro's position will be retained.

Parameters

<i>gyro</i>	the Gyro object from gyroInit() to stop
-------------	---

7.1.4.35 i2cRead()

```
bool i2cRead (
    uint8_t addr,
    uint8_t * data,
    uint16_t count )
```

i2cRead - Reads the specified number of data bytes from the specified 7-bit I2C address. The bytes will be stored at the specified location. Returns true if successful or false if failed. If only some bytes could be read, false is still returned.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Since most I2C devices use an 8-bit register architecture, this method has limited usefulness. Consider [i2cReadRegister](#) instead for the vast majority of applications.

7.1.4.36 i2cReadRegister()

```
bool i2cReadRegister (
    uint8_t addr,
    uint8_t reg,
    uint8_t * value,
    uint16_t count )
```

i2cReadRegister - Reads the specified amount of data from the given register address on the specified 7-bit I2C address. Returns true if successful or false if failed. If only some bytes could be read, false is still returned.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Most I2C devices support an auto-increment address feature, so using this method to read more than one byte will usually read a block of sequential registers. Try to merge reads to separate registers into a larger read using this function whenever possible to improve code reliability, even if a few intermediate values need to be thrown away.

7.1.4.37 i2cWrite()

```
bool i2cWrite (
    uint8_t addr,
    uint8_t * data,
    uint16_t count )
```

i2cWrite - Writes the specified number of data bytes to the specified 7-bit I2C address. Returns true if successful or false if failed. If only smoe bytes could be written, false is still returned.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Since most I2C devices use an 8-bit register architecture, this method is mostly useful for setting the register position (most devices remember the last-used address) or writing a sequence of bytes to one register address using an auto-increment feature. In these cases, the first byte written from the data buffer should have the register address to use.

7.1.4.38 i2cWriteRegister()

```
bool i2cWriteRegister (
    uint8_t addr,
    uint8_t reg,
    uint16_t value )
```

i2cWriteRegister - Writes the specified data byte to a register address on the specified 7-bit I2C address. Returns true if successful or false if failed.

The I2C address should be right-aligned; the R/W bit is automatically supplied.

Only one byte can be written to each register address using this method. While useful for the vast majority of I2C operations, writing multiple bytes requires the i2cWrite method.

7.1.4.39 imeGet()

```
bool imeGet (
    unsigned char address,
    int * value )
```

Gets the current 32-bit count of the specified IME.

Much like the count for a quadrature encoder, the tick count is signed and cumulative. The value reflects total counts since the last reset. Different VEX Motor Encoders have a different number of counts per revolution:

- 240.448 for the 269 IME
- 627.2 for the 393 IME in high torque mode (factory default)
- 392 for the 393 IME in high speed mode

If the IME address is invalid, or the IME has not been reset or initialized, the value stored in *value is undefined.

Parameters

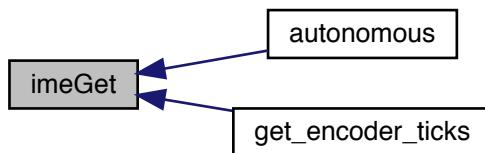
<code>address</code>	the IME address to fetch from 0 to IME_ADDR_MAX
<code>value</code>	a pointer to the location where the value will be stored (obtained using the "&" operator on the target variable name e.g. <code>imeGet (2, &counts)</code>)

Returns

true if the count was successfully read and the value stored in *value is valid; false otherwise

Referenced by `autonomous()`, and `get_encoder_ticks()`.

Here is the caller graph for this function:



7.1.4.40 imeGetVelocity()

```
bool imeGetVelocity (
    unsigned char address,
    int * value )
```

Gets the current rotational velocity of the specified IME.

In this version of PROS, the velocity is positive if the IME count is increasing and negative if the IME count is decreasing. The velocity is in RPM of the internal encoder wheel. Since checking the IME for its type cannot reveal whether the motor gearing is high speed or high torque (in the 2-Wire Motor 393 case), the user must divide the return value by the number of output revolutions per encoder revolution:

- 30.056 for the 269 IME
- 39.2 for the 393 IME in high torque mode (factory default)
- 24.5 for the 393 IME in high speed mode

If the IME address is invalid, or the IME has not been reset or initialized, the value stored in `*value` is undefined.

Parameters

<code>address</code>	the IME address to fetch from 0 to IME_ADDR_MAX
<code>value</code>	a pointer to the location where the value will be stored (obtained using the "&" operator on the target variable name e.g. <code>imeGetVelocity(2, &counts)</code>)

Returns

true if the velocity was successfully read and the value stored in `*value` is valid; false otherwise

Referenced by `get_encoder_velocity()`.

Here is the caller graph for this function:



7.1.4.41 imelInitializeAll()

```
unsigned int imelInitializeAll ( )
```

Initializes all IMEs.

IMEs are assigned sequential incrementing addresses, beginning with the first IME on the chain (closest to the VEX Cortex I2C port). Therefore, a given configuration of IMEs will always have the same ID assigned to each encoder. The addresses range from 0 to IME_ADDR_MAX, so the first encoder gets 0, the second gets 1, ...

This function should most likely be used in [initialize\(\)](#). Do not use it in [initializeIO\(\)](#) or at any other time when the scheduler is paused (like an interrupt). Checking the return value of this function is important to ensure that all IMEs are plugged in and responding as expected.

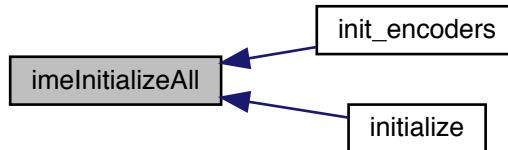
This function, unlike the other IME functions, is not thread safe. If using `imeInitializeAll` to re-initialize encoders, calls to other IME functions might behave unpredictably during this function's execution.

Returns

the number of IMEs successfully initialized.

Referenced by `init_encoders()`, and `initialize()`.

Here is the caller graph for this function:



7.1.4.42 `imeReset()`

```
bool imeReset (
    unsigned char address )
```

Resets the specified IME's counters to zero.

This method can be used while the IME is rotating.

Parameters

<code>address</code>	the IME address to reset from 0 to IME_ADDR_MAX
----------------------	---

Returns

true if the reset succeeded; false otherwise

Referenced by autonomous().

Here is the caller graph for this function:



7.1.4.43 imeShutdown()

```
void imeShutdown ( )
```

Shuts down all IMEs on the chain; their addresses return to the default and the stored counts and velocities are lost. This function, unlike the other IME functions, is not thread safe.

To use the IME chain again, wait at least 0.25 seconds before using imeInitializeAll again.

7.1.4.44 ioClearInterrupt()

```
void ioClearInterrupt (
    unsigned char pin )
```

Disables interrupts on the specified pin.

Disabling interrupts on interrupt pins which are not in use conserves processing time.

Parameters

<i>pin</i>	the pin on which to reset interrupts from 1-9,11-12
------------	---

7.1.4.45 ioSetInterrupt()

```
void ioSetInterrupt (
    unsigned char pin,
    unsigned char edges,
    InterruptHandler handler )
```

Sets up an interrupt to occur on the specified pin, and resets any counters or timers associated with the pin.

Each time the specified change occurs, the function pointer passed in will be called with the pin that changed as an argument. Enabling pin-change interrupts consumes processing time, so it is best to only enable necessary interrupts and to keep the InterruptHandler function short. Pin change interrupts can only be enabled on pins 1-9 and 11-12.

Do not use API functions such as [delay\(\)](#) inside the handler function, as the function will run in an ISR where the scheduler is paused and no other interrupts can execute. It is best to quickly update some state and allow a task to perform the work.

Do not use this function on pins that are also being used by the built-in ultrasonic or shaft encoder drivers, or on pins which have been switched to output mode.

Parameters

<i>pin</i>	the pin on which to enable interrupts from 1-9,11-12
<i>edges</i>	one of INTERRUPT_EDGE_RISING, INTERRUPT_EDGE_FALLING, or INTERRUPT_EDGE_BOTH
<i>handler</i>	the function to call when the condition is satisfied

7.1.4.46 isAutonomous()

```
bool isAutonomous( )
```

Returns true if the robot is in autonomous mode, or false otherwise.

While in autonomous mode, joystick inputs will return a neutral value, but serial port communications (even over VexNET) will still work properly.

7.1.4.47 isEnabled()

```
bool isEnabled( )
```

Returns true if the robot is enabled, or false otherwise.

While disabled via the VEX Competition Switch or VEX Field Controller, motors will not function. However, the digital I/O ports can still be changed, which may indirectly affect the robot state (e.g. solenoids). Avoid performing externally visible actions while disabled (the kernel should take care of this most of the time).

Referenced by [display_menu\(\)](#).

Here is the caller graph for this function:



7.1.4.48 isJoystickConnected()

```
bool isJoystickConnected (
    unsigned char joystick )
```

Returns true if a joystick is connected to the specified slot number (1 or 2), or false otherwise.

Useful for automatically merging joysticks for one operator, or splitting for two. This function does not work properly during [initialize\(\)](#) or [initializeIO\(\)](#) and can return false positives. It should be checked once and stored at the beginning of [operatorControl\(\)](#).

Parameters

<i>joystick</i>	the joystick slot to check
-----------------	----------------------------

7.1.4.49 isOnline()

```
bool isOnline ( )
```

Returns true if a VEX field controller or competition switch is connected, or false otherwise.

When in online mode, the switching between [autonomous\(\)](#) and [operatorControl\(\)](#) tasks is managed by the PROS kernel.

7.1.4.50 joystickGetAnalog()

```
int joystickGetAnalog (
    unsigned char joystick,
    unsigned char axis )
```

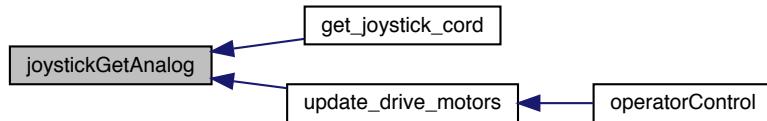
Gets the value of a control axis on the VEX joystick. Returns the value from -127 to 127, or 0 if no joystick is connected to the requested slot.

Parameters

<i>joystick</i>	the joystick slot to check
<i>axis</i>	one of 1, 2, 3, 4, ACCEL_X, or ACCEL_Y

Referenced by [get_joystick_cord\(\)](#), and [update_drive_motors\(\)](#).

Here is the caller graph for this function:



7.1.4.51 joystickGetDigital()

```

bool joystickGetDigital (
    unsigned char joystick,
    unsigned char buttonGroup,
    unsigned char button )
  
```

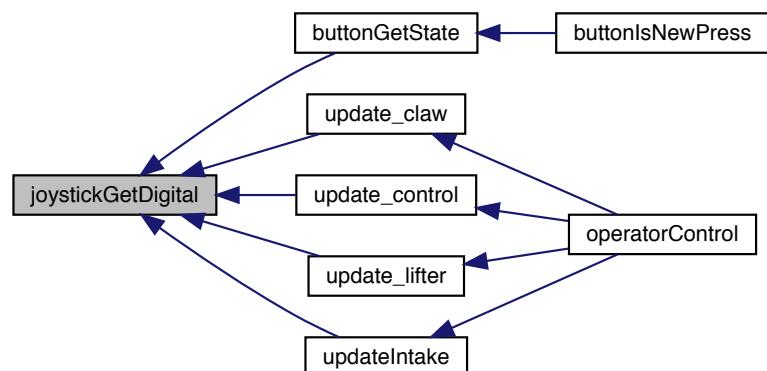
Gets the value of a button on the VEX joystick. Returns true if that button is pressed, or false otherwise. If no joystick is connected to the requested slot, returns false.

Parameters

<i>joystick</i>	the joystick slot to check
<i>buttonGroup</i>	one of 5, 6, 7, or 8 to request that button as labelled on the joystick
<i>button</i>	one of JOY_UP, JOY_DOWN, JOY_LEFT, or JOY_RIGHT; requesting JOY_LEFT or JOY_RIGHT for groups 5 or 6 will cause an undefined value to be returned

Referenced by buttonGetState(), update_claw(), update_control(), update_lifter(), and updateIntake().

Here is the caller graph for this function:



7.1.4.52 lcdClear()

```
void lcdClear (
    PROS_FILE * lcdPort )
```

Clears the LCD screen on the specified port.

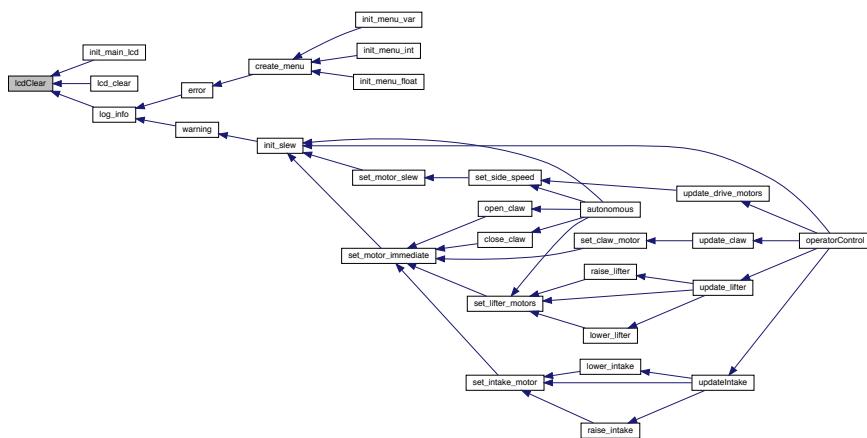
Printing to a line implicitly overwrites the contents, so clearing should only be required at startup.

Parameters

<i>lcdPort</i>	the LCD to clear, either uart1 or uart2
----------------	---

Referenced by init_main_lcd(), lcd_clear(), and log_info().

Here is the caller graph for this function:



7.1.4.53 lcdInit()

```
void lcdInit (
    PROS_FILE * lcdPort )
```

Initializes the LCD port, but does not change the text or settings.

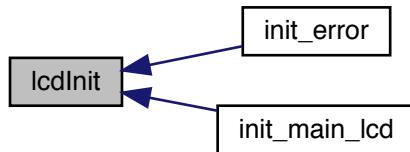
If the LCD was not initialized before, the text currently on the screen will be undefined. The port will not be usable with standard serial port functions until the LCD is stopped.

Parameters

<i>lcdPort</i>	the LCD to initialize, either uart1 or uart2
----------------	--

Referenced by init_error(), and init_main_lcd().

Here is the caller graph for this function:



7.1.4.54 lcdReadButtons()

```
void unsigned char const char unsigned int lcdReadButtons (
    PROS_FILE * lcdPort )
```

Reads the user button status from the LCD display.

For example, if the left and right buttons are pushed, $(1 | 4) = 5$ will be returned. 0 is returned if no buttons are pushed.

Parameters

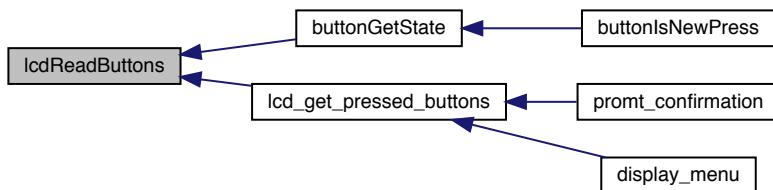
<i>lcdPort</i>	the LCD to poll, either uart1 or uart2
----------------	--

Returns

the buttons pressed as a bit mask

Referenced by buttonGetState(), and lcd_get_pressed_buttons().

Here is the caller graph for this function:



7.1.4.55 lcdSetBacklight()

```
void lcdSetBacklight (
    PROS_FILE * lcdPort,
    bool backlight )
```

Sets the specified LCD backlight to be on or off.

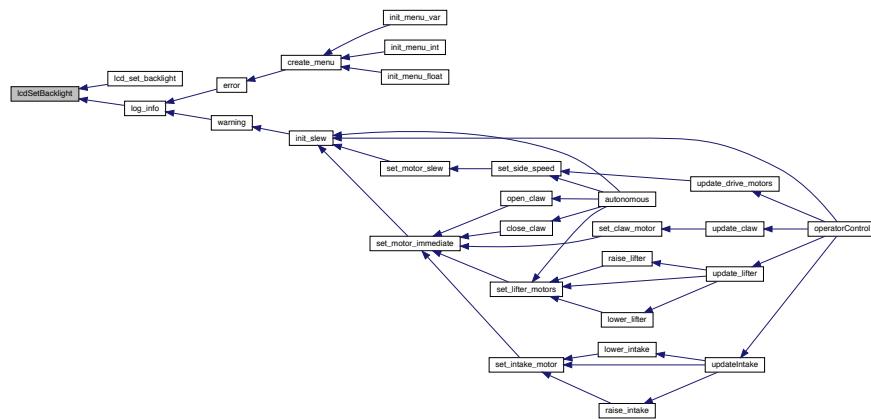
Turning it off will save power but may make it more difficult to read in dim conditions.

Parameters

<i>lcdPort</i>	the LCD to adjust, either uart1 or uart2
<i>backlight</i>	true to turn the backlight on, or false to turn it off

Referenced by `lcd_set_backlight()`, and `log_info()`.

Here is the caller graph for this function:



7.1.4.56 lcdSetText()

```
void lcdSetText (
    PROS_FILE * lcdPort,
    unsigned char line,
    const char * buffer )
```

Prints the string buffer to the attached LCD.

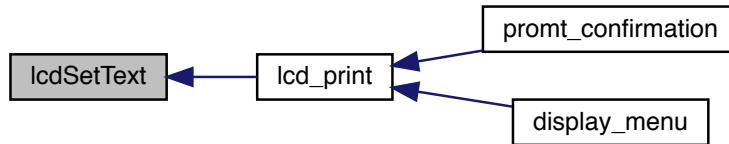
The output string will be truncated as necessary to fit on the LCD screen, 16 characters wide. This function, like `fprint()`, is much, much faster than a formatted routine such as `LcdPrint()` and consumes less memory.

Parameters

<i>lcdPort</i>	the LCD to write, either uart1 or uart2
<i>line</i>	the LCD line to write, either 1 or 2
<i>buffer</i> Generated by Doxygen	the string to write

Referenced by lcd_print().

Here is the caller graph for this function:



7.1.4.57 lcdShutdown()

```
void lcdShutdown (
    PROS_FILE * lcdPort )
```

Shut down the specified LCD port.

Parameters

<code>lcdPort</code>	the LCD to stop, either uart1 or uart2
----------------------	--

7.1.4.58 micros()

```
unsigned long micros ( )
```

Returns the number of microseconds since Cortex power-up. There are 10^6 microseconds in a second, so as a 32-bit integer, this will overflow and wrap back to zero every two hours or so.

This function is Wiring-compatible.

Returns

the number of microseconds since the Cortex was turned on or the last overflow

7.1.4.59 millis()

```
unsigned long millis ( )
```

Returns the number of milliseconds since Cortex power-up. There are 1000 milliseconds in a second, so as a 32-bit integer, this will not overflow for 50 days.

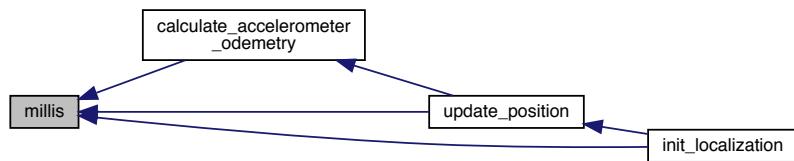
This function is Wiring-compatible.

Returns

the number of milliseconds since the Cortex was turned on

Referenced by calculate_accelerometer_odometry(), init_localization(), and update_position().

Here is the caller graph for this function:



7.1.4.60 motorGet()

```
int motorGet (
    unsigned char channel )
```

Gets the last set speed of the specified motor channel.

This speed may have been set by any task or the PROS kernel itself. This is not guaranteed to be the speed that the motor is actually running at, or even the speed currently being sent to the motor, due to latency in the Motor Controller 29 protocol and physical loading. To measure actual motor shaft revolution speed, attach a VEX Integrated Motor Encoder or VEX Quadrature Encoder and use the velocity functions associated with each.

Parameters

<i>channel</i>	the motor channel to fetch from 1-10
----------------	--------------------------------------

Returns

the speed last sent to this channel; -127 is full reverse and 127 is full forward, with 0 being off

7.1.4.61 motorSet()

```
void motorSet (
    unsigned char channel,
    int speed )
```

Sets the speed of the specified motor channel.

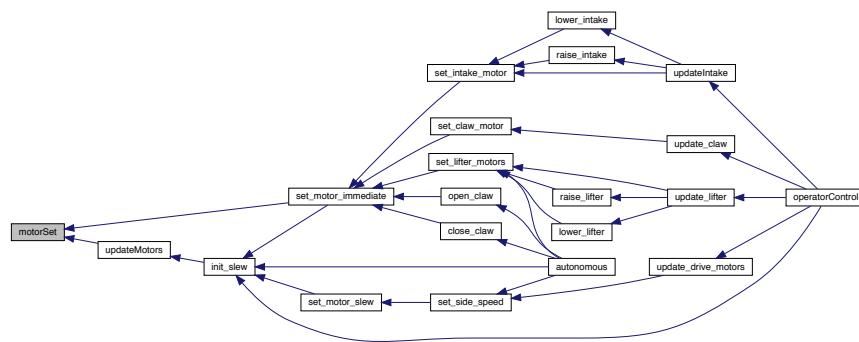
Do not use [motorSet\(\)](#) with the same channel argument from two different tasks. It is safe to use [motorSet\(\)](#) with different channel arguments from different tasks.

Parameters

<i>channel</i>	the motor channel to modify from 1-10
<i>speed</i>	the new signed speed; -127 is full reverse and 127 is full forward, with 0 being off

Referenced by [set_motor_immediate\(\)](#), and [updateMotors\(\)](#).

Here is the caller graph for this function:



7.1.4.62 motorStop()

```
void motorStop (
    unsigned char channel )
```

Stops the motor on the specified channel, equivalent to calling [motorSet\(\)](#) with an argument of zero.

This performs a coasting stop, not an active brake. Since [motorStop](#) is similar to [motorSet\(0\)](#), see the note for [motorSet\(\)](#) about use from multiple tasks.

Parameters

<i>channel</i>	the motor channel to stop from 1-10
----------------	-------------------------------------

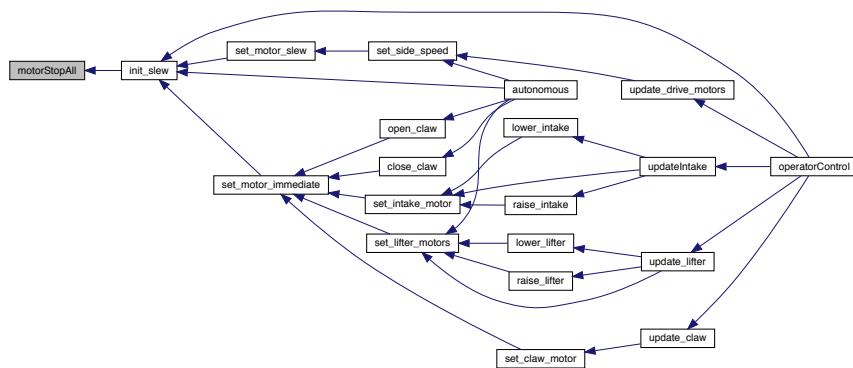
7.1.4.63 motorStopAll()

```
void motorStopAll ( )
```

Stops all motors; significantly faster than looping through all motor ports and calling motorSet(channel, 0) on each one.

Referenced by init_slew().

Here is the caller graph for this function:



7.1.4.64 mutexCreate()

```
Mutex mutexCreate ( )
```

Creates a mutex intended to allow only one task to use a resource at a time. For signalling and synchronization, try using semaphores.

Mutexes created using this function can be accessed using the [mutexTake\(\)](#) and [mutexGive\(\)](#) functions. The semaphore functions must not be used on objects of this type.

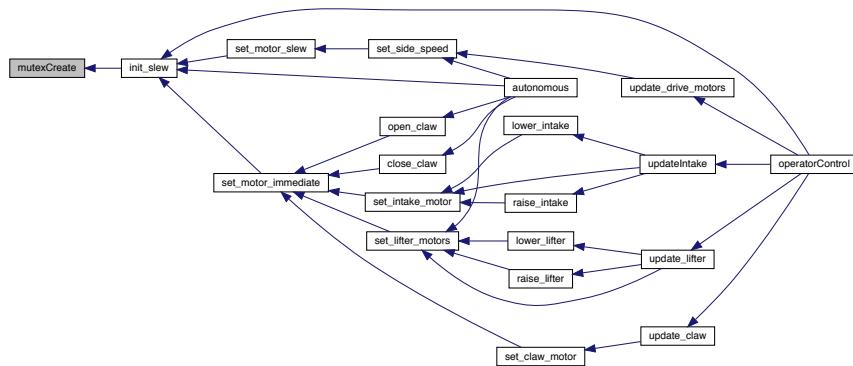
This type of object uses a priority inheritance mechanism so a task 'taking' a mutex MUST ALWAYS 'give' the mutex back once the mutex is no longer required.

Returns

a handle to the created mutex

Referenced by init_slew().

Here is the caller graph for this function:



7.1.4.65 mutexDelete()

```
void mutexDelete (
    Mutex mutex )
```

Deletes the specified mutex. This function can be dangerous; deleting semaphores being waited on by a task may cause deadlock or a crash.

Parameters

<code>mutex</code>	the mutex to destroy
--------------------	----------------------

7.1.4.66 mutexGive()

```
bool mutexGive (
    Mutex mutex )
```

Relinquishes a mutex so that other tasks can use the resource it guards. The mutex must be held by the current task using a corresponding call to `mutexTake`.

Parameters

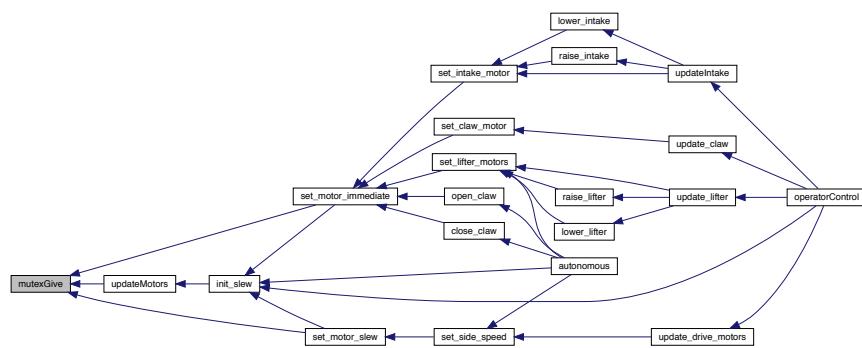
<code>mutex</code>	the mutex to release
--------------------	----------------------

Returns

true if the mutex was released, or false if the mutex was not already held

Referenced by `set_motor_immediate()`, `set_motor_slew()`, and `updateMotors()`.

Here is the caller graph for this function:



7.1.4.67 mutexTake()

```
bool mutexTake (
    Mutex mutex,
    const unsigned long blockTime )
```

Requests a mutex so that other tasks cannot simultaneously use the resource it guards. The mutex must not already be held by the current task. If another task already holds the mutex, the function will wait for the mutex to be released. Other tasks can run during this time.

Parameters

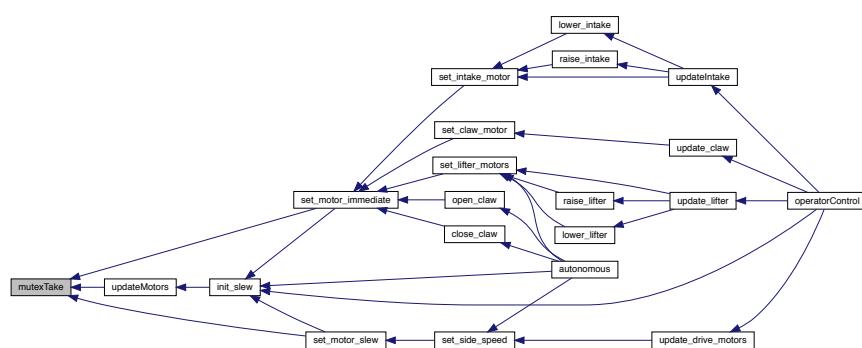
<i>mutex</i>	the mutex to request
<i>blockTime</i>	the maximum time to wait for the mutex to be available, where -1 specifies an infinite timeout

Returns

true if the mutex was successfully taken, or false if the timeout expired

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

Here is the caller graph for this function:



7.1.4.68 pinMode()

```
void pinMode (
    unsigned char pin,
    unsigned char mode )
```

Configures the pin as an input or output with a variety of settings.

Do note that INPUT by default turns on the pull-up resistor, as most VEX sensors are open-drain active low. It should not be a big deal for most push-pull sources. This function is Wiring-compatible.

Parameters

<i>pin</i>	the pin to modify from 1-26
<i>mode</i>	one of INPUT, INPUT_ANALOG, INPUT_FLOATING, OUTPUT, or OUTPUT_OD

7.1.4.69 powerLevelBackup()

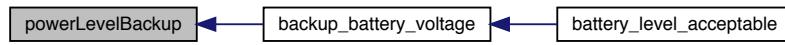
```
unsigned int powerLevelBackup ( )
```

Returns the backup battery voltage in millivolts.

If no backup battery is connected, returns 0.

Referenced by `backup_battery_voltage()`.

Here is the caller graph for this function:



7.1.4.70 powerLevelMain()

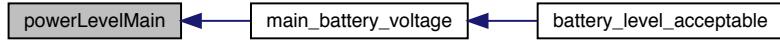
```
unsigned int powerLevelMain ( )
```

Returns the main battery voltage in millivolts.

In rare circumstances, this method might return 0. Check the output value for reasonability before blindly blasting the user.

Referenced by main_battery_voltage().

Here is the caller graph for this function:



7.1.4.71 print()

```
void print (
    const char * string )
```

Prints the simple string to the debug terminal without formatting.

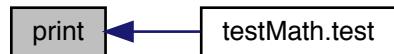
This method is much, much faster than [printf\(\)](#).

Parameters

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

Referenced by testMath::test().

Here is the caller graph for this function:



7.1.4.72 printf()

```
int printf (
    const char * formatString,
    ... )
```

Prints the formatted string to the debug stream (the PC terminal).

Parameters

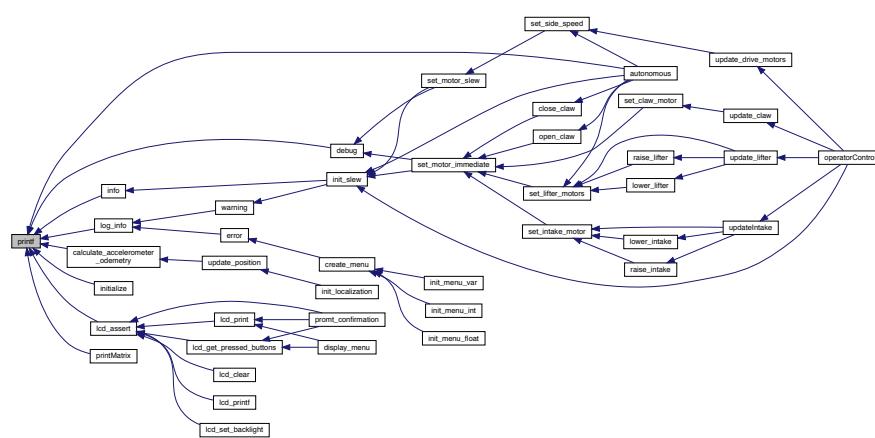
<i>formatString</i>	the format string as specified in fprintf()
---------------------	---

Returns

the number of characters written

Referenced by autonomous(), calculate_accelerometer_odometry(), debug(), info(), initialize(), lcd_assert(), log_← info(), and printMatrix().

Here is the caller graph for this function:

**7.1.4.73 putchar()**

```
int putchar (
    int value )
```

Writes one character to "stdout", which is the PC debug terminal, and returns the input value.

When using a wireless connection, one may need to press the spacebar before the input is visible on the terminal.

Parameters

<i>value</i>	the character to write (a value of type "char" can be used)
--------------	---

Returns

the character written

7.1.4.74 puts()

```
int puts (
    const char * string )
```

Behaves the same as the "print" function, and appends a trailing newline ("\n").

Parameters

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

Returns

the number of characters written, excluding the new line

7.1.4.75 semaphoreCreate()

```
Semaphore semaphoreCreate ( )
```

Creates a semaphore intended for synchronizing tasks. To prevent some critical code from simultaneously modifying a shared resource, use mutexes instead.

Semaphores created using this function can be accessed using the [semaphoreTake\(\)](#) and [semaphoreGive\(\)](#) functions. The mutex functions must not be used on objects of this type.

This type of object does not need to have balanced take and give calls, so priority inheritance is not used. Semaphores can be signalled by an interrupt routine.

Returns

a handle to the created semaphore

7.1.4.76 semaphoreDelete()

```
void semaphoreDelete (
    Semaphore semaphore )
```

Deletes the specified semaphore. This function can be dangerous; deleting semaphores being waited on by a task may cause deadlock or a crash.

Parameters

<i>semaphore</i>	the semaphore to destroy
------------------	--------------------------

7.1.4.77 semaphoreGive()

```
bool semaphoreGive (
    Semaphore semaphore )
```

Signals a semaphore. Tasks waiting for a signal using [semaphoreTake\(\)](#) will be unblocked by this call and can continue execution.

Slow processes can give semaphores when ready, and fast processes waiting to take the semaphore will continue at that point.

Parameters

<i>semaphore</i>	the semaphore to signal
------------------	-------------------------

Returns

true if the semaphore was successfully given, or false if the semaphore was not taken since the last give

7.1.4.78 semaphoreTake()

```
bool semaphoreTake (
    Semaphore semaphore,
    const unsigned long blockTime )
```

Waits on a semaphore. If the semaphore is already in the "taken" state, the current task will wait for the semaphore to be signaled. Other tasks can run during this time.

Parameters

<i>semaphore</i>	the semaphore to wait
<i>blockTime</i>	the maximum time to wait for the semaphore to be given, where -1 specifies an infinite timeout

Returns

true if the semaphore was successfully taken, or false if the timeout expired

7.1.4.79 setTeamName()

```
void setTeamName (
    const char * name )
```

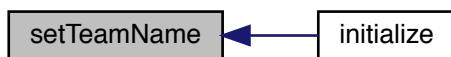
Sets the team name displayed to the VEX field control and VEX Firmware Upgrade.

Parameters

<i>name</i>	a string containing the team name; only the first eight characters will be shown
-------------	--

Referenced by initialize().

Here is the caller graph for this function:



7.1.4.80 snprintf()

```
int snprintf (
    char * buffer,
    size_t limit,
    const char * formatString,
    ...
)
```

Prints the formatted string to the string buffer with the specified length limit.

The length limit, as per the C standard, includes the trailing null character, so an argument of 256 will cause a maximum of 255 non-null characters to be printed, and one null terminator in all cases.

Parameters

<i>buffer</i>	the string buffer where characters can be placed
<i>limit</i>	the maximum number of characters to write
<i>formatString</i>	the format string as specified in fprintf()

Returns

the number of characters stored

7.1.4.81 speakerInit()

```
void speakerInit ( )
```

Initializes VEX speaker support.

The VEX speaker is not thread safe; it can only be used from one task at a time. Using the VEX speaker may impact robot performance. Teams may benefit from an if statement that only enables sound if [isOnline\(\)](#) returns false.

7.1.4.82 speakerPlayArray()

```
void speakerPlayArray (
    const char ** songs )
```

Plays up to three RTTTL (Ring Tone Text Transfer Language) songs simultaneously over the VEX speaker. The audio is mixed to allow polyphonic sound to be played. Many simple songs are available in RTTTL format online, or compose your own.

The song must not be NULL, but unused tracks within the song can be set to NULL. If any of the three song tracks is invalid, the result of this function is undefined.

The VEX speaker is not thread safe; it can only be used from one task at a time. Using the VEX speaker may impact robot performance. Teams may benefit from an if statement that only enables sound if [isOnline\(\)](#) returns false.

Parameters

<i>songs</i>	an array of up to three (3) RTTTL songs as string values to play
--------------	--

7.1.4.83 speakerPlayRtttl()

```
void speakerPlayRtttl (
    const char * song )
```

Plays an RTTTL (Ring Tone Text Transfer Language) song over the VEX speaker. Many simple songs are available in RTTTL format online, or compose your own.

The song must not be NULL. If an invalid song is specified, the result of this function is undefined.

The VEX speaker is not thread safe; it can only be used from one task at a time. Using the VEX speaker may impact robot performance. Teams may benefit from an if statement that only enables sound if [isOnline\(\)](#) returns false.

Parameters

<i>song</i>	the RTTTL song as a string value to play
-------------	--

7.1.4.84 speakerShutdown()

```
void speakerShutdown ( )
```

Powers down and disables the VEX speaker.

If a song is currently being played in another task, the behavior of this function is undefined, since the VEX speaker is not thread safe.

7.1.4.85 sprintf()

```
int sprintf (
    char * buffer,
    const char * formatString,
    ...
)
```

Prints the formatted string to the string buffer.

If the buffer is not big enough to contain the complete formatted output, undefined behavior occurs. See [snprintf\(\)](#) for a safer version of this function.

Parameters

<i>buffer</i>	the string buffer where characters can be placed
<i>formatString</i>	the format string as specified in fprintf()

Returns

the number of characters stored

7.1.4.86 standaloneModeEnable()

```
void standaloneModeEnable ( )
```

Enables the Cortex to run the op control task in a standalone mode- no VEXnet connection required.

This function should only be called once in [initializeIO\(\)](#)

7.1.4.87 taskCreate()

```
TaskHandle taskCreate (
    TaskCode taskCode,
    const unsigned int stackDepth,
    void * parameters,
    const unsigned int priority )
```

Creates a new task and add it to the list of tasks that are ready to run.

Parameters

<i>taskCode</i>	the function to execute in its own task
<i>stackDepth</i>	the number of variables available on the stack (4 * stackDepth bytes will be allocated on the Cortex)
<i>parameters</i>	an argument passed to the taskCode function
<i>priority</i>	a value from TASK_PRIORITY_LOWEST to TASK_PRIORITY_HIGHEST determining the initial priority of the task

Returns

a handle to the created task, or NULL if an error occurred

7.1.4.88 taskDelay()

```
void taskDelay (
    const unsigned long msToDelay )
```

Delays the current task for a given number of milliseconds.

Delaying for a period of zero will force a reschedule, where tasks of equal priority may be scheduled if available. The calling task will still be available for immediate rescheduling once the other tasks have had their turn or if nothing of equal or higher priority is available to be scheduled.

This is not the best method to have a task execute code at predefined intervals, as the delay time is measured from when the delay is requested. To delay cyclically, use [taskDelayUntil\(\)](#).

Parameters

<i>msToDelay</i>	the number of milliseconds to wait, with 1000 milliseconds per second
------------------	---

7.1.4.89 taskDelayUntil()

```
void taskDelayUntil (
    unsigned long * previousWakeTime,
    const unsigned long cycleTime )
```

Delays the current task until a specified time. The task will be unblocked at the time *previousWakeTime + cycleTime, and *previousWakeTime will be changed to reflect the time at which the task will unblock.

If the target time is in the past, no delay occurs, but a reschedule is forced, as if [taskDelay\(\)](#) was called with an argument of zero. If the sum of cycleTime and *previousWakeTime overflows or underflows, undefined behavior occurs.

This function should be used by cyclical tasks to ensure a constant execution frequency. While [taskDelay\(\)](#) specifies a wake time relative to the time at which the function is called, [taskDelayUntil\(\)](#) specifies the absolute future time at which it wishes to unblock. Calling taskDelayUntil with the same cycleTime parameter value in a loop, with previousWakeTime referring to a local variable initialized to [millis\(\)](#), will cause the loop to execute with a fixed period.

Parameters

<i>previousWakeTime</i>	a pointer to the location storing the last unblock time, obtained by using the "&" operator on a variable (e.g. "taskDelayUntil(&now, 50);")
<i>cycleTime</i>	the number of milliseconds to wait, with 1000 milliseconds per second

7.1.4.90 taskDelete()

```
void taskDelete (
    TaskHandle taskToDelete )
```

Kills and removes the specified task from the kernel task list.

Deleting the last task will end the program, possibly leading to undesirable states as some outputs may remain in their last set configuration.

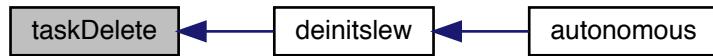
NOTE: The idle task is responsible for freeing the kernel allocated memory from tasks that have been deleted. It is therefore important that the idle task is not starved of processing time. Memory allocated by the task code is not automatically freed, and should be freed before the task is deleted.

Parameters

<i>taskToDelete</i>	the task to kill; passing NULL kills the current task
---------------------	---

Referenced by deinitstlew().

Here is the caller graph for this function:



7.1.4.91 taskGetCount()

```
unsigned int taskGetCount ( )
```

Determines the number of tasks that are currently being managed.

This includes all ready, blocked and suspended tasks. A task that has been deleted but not yet freed by the idle task will also be included in the count. Tasks recently created may take one context switch to be counted.

Returns

the number of tasks that are currently running, waiting, or suspended

7.1.4.92 taskGetState()

```
unsigned int taskGetState (
    TaskHandle task )
```

Retrieves the state of the specified task. Note that the state of tasks which have died may be re-used for future tasks, causing the value returned by this function to reflect a different task than possibly intended in this case.

Parameters

<i>task</i>	Handle to the task to query. Passing NULL will query the current task status (which will, by definition, be TASK_RUNNING if this call returns)
-------------	--

Returns

A value reflecting the task's status, one of the constants TASK_DEAD, TASK_RUNNING, TASK_RUNNABLE, TASK_SLEEPING, or TASK_SUSPENDED

7.1.4.93 taskPriorityGet()

```
unsigned int taskPriorityGet (
    const TaskHandle task )
```

Obtains the priority of the specified task.

Parameters

<i>task</i>	the task to check; passing NULL checks the current task
-------------	---

Returns

the priority of that task from 0 to TASK_MAX_PRIORITIES

7.1.4.94 taskPrioritySet()

```
void taskPrioritySet (
    TaskHandle task,
    const unsigned int newPriority )
```

Sets the priority of the specified task.

A context switch may occur before the function returns if the priority being set is higher than the currently executing task and the task being mutated is available to be scheduled.

Parameters

<i>task</i>	the task to change; passing NULL changes the current task
<i>newPriority</i>	a value between TASK_PRIORITY_LOWEST and TASK_PRIORITY_HIGHEST inclusive indicating the new task priority

7.1.4.95 taskResume()

```
void taskResume (
    TaskHandle taskToResume )
```

Resumes the specified task.

A task that has been suspended by one or more calls to [taskSuspend\(\)](#) will be made available for scheduling again by a call to [taskResume\(\)](#). If the task was not suspended at the time of the call to [taskResume\(\)](#), undefined behavior occurs.

Parameters

<i>taskToResume</i>	the task to change; passing NULL is not allowed as the current task cannot be suspended (it is obviously running if this function is called)
---------------------	--

7.1.4.96 taskRunLoop()

```
TaskHandle taskRunLoop (
    void(*)(void) fn,
    const unsigned long increment )
```

Starts a task which will periodically call the specified function.

Intended for use as a quick-start skeleton for cyclic tasks with higher priority than the "main" tasks. The created task will have priority TASK_PRIORITY_DEFAULT + 1 with the default stack size. To customize behavior, create a task manually with the specified function.

This task will automatically terminate after one further function invocation when the robot is disabled or when the robot mode is switched.

Parameters

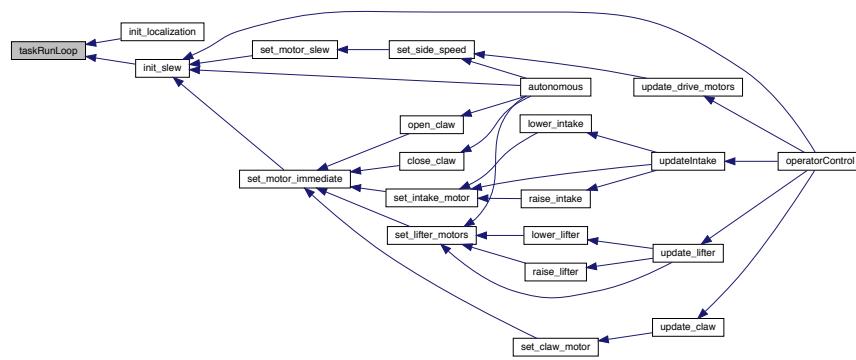
<i>fn</i>	the function to call in this loop
<i>increment</i>	the delay between successive calls in milliseconds; the taskDelayUntil() function is used for accurate cycle timing

Returns

a handle to the task, or NULL if an error occurred

Referenced by init_localization(), and init_slew().

Here is the caller graph for this function:

**7.1.4.97 taskSuspend()**

```
void taskSuspend (
    TaskHandle taskToSuspend )
```

Suspends the specified task.

When suspended a task will not be scheduled, regardless of whether it might be otherwise available to run.

Parameters

<i>taskToSuspend</i>	the task to suspend; passing NULL suspends the current task
----------------------	---

7.1.4.98 ultrasonicGet()

```
int ultrasonicGet (
    Ultrasonic ult )
```

Gets the current ultrasonic sensor value in centimeters.

If no object was found or if the ultrasonic sensor is polled while it is pinging and waiting for a response, -1 (ULTR \leftarrow A_BAD_RESPONSE) is returned. If the ultrasonic sensor was never started, the return value is undefined. Round and fluffy objects can cause inaccurate values to be returned.

Parameters

<i>ult</i>	the Ultrasonic object from ultrasonicInit() to read
------------	---

Returns

the distance to the nearest object in centimeters

7.1.4.99 ultrasonicInit()

```
Ultrasonic ultrasonicInit (
    unsigned char portEcho,
    unsigned char portPing )
```

Initializes an ultrasonic sensor on the specified digital ports.

The ultrasonic sensor will be polled in the background in concert with the other sensors registered using this method. NULL will be returned if either port is invalid or the ultrasonic sensor port is already in use.

Parameters

<i>portEcho</i>	the port connected to the orange cable from 1-9,11-12
<i>portPing</i>	the port connected to the yellow cable from 1-12

Returns

an Ultrasonic object to be stored and used for later calls to ultrasonic functions

7.1.4.100 ultrasonicShutdown()

```
void ultrasonicShutdown (
    Ultrasonic ult )
```

Stops and disables the ultrasonic sensor.

The last distance it had before stopping will be retained. One more ping operation may occur before the sensor is fully disabled.

Parameters

<i>ult</i>	the Ultrasonic object from ultrasonicInit() to stop
------------	---

7.1.4.101 usartInit()

```
void usartInit (
    PROS_FILE * usart,
    unsigned int baud,
    unsigned int flags )
```

Initialize the specified serial interface with the given connection parameters.

I/O to the port is accomplished using the "standard" I/O functions such as [fputs\(\)](#), [fprintf\(\)](#), and [fputc\(\)](#).

Re-initializing an open port may cause loss of data in the buffers. This routine may be safely called from [initializeIO\(\)](#) or when the scheduler is paused. If I/O is attempted on a serial port which has never been opened, the behavior will be the same as if the port had been disabled.

Parameters

<i>usart</i>	the port to open, either "uart1" or "uart2"
<i>baud</i>	the baud rate to use from 2400 to 1000000 baud
<i>flags</i>	a bit mask combination of the SERIAL_* flags specifying parity, stop, and data bits

7.1.4.102 usartShutdown()

```
void usartShutdown (
    PROS_FILE * usart )
```

Disables the specified USART interface.

Any data in the transmit and receive buffers will be lost. Attempts to read from the port when it is disabled will deadlock, and attempts to write to it may deadlock depending on the state of the buffer.

Parameters

<i>usart</i>	the port to close, either "uart1" or "uart2"
--------------	--

7.1.4.103 wait()

```
void wait (
    const unsigned long time )
```

Alias of [taskDelay\(\)](#) intended to help EasyC users.

Parameters

<i>time</i>	the duration of the delay in milliseconds (1 000 milliseconds per second)
-------------	---

7.1.4.104 waitUntil()

```
void waitUntil (
    unsigned long * previousWakeTime,
    const unsigned long time )
```

Alias of [taskDelayUntil\(\)](#) intended to help EasyC users.

Parameters

<i>previousWakeTime</i>	a pointer to the last wakeup time
<i>time</i>	the duration of the delay in milliseconds (1 000 milliseconds per second)

7.1.4.105 watchdogInit()

```
void watchdogInit ( )
```

Enables IWDG watchdog timer which will reset the cortex if it locks up due to static shock or a misbehaving task preventing the timer to be reset. Not recovering from static shock will cause the robot to continue moving its motors indefinitely until turned off manually.

This function should only be called once in [initializeIO\(\)](#)

Referenced by [initializeIO\(\)](#).

Here is the caller graph for this function:



7.1.5 Variable Documentation

7.1.5.1 formatString

```
void unsigned char const char* formatString
```

Definition at line 1182 of file API.h.

7.1.5.2 line

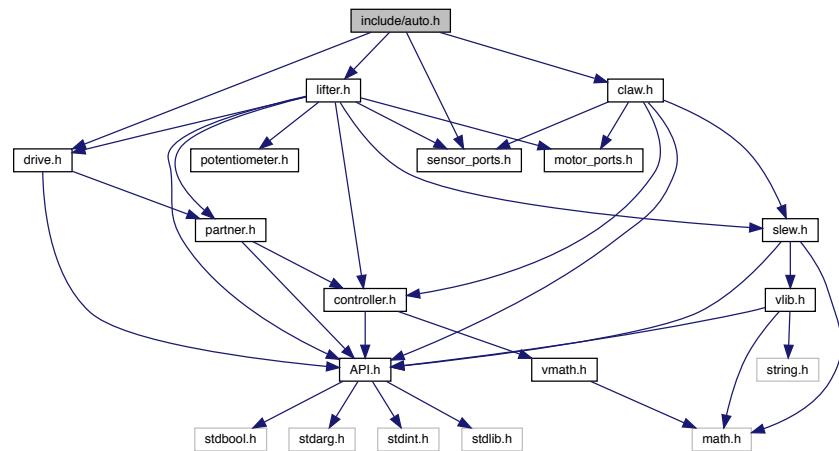
```
void unsigned char line
```

Definition at line 1182 of file API.h.

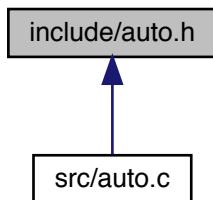
7.2 include/auto.h File Reference

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

Include dependency graph for auto.h:



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



Macros

- #define FRONT_LEFT_DRIVE 0
- #define GOAL_HEIGHT 1325
- #define MID_LEFT_DRIVE 1
- #define MID_RIGHT_DRIVE 4
- #define STOP_ONE 500
- #define STOP_TWO 700

7.2.1 Macro Definition Documentation

7.2.1.1 FRONT_LEFT_DRIVE

```
#define FRONT_LEFT_DRIVE 0
```

Definition at line 9 of file auto.h.

7.2.1.2 GOAL_HEIGHT

```
#define GOAL_HEIGHT 1325
```

Definition at line 14 of file auto.h.

Referenced by autonomous().

7.2.1.3 MID_LEFT_DRIVE

```
#define MID_LEFT_DRIVE 1
```

Definition at line 10 of file auto.h.

Referenced by autonomous().

7.2.1.4 MID_RIGHT_DRIVE

```
#define MID_RIGHT_DRIVE 4
```

Definition at line 11 of file auto.h.

Referenced by autonomous().

7.2.1.5 STOP_ONE

```
#define STOP_ONE 500
```

Definition at line 12 of file auto.h.

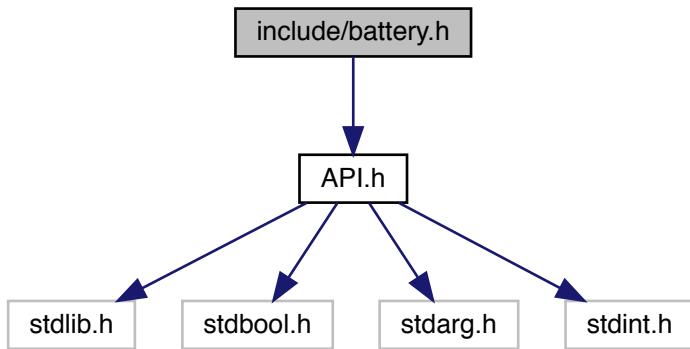
7.2.1.6 STOP_TWO

```
#define STOP_TWO 700
```

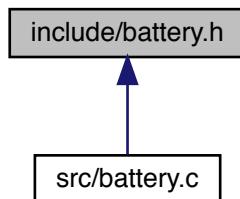
Definition at line 13 of file auto.h.

7.3 include/battery.h File Reference

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



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



Macros

- #define MIN_BACKUP_VOLTAGE 7.8
- #define MIN_MAIN_VOLTAGE 7.8

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 Macro Definition Documentation

7.3.1.1 MIN_BACKUP_VOLTAGE

```
#define MIN_BACKUP_VOLTAGE 7.8
```

The minimum acceptable backup battery voltage before a match.

Definition at line 14 of file battery.h.

Referenced by `battery_level_acceptable()`.

7.3.1.2 MIN_MAIN_VOLTAGE

```
#define MIN_MAIN_VOLTAGE 7.8
```

The minimum acceptable main battery voltage before a match.

Definition at line 9 of file battery.h.

Referenced by `battery_level_acceptable()`.

7.3.2 Function Documentation

7.3.2.1 backup_battery_voltage()

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

Author

Chris Jerrett

Definition at line 8 of file battery.c.

References powerLevelBackup().

Referenced by battery_level_acceptable().

```
8      {
9      return powerLevelBackup() / 1000.0;
10 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.3.2.2 battery_level_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

See also

[MIN_MAIN_VOLTAGE](#)
[MIN_BACKUP_VOLTAGE](#)

Author

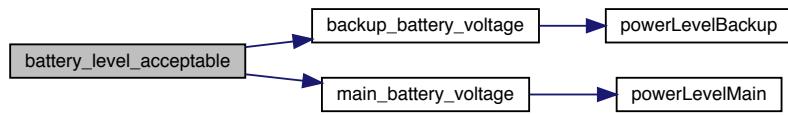
Chris Jerrett

Definition at line 12 of file battery.c.

References `backup_battery_voltage()`, `main_battery_voltage()`, `MIN_BACKUP_VOLTAGE`, and `MIN_MAIN_VOLTAGE`.

```
12  {
13  if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
14  if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
15  return true;
16 }
```

Here is the call graph for this function:



7.3.2.3 main_battery_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

Author

Chris Jerrett

Definition at line 4 of file battery.c.

References powerLevelMain().

Referenced by battery_level_acceptable().

```
4 {  
5     return powerLevelMain() / 1000.0;  
6 }
```

Here is the call graph for this function:



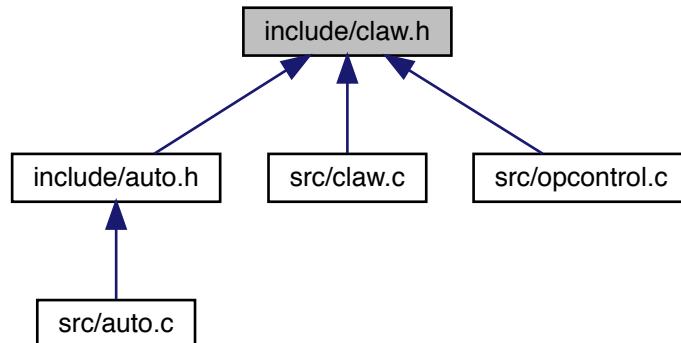
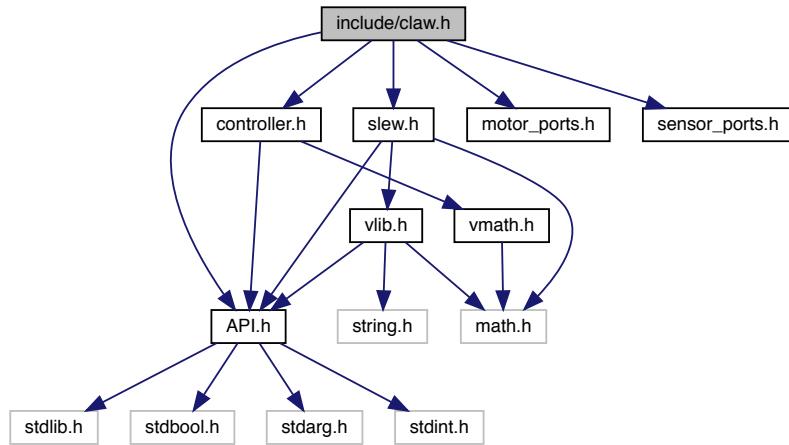
Here is the caller graph for this function:



7.4 include/claw.h File Reference

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

```
#include "sensor_ports.h"
Include dependency graph for claw.h:
```



Macros

- `#define CLAW_CLOSE_MASTER, 6, JOY_UP`
The joystick parameters for closing the claw.
- `#define CLAW_CLOSE_VAL 3000`
The potentiometer value for a closed claw.
- `#define CLAW_D .1`
The derivative constant for the claw PID controller.
- `#define CLAW_I 0`
The integral constant for the claw PID controller.

- `#define CLAW_OPEN_MASTER, 6, JOY_DOWN`
The joystick parameters for opening the claw.
- `#define CLAW_OPEN_VAL 1500`
The potentiometer value for a open claw.
- `#define CLAW_P .1`
The proportional constant for the claw PID controller.
- `#define MAX_CLAW_SPEED 50`
The max motor vlaue of the claw.
- `#define MIN_CLAW_SPEED -50`
The min motor vlaue of the claw.

Enumerations

- enum `claw_state` { `CLAW_OPEN_STATE`, `CLAW_CLOSE_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.

7.4.1 Macro Definition Documentation

7.4.1.1 CLAW_CLOSE

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

The joystick parameters for closing the claw.

Author

Chris Jerrett

Definition at line 41 of file claw.h.

Referenced by update_claw().

7.4.1.2 CLAW_CLOSE_VAL

```
#define CLAW_CLOSE_VAL 3000
```

The potentiometer value for a closed claw.

Author

Chris Jerrett

Definition at line 53 of file claw.h.

Referenced by update_claw().

7.4.1.3 CLAW_D

```
#define CLAW_D .1
```

The derivative constant for the claw PID controller.

Author

Chris Jerrett

Definition at line 19 of file claw.h.

7.4.1.4 CLAW_I

```
#define CLAW_I 0
```

The integral constant for the claw PID controller.

Author

Chris Jerrett

Definition at line 24 of file claw.h.

7.4.1.5 CLAW_OPEN

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

The joystick parameters for opening the claw.

Author

Chris Jerrett

Definition at line 47 of file claw.h.

Referenced by update_claw().

7.4.1.6 CLAW_OPEN_VAL

```
#define CLAW_OPEN_VAL 1500
```

The potentiometer value for a open claw.

Author

Chris Jerrett

Definition at line 59 of file claw.h.

Referenced by update_claw().

7.4.1.7 CLAW_P

```
#define CLAW_P .1
```

The proportional constant for the claw PID controller.

Author

Chris Jerrett

Definition at line 14 of file claw.h.

Referenced by update_claw().

7.4.1.8 MAX_CLAW_SPEED

```
#define MAX_CLAW_SPEED 50
```

The max motor value of the claw.

Author

Chris Jerrett

Definition at line 30 of file claw.h.

Referenced by open_claw().

7.4.1.9 MIN_CLAW_SPEED

```
#define MIN_CLAW_SPEED -50
```

The min motor value of the claw.

Author

Chris Jerrett

Definition at line 35 of file claw.h.

Referenced by close_claw().

7.4.2 Enumeration Type Documentation

7.4.2.1 claw_state

```
enum claw_state
```

The different states of the claw.

Author

Chris Jerrett

Enumerator

CLAW_OPEN_STATE	
CLAW_CLOSE_STATE	

Definition at line 95 of file claw.h.

```
95
96     CLAW_OPEN_STATE,
97     CLAW_CLOSE_STATE
98 };
```

7.4.3 Function Documentation

7.4.3.1 close_claw()

```
void close_claw ( )
```

Drives the motors to close the claw.

Author

Chris Jerrett

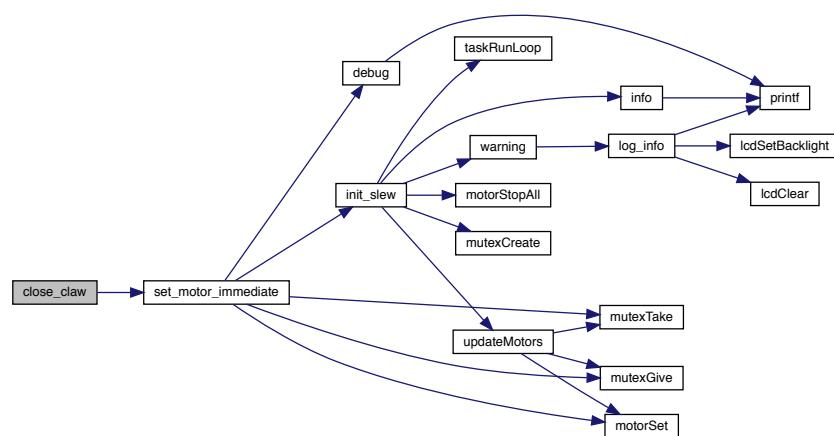
Definition at line 37 of file claw.c.

References CLAW_MOTOR, MIN_CLAW_SPEED, and set_motor_immediate().

Referenced by autonomous().

```
37
38     set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
39 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.2 getClawTicks()

```
unsigned int getClawTicks( )
```

Gets the claw position in potentiometer ticks.

Author

Chris Jerrett

Definition at line 29 of file claw.c.

References analogRead(), and CLAW_POT.

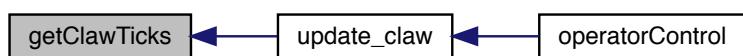
Referenced by update_claw().

```
29 {  
30     return analogRead(CLAW_POT);  
31 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.3 open_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

Author

Chris Jerrett

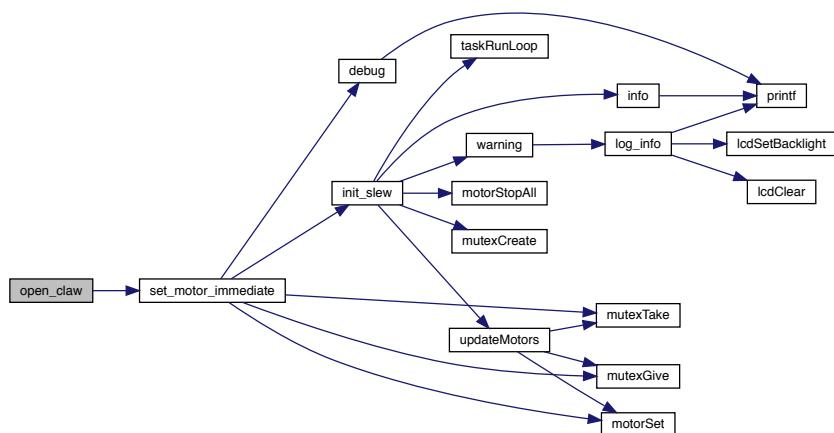
Definition at line 33 of file claw.c.

References CLAW_MOTOR, MAX_CLAW_SPEED, and set_motor_immediate().

Referenced by autonomous().

```
33          {  
34     set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);  
35 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.4 set_claw_motor()

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

sets the claw motor speed

Author

Chris Jerrett

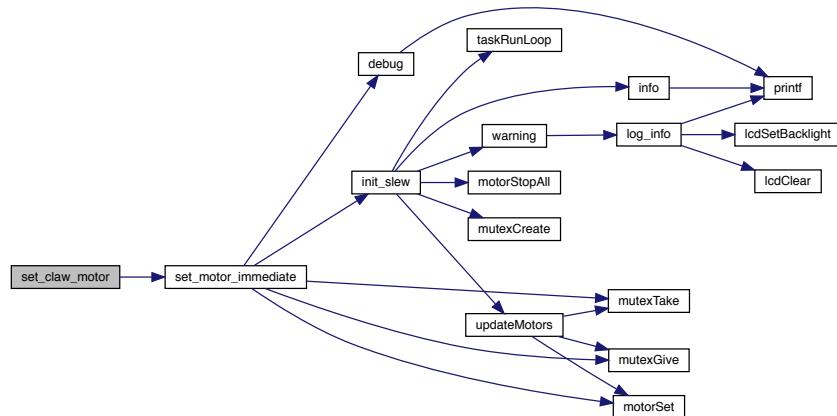
Definition at line 25 of file claw.c.

References CLAW_MOTOR, and set_motor_immediate().

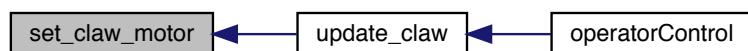
Referenced by update_claw().

```
25
26     set_motor_immediate(CLAW_MOTOR, v);
27 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.4.3.5 update_claw()

```
void update_claw ( )
```

Updates the claw motor values.

Author

Chris Jerrett

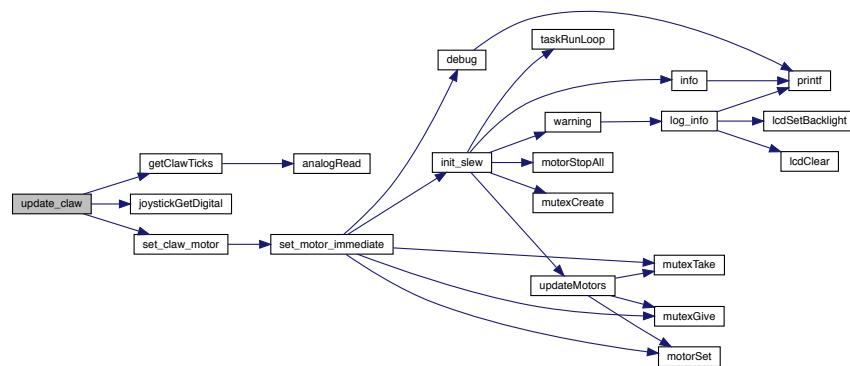
Definition at line 3 of file claw.c.

References CLAW_CLOSE, CLAW_CLOSE_STATE, CLAW_CLOSE_VAL, CLAW_OPEN, CLAW_OPEN_STA←TE, CLAW_OPEN_VAL, CLAW_P, getClawTicks(), joystickGetDigital(), and set_claw_motor().

Referenced by operatorControl().

```
3             {
4     static int last_error = 0;
5     static enum claw_state state = CLAW_OPEN_STATE;
6     if(joystickGetDigital(CLAW_CLOSE)) {
7         state = CLAW_CLOSE_STATE;
8     }
9     else if(joystickGetDigital(CLAW_OPEN)) {
10        state = CLAW_OPEN_STATE;
11    } else {
12        int p = 0;
13        if(state == CLAW_OPEN_STATE) {
14            p = getClawTicks() - CLAW_OPEN_VAL;
15        } else {
16            p = getClawTicks() - CLAW_CLOSE_VAL;
17        }
18        int d = (p - last_error);
19        int motor = CLAW_P * p;
20        set_claw_motor(motor);
21    }
22 }
23 }
```

Here is the call graph for this function:



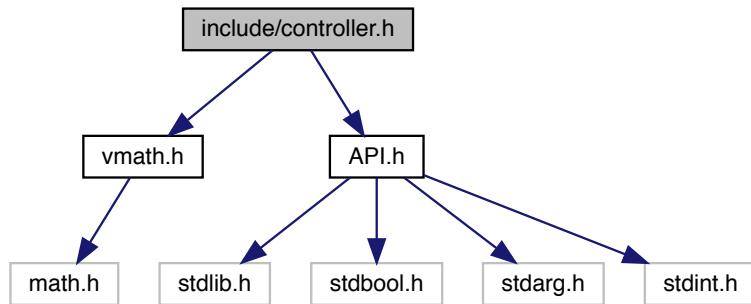
Here is the caller graph for this function:



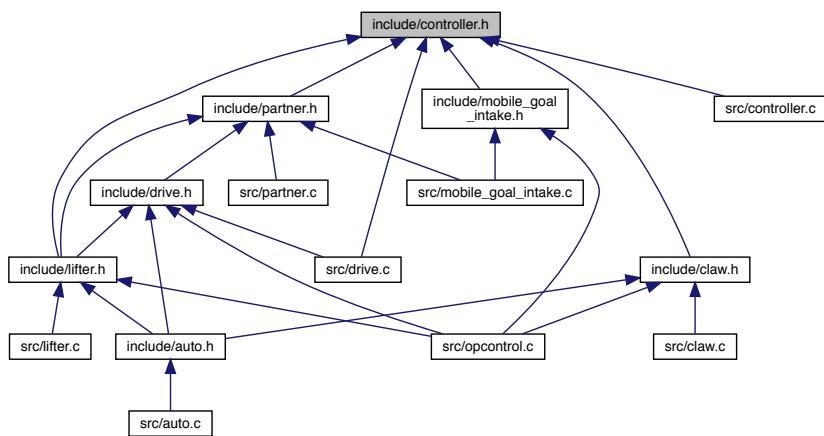
7.5 include/controller.h File Reference

controller definitions, macros

```
#include "vmath.h"
#include <API.h>
Include dependency graph for controller.h:
```



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



Macros

- `#define LEFT_BUMPERS 6`
- `#define LEFT_BUTTONS 7`
- `#define LEFT_JOY_X 4`
the left x joystick on controller
- `#define LEFT_JOY_Y 3`
the left y joystick on controller

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

Enumerations

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

Functions

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

7.5.1 Detailed Description

controller definitions, macros

Author

Chris Jerrett

Date

9/9/2017

7.5.2 Macro Definition Documentation

7.5.2.1 LEFT_BUMPERS

```
#define LEFT_BUMPERS 6
```

Definition at line 17 of file controller.h.

7.5.2.2 LEFT_BUTTONS

```
#define LEFT_BUTTONS 7
```

Definition at line 15 of file controller.h.

7.5.2.3 LEFT_JOY_X

```
#define LEFT_JOY_X 4
```

the left x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 52 of file controller.h.

Referenced by get_joystick_cord().

7.5.2.4 LEFT_JOY_Y

```
#define LEFT_JOY_Y 3
```

the left y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 59 of file controller.h.

Referenced by get_joystick_cord().

7.5.2.5 MASTER

```
#define MASTER 1
```

the master controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 24 of file controller.h.

Referenced by update_drive_motors(), and updateIntake().

7.5.2.6 PARTNER

```
#define PARTNER 2
```

the slave/partner controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 31 of file controller.h.

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

7.5.2.7 RIGHT_BUMPERS

```
#define RIGHT_BUMPERS 5
```

Definition at line 16 of file controller.h.

7.5.2.8 RIGHT_BUTTONS

```
#define RIGHT_BUTTONS 8
```

Definition at line 14 of file controller.h.

7.5.2.9 RIGHT_JOY_X

```
#define RIGHT_JOY_X 1
```

the right x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 38 of file controller.h.

Referenced by get_joystick_cord().

7.5.2.10 RIGHT_JOY_Y

```
#define RIGHT_JOY_Y 2
```

the right y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 45 of file controller.h.

Referenced by get_joystick_cord().

7.5.3 Enumeration Type Documentation

7.5.3.1 joystick

```
enum joystick
```

Represents a joystick on the controller.

Date

9/10/2017

Author

Chris Jerrett

Enumerator

<code>RIGHT_JOY</code>	The right joystick
<code>LEFT_JOY</code>	The left joystick

Definition at line 66 of file controller.h.

```
66
68     RIGHT_JOY,
70     LEFT_JOY,
71 };
```

7.5.4 Function Documentation

7.5.4.1 `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 3 of file controller.c.

References `joystickGetAnalog()`, `LEFT_JOY_X`, `LEFT_JOY_Y`, `RIGHT_JOY`, `RIGHT_JOY_X`, `RIGHT_JOY_Y`, `cord::x`, and `cord::y`.

```
3
4     int x;
5     int y;
6     if(side == RIGHT_JOY) {
7         y = joystickGetAnalog(controller, RIGHT_JOY_X);
8         x = joystickGetAnalog(controller, RIGHT_JOY_Y);
9     } else {
10        y = joystickGetAnalog(controller, LEFT_JOY_X);
11        x = joystickGetAnalog(controller, LEFT_JOY_Y);
12    }
13    struct cord c;
14    c.x = x;
15    c.y = y;
16    return c;
17 }
```

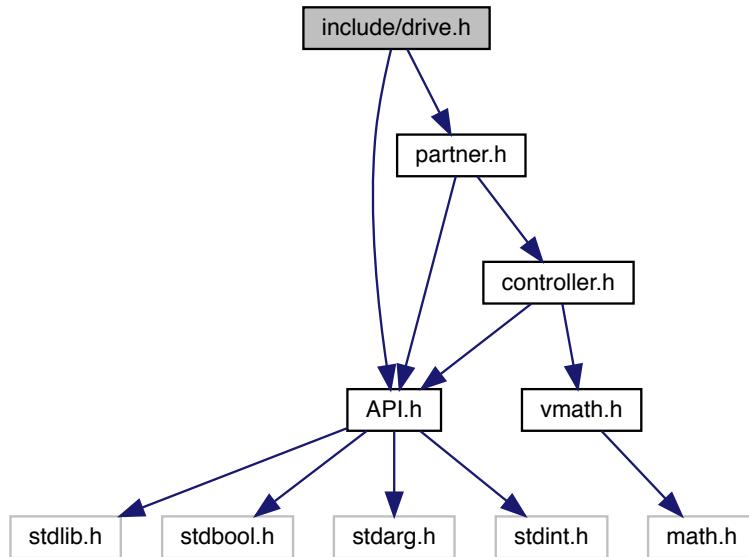
Here is the call graph for this function:



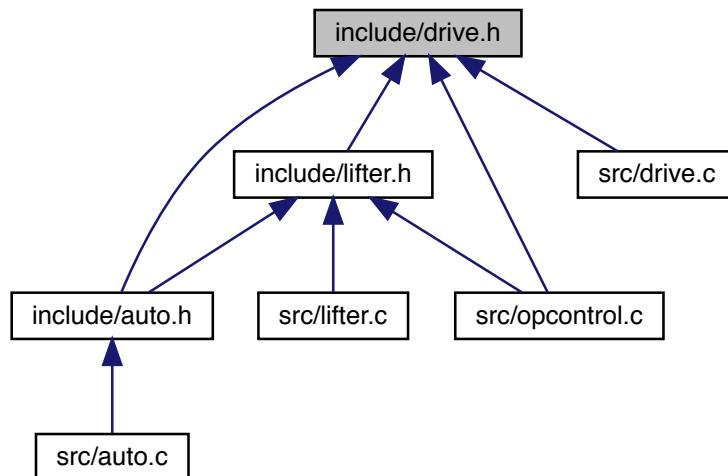
7.6 include/drive.h File Reference

Drive base definitions and enumerations.

```
#include <API.h>
#include "partner.h"
Include dependency graph for drive.h:
```



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



Macros

- `#define DEADSPOT 10`
- `#define THRESHOLD 10`

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.

7.6.1 Detailed Description

Drive base definitions and enumerations.

Author

Christian Desimone

Date

9/9/2017

7.6.2 Macro Definition Documentation

7.6.2.1 DEADSPOT

```
#define DEADSPOT 10
```

Definition at line 14 of file drive.h.

7.6.2.2 THRESHOLD

```
#define THRESHOLD 10
```

Definition at line 15 of file drive.h.

Referenced by joystickExp().

7.6.3 Typedef Documentation

7.6.3.1 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 or left. Contained in side typedef, so enum is unnecessary.

7.6.4 Enumeration Type Documentation

7.6.4.1 side

```
enum side
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

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

Enumerator

LEFT	
BOTH	
RIGHT	

Generated by Doxygen

Definition at line 22 of file drive.h.

```
22      {
23      LEFT,
24      BOTH,
25      RIGHT
26 } side_t;
```

7.6.5 Function Documentation

7.6.5.1 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

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

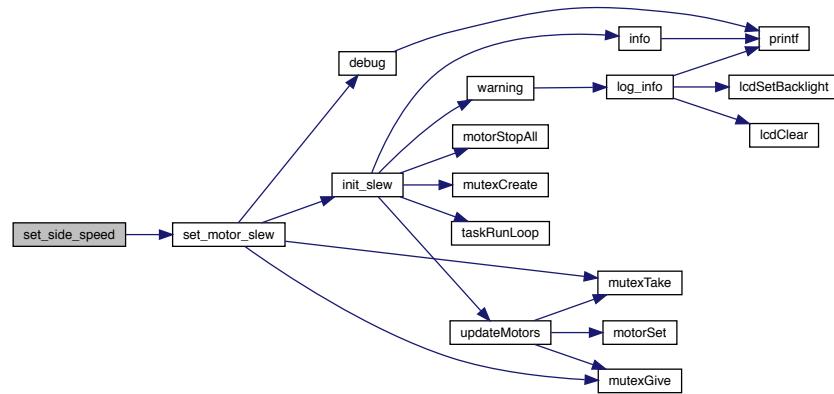
Definition at line 50 of file drive.c.

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().

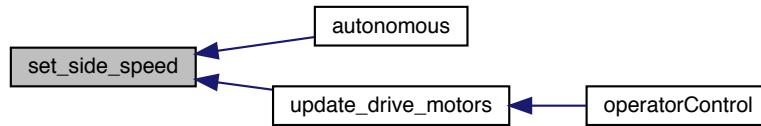
Referenced by autonomous(), and update_drive_motors().

```
50      if(side == RIGHT || side == BOTH) {
51          set_motor_slew(MOTOR_BACK_RIGHT, -speed);
52          set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
53          set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
54      }
55      if(side == LEFT || side == BOTH) {
56          set_motor_slew(MOTOR_BACK_LEFT, speed);
57          set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
58          set_motor_slew(MOTOR_FRONT_LEFT, speed);
59      }
60  }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.6.5.2 setThresh()

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

Sets the deadzone threshhold on the drive.

Author

Chris Jerrett

Definition at line 17 of file `drive.c`.

References `thresh`.

```
17
18     thresh = t;
19 }
```

7.6.5.3 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

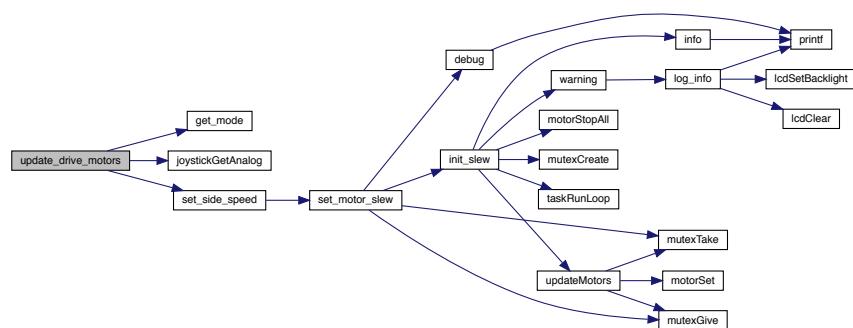
Definition at line 21 of file drive.c.

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

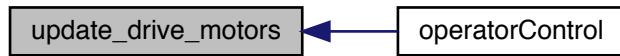
Referenced by operatorControl().

```
21
22
23     int x = 0;
24     int y = 0;
25     if(get_mode() == PARTNER_CONTROLLER_MODE) {
26         x = (joystickGetAnalog(PARTNER, 3));
27         y = (joystickGetAnalog(PARTNER, 1));
28     } else {
29         x = -(joystickGetAnalog(MASTER, 3));
30         y = (joystickGetAnalog(MASTER, 1));
31     }
32
33 //x = joystickExp(x);
34 //y = joystickExp(y);
35 if(x < thresh && x > -thresh) {
36     x = 0;
37 }
38 if(y < thresh && y > -thresh) {
39     y = 0;
40 }
41
42     int r = (x + y);
43     int l = -(x - y);
44
45     set_side_speed(LEFT, l);
46     set_side_speed(RIGHT, -r);
47
48 }
```

Here is the call graph for this function:



Here is the caller graph for this function:

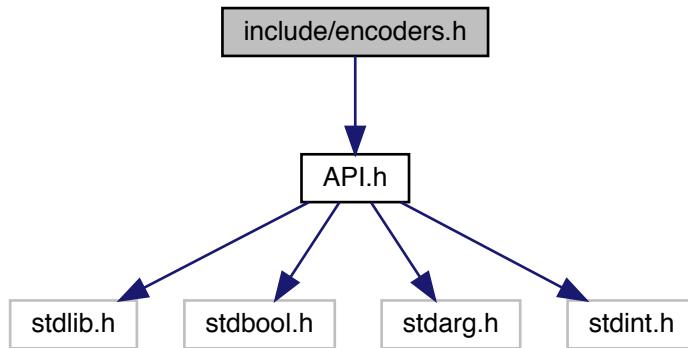


7.7 include/encoders.h File Reference

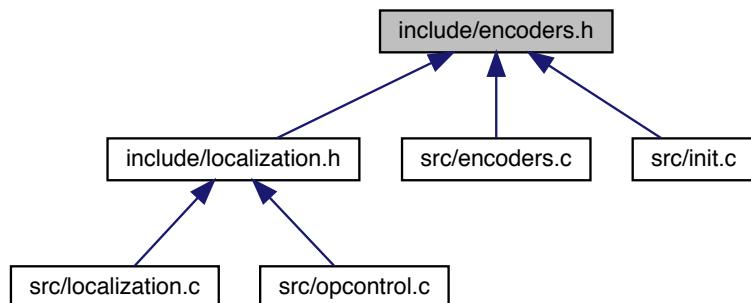
wrapper around encoder functions

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

Include dependency graph for encoders.h:



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



Macros

- `#define IME_NUMBER 6`

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

Functions

- `int get_encoder_ticks (unsigned char address)`
Gets the encoder ticks since last reset.
- `int get_encoder_velocity (unsigned char address)`
Gets the encoder reads.
- `bool init_encoders ()`
Initializes all motor encoders.

7.7.1 Detailed Description

wrapper around encoder functions

Author

Chris Jerrett

Date

9/9/2017

7.7.2 Macro Definition Documentation

7.7.2.1 IME_NUMBER

```
#define IME_NUMBER 6
```

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

See also

[init_encoders\(\)](#)

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#)

Definition at line 21 of file encoders.h.

Referenced by `init_encoders()`.

7.7.3 Function Documentation

7.7.3.1 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 12 of file encoders.c.

References imeGet().

```
12
13     int i = 0;
14     imeGet(address, &i);
15     return i;
16 }
```

Here is the call graph for this function:



7.7.3.2 get_encoder_velocity()

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

Gets the encoder reads.

Author

Chris Jerrett

Date

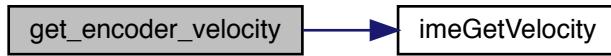
9/15/2017

Definition at line 18 of file encoders.c.

References `imeGetVelocity()`.

```
18
19     int i = 0;
20     imeGetVelocity(address, &i);
21     return i;
22 }
```

Here is the call graph for this function:



7.7.3.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#)

Definition at line 4 of file encoders.c.

References [IME_NUMBER](#), and [imeInitializeAll\(\)](#).

```
4
5  #ifdef IME_NUMBER
6  return imeInitializeAll() == IME_NUMBER;
7  #else
8  return imeInitializeAll();
9  #endif
10 }
```

Here is the call graph for this function:

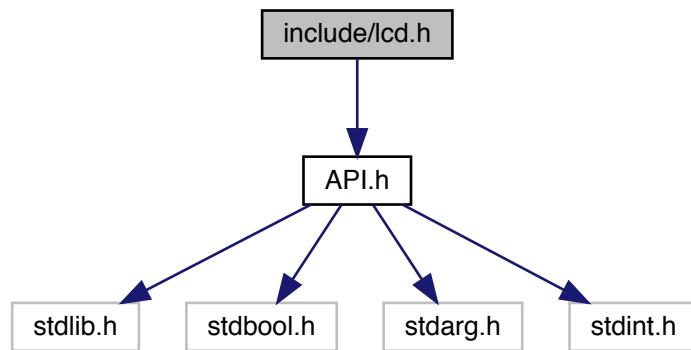


7.8 include/lcd.h File Reference

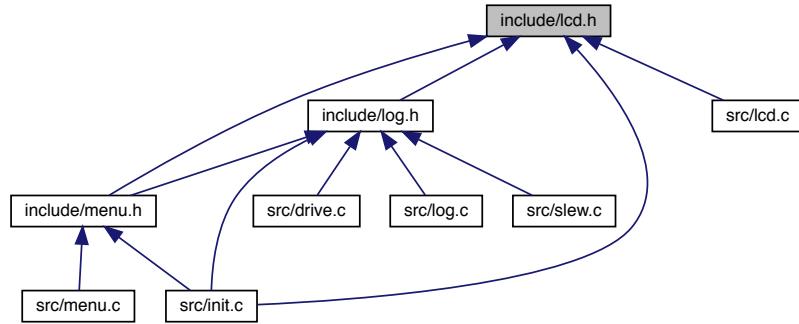
LCD wrapper functions and macros.

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

Include dependency graph for lcd.h:



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



Data Structures

- struct `lcd_buttons`
represents the state of the lcd buttons

Macros

- #define `BOTTOM_ROW` 2
The bottom row on the lcd screen.
- #define `TOP_ROW` 1
The top row on the lcd screen.

Enumerations

- enum `button_state` { `RELEASED` = false, `PRESSED` = true }
Represents the state of a button.

Functions

- void `init_main_lcd` (`FILE *lcd`)
Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- void `lcd_clear` ()
Clears the lcd.
- `lcd_buttons lcd_get_pressed_buttons` ()
Returns the pressed buttons.
- void `lcd_print` (`unsigned int line, const char *str`)
prints a string to a line on the lcd
- void `lcd_printf` (`unsigned int line, const char *format_str, ...`)
prints a formated string to a line on the lcd. Smilar to printf
- void `lcd_set_backlight` (`bool state`)
sets the backlight of the lcd
- void `prompt_confirmation` (`const char *confirm_text`)
Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

7.8.1 Detailed Description

LCD wrapper functions and macros.

Author

Chris Jerrett

Date

9/9/2017

7.8.2 Macro Definition Documentation

7.8.2.1 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().

7.8.2.2 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().

7.8.3 Enumeration Type Documentation

7.8.3.1 button_state

```
enum button_state
```

Represents the state of a button.

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

Author

Chris Jerrett

Date

9/9/2017

Enumerator

RELEASED	A released button
PRESSED	A pressed button

Definition at line 36 of file lcd.h.

```
36      {
38      RELEASED = false,
40      PRESSED = true,
41 } button_state;
```

7.8.4 Function Documentation

7.8.4.1 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

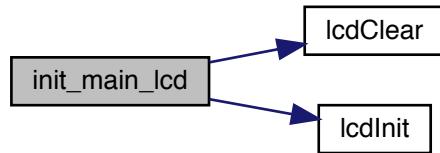
9/9/2017

Definition at line 39 of file lcd.c.

References `lcd_port`, `lcdClear()`, and `lcdInit()`.

```
39      lcdInit(lcd);  
40      lcdClear(lcd);  
41      lcd_port = lcd;  
42  }  
43 }
```

Here is the call graph for this function:



7.8.4.2 `lcd_clear()`

```
void lcd_clear( )
```

Clears the lcd.

Author

Chris Jerrett

Date

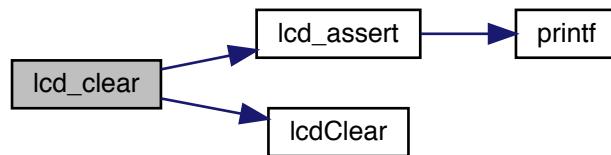
9/9/2017

Definition at line 34 of file lcd.c.

References `lcd_assert()`, `lcd_port`, and `lcdClear()`.

```
34      {
35     lcd_assert();
36     lcdClear(lcd_port);
37 }
```

Here is the call graph for this function:



7.8.4.3 `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](#)

Definition at line 20 of file lcd.c.

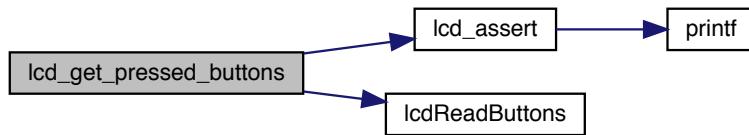
References `lcd_assert()`, `lcd_port`, `lcdReadButtons()`, `lcd_buttons::left`, `lcd_buttons::middle`, `PRESSED`, `RELEASED`, and `lcd_buttons::right`.

Referenced by `display_menu()`, and `prompt_confirmation()`.

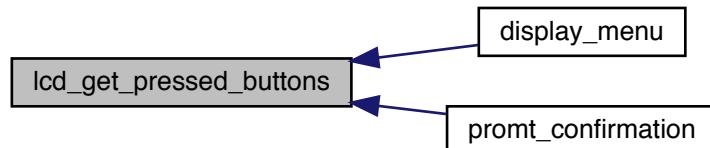
```

20
21     lcd_assert();
22     unsigned int btn_binary = lcdReadButtons(lcd_port);
23     bool left = btn_binary & 0x1;
24     bool middle = btn_binary & 0x2;
25     bool right = btn_binary & 0x4;
26     lcd_buttons btns;
27     btns.left = left ? PRESSED : RELEASED;
28     btns.middle = middle ? PRESSED : RELEASED;
29     btns.right = right ? PRESSED : RELEASED;
30
31     return btns;
32 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4.4 lcd_print()

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

prints a string to a line on the lcd

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

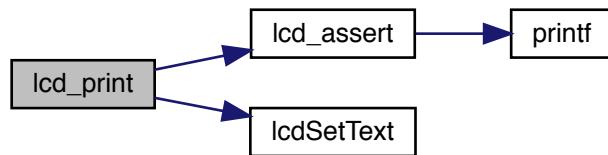
Definition at line 45 of file lcd.c.

References `lcd_assert()`, `lcd_port`, and `lcdSetText()`.

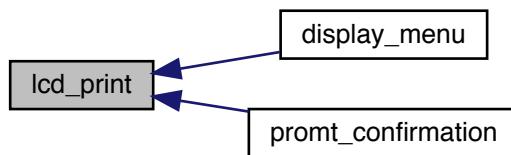
Referenced by `display_menu()`, and `prompt_confirmation()`.

```
45
46     lcd_assert();
47     lcdSetText(lcd_port, line, str);
48 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.8.4.5 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

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

Author

Chris Jerrett

Date

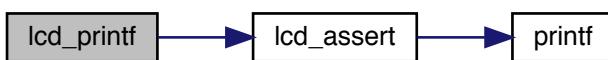
9/9/2017

Definition at line 50 of file lcd.c.

References lcd_assert(), and lcd_port.

```
50
51     lcd_assert();
52     lcdPrint(lcd_port, line, format_str);
53 }
```

Here is the call graph for this function:



7.8.4.6 lcd_set_backlight()

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

sets the backlight of the lcd

Parameters

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

Author

Chris Jerrett

Date

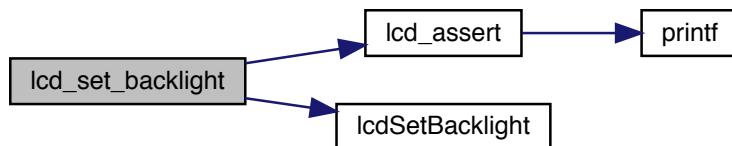
9/9/2017

Definition at line 55 of file lcd.c.

References `lcd_assert()`, `lcd_port`, and `lcdSetBacklight()`.

```
55
56     lcd_assert();
57     lcdSetBacklight(lcd_port, state);
58 }
```

Here is the call graph for this function:

**7.8.4.7 prompt_confirmation()**

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

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

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

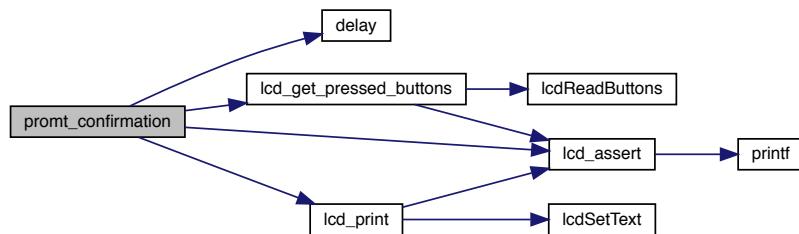
Definition at line 60 of file lcd.c.

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

```

60
61     lcd_assert();
62     lcd_print(1, confirm_text);
63     while(lcd_get_pressed_buttons().middle != PRESSED) {
64         delay(200);
65     }
66 }
```

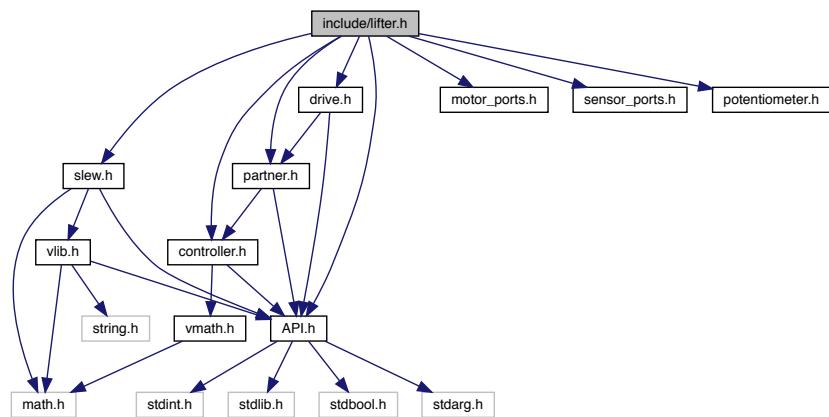
Here is the call graph for this function:



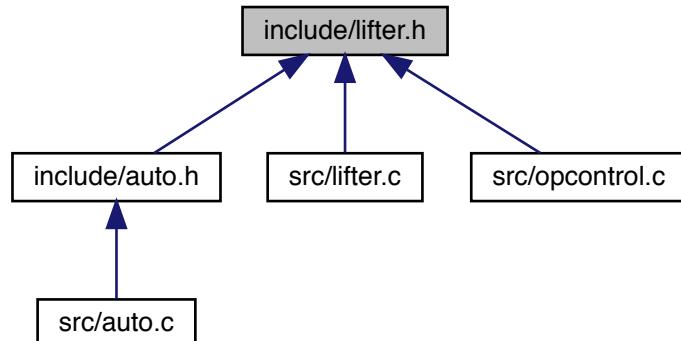
7.9 include/lifter.h File Reference

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

```
#include "drive.h"
Include dependency graph for lifter.h:
```



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



Macros

- #define HEIGHT 19.1 - 3.8
- #define LENGTH_LONG 13.5
- #define LENGTH_SHORT 10.5
- #define LIFTER_D 0
- #define LIFTER_DOWN_MASTER, 5, JOY_DOWN
- #define LIFTER_DOWN_PARTNER PARTNER, 5, JOY_DOWN
- #define LIFTER_DRIVER_LOAD_MASTER, RIGHT_BUTTONS, JOY_RIGHT
- #define LIFTER_I 0
- #define LIFTER_P .15
- #define LIFTER_UP_MASTER, 5, JOY_UP
- #define LIFTER_UP_PARTNER PARTNER, 5, JOY_UP
- #define TICK_DIFF 680

Functions

- double `getLifterHeight ()`
- int `getLifterTicks ()`
- float `lifterPotentiometerToDegree (int x)`
- void `set_lifter_motors (const int)`
- void `set_lifter_pos (int pos)`
- void `update_lifter ()`

7.9.1 Macro Definition Documentation

7.9.1.1 HEIGHT

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

Definition at line 20 of file lifter.h.

7.9.1.2 LENGTH_LONG

```
#define LENGTH_LONG 13.5
```

Definition at line 19 of file lifter.h.

7.9.1.3 LENGTH_SHORT

```
#define LENGTH_SHORT 10.5
```

Definition at line 21 of file lifter.h.

7.9.1.4 LIFTER_D

```
#define LIFTER_D 0
```

Definition at line 16 of file lifter.h.

Referenced by `update_lifter()`.

7.9.1.5 LIFTER_DOWN

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

Definition at line 24 of file lifter.h.

Referenced by update_lifter().

7.9.1.6 LIFTER_DOWN_PARTNER

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

Definition at line 28 of file lifter.h.

Referenced by update_lifter().

7.9.1.7 LIFTER_DRIVER_LOAD

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

Definition at line 25 of file lifter.h.

Referenced by update_lifter().

7.9.1.8 LIFTER_I

```
#define LIFTER_I 0
```

Definition at line 17 of file lifter.h.

Referenced by update_lifter().

7.9.1.9 LIFTER_P

```
#define LIFTER_P .15
```

Definition at line 15 of file lifter.h.

Referenced by update_lifter().

7.9.1.10 LIFTER_UP

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

Definition at line 23 of file lifter.h.

Referenced by update_lifter().

7.9.1.11 LIFTER_UP_PARTNER

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

Definition at line 27 of file lifter.h.

Referenced by update_lifter().

7.9.1.12 TICK_DIFF

```
#define TICK_DIFF 680
```

Definition at line 13 of file lifter.h.

Referenced by lifterPotentiometerToDegree().

7.9.2 Function Documentation

7.9.2.1 getLifterHeight()

```
double getLifterHeight ( )
```

Definition at line 86 of file lifter.c.

References getLifterTicks().

```
86      {
87      unsigned int ticks = getLifterTicks();
88      return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks)))) + 0.0198 * ticks + 2.3033;
89 }
```

Here is the call graph for this function:



7.9.2.2 getLifterTicks()

```
int getLifterTicks ( )
```

Definition at line 81 of file lifter.c.

References analogRead(), and LIFTER.

Referenced by getLifterHeight(), and update_lifter().

```
81     {
82     return analogRead(LIFTER);
83 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.3 lifterPotentiometerToDegree()

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

Definition at line 77 of file lifter.c.

References DEG_MAX, TICK_DIFF, and TICK_MAX.

```
77     {
78     return (x - TICK_DIFF) / TICK_MAX * DEG_MAX;
79 }
```

7.9.2.4 set_lifter_motors()

```
void set_lifter_motors (
    const int    )
```

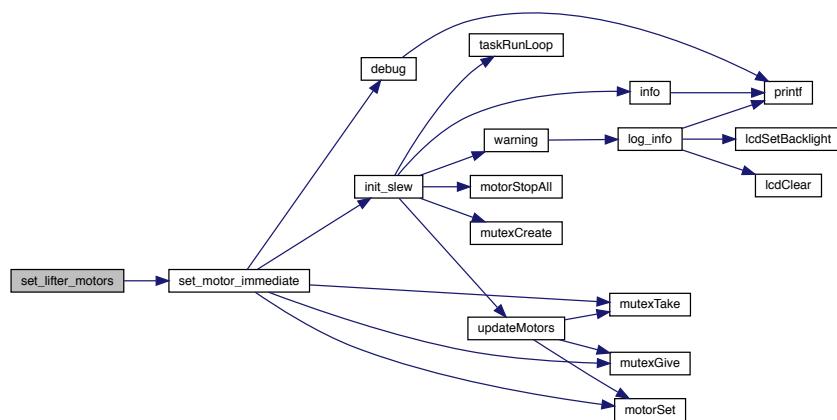
Definition at line 3 of file lifter.c.

References MOTOR_LIFT_TOP_LEFT, MOTOR_LIFT_TOP_RIGHT, and set_motor_immediate().

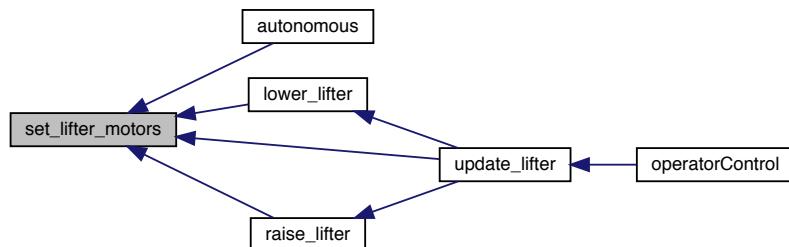
Referenced by autonomous(), lower_lifter(), raise_lifter(), and update_lifter().

```
3     {
4     set_motor_immediate(MOTOR_LIFT_TOP_RIGHT, -v);
5     set_motor_immediate(MOTOR_LIFT_TOP_LEFT, -v);
6 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.9.2.5 set_lifter_pos()

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

Definition at line 8 of file lifter.c.

```
8
9
10 }
```

7.9.2.6 update_lifter()

```
void update_lifter ( )
```

Definition at line 20 of file lifter.c.

References get_mode(), getLifterTicks(), joystickGetDigital(), LIFTER_D, LIFTER_DOWN, LIFTER_DOWN_PARTNER, LIFTER_DRIVER_LOAD, LIFTER_I, LIFTER_P, LIFTER_UP, LIFTER_UP_PARTNER, lower_lifter(), MAIN_CONTROLLER_MODE, PARTNER_CONTROLLER_MODE, raise_lifter(), and set_lifter_motors().

Referenced by operatorControl().

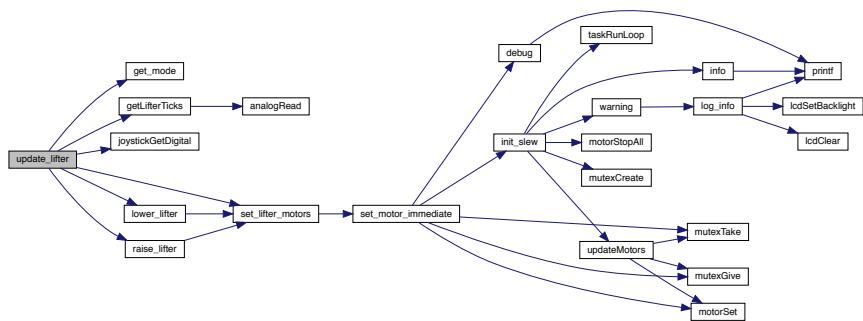
```
20
21     static bool changed = true;
22     static unsigned int target = 0;
23     static bool first_run = true;
24     if(first_run) {
25         target = getLifterTicks();
26         first_run = false;
27     }
28     static int last_error = 0;
29     static long long i = 0;
30     if((joystickGetDigital(LIFTER_UP) && get_mode() == MAIN_CONTROLLER_MODE)
31     || (joystickGetDigital(LIFTER_UP_PARTNER) && get_mode() == PARTNER_CONTROLLER_MODE)) {
32         changed = true;
33         i = 0;
34         target = getLifterTicks() + 200;
35         lower_lifter();
36     }
37     else if((joystickGetDigital(LIFTER_DOWN) &&
38             get_mode() == MAIN_CONTROLLER_MODE)
39     || (joystickGetDigital(LIFTER_DOWN_PARTNER) &&
40             get_mode() == PARTNER_CONTROLLER_MODE)) {
41         changed = true;
42         i = 0;
43         target = getLifterTicks();
44         raise_lifter();
45     }
46     else if(joystickGetDigital(LIFTER_DRIVER_LOAD) &&
47             get_mode() == MAIN_CONTROLLER_MODE) {
48         changed = true;
49         i = 0;
50         int k = 0;
51         if(getLifterTicks() < 1270) {
52             lower_lifter();
53         }
54         if(getLifterTicks() > 1230) {
55             raise_lifter();
56             //k = 100;
57         }
58         target = 1250;
59         //raise_lifter();
60     }
61     else {
```

```

60     if(get_mode() == PARTNER_CONTROLLER_MODE) {
61         set_lifter_motors(0);
62         return;
63     }
64     int p = getLifterTicks() - target;
65     int d = p - last_error;
66     i += p;
67     int motor = LIFTER_P * p + LIFTER_D * d + LIFTER_I * i;
68     if (motor < 10) {
69         set_lifter_motors(0);
70     } else {
71         set_lifter_motors(motor);
72     }
73 }
74 }
75 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



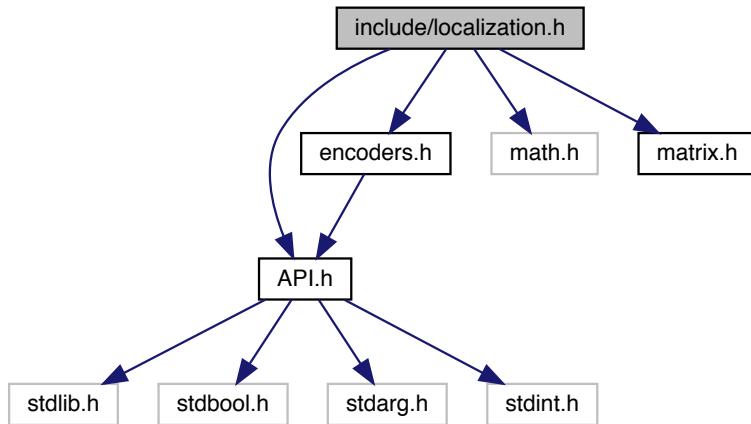
7.10 include/localization.h File Reference

```

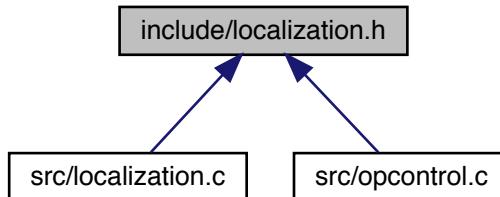
#include <API.h>
#include "encoders.h"
#include <math.h>
#include "matrix.h"

```

Include dependency graph for localization.h:



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



Data Structures

- struct [location](#)

Macros

- #define [LOCALIZATION_UPDATE_FREQUENCY](#) 0.500

Functions

- struct [location get_position \(\)](#)
- bool [init_localization](#) (const unsigned char gyro1, unsigned short multiplier, int start_x, int start_y, int start_theta)

Starts the localization process.

7.10.1 Macro Definition Documentation

7.10.1.1 LOCALIZATION_UPDATE_FREQUENCY

```
#define LOCALIZATION_UPDATE_FREQUENCY 0.500
```

Definition at line 9 of file localization.h.

Referenced by calculate_gryo_angular_velocity(), init_localization(), and integrate_gyro_w().

7.10.2 Function Documentation

7.10.2.1 get_position()

```
struct location get_position ( )
```

Definition at line 25 of file localization.c.

```
25
26
27 }
```

7.10.2.2 init_localization()

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

Starts the localization process.

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.
-------	---

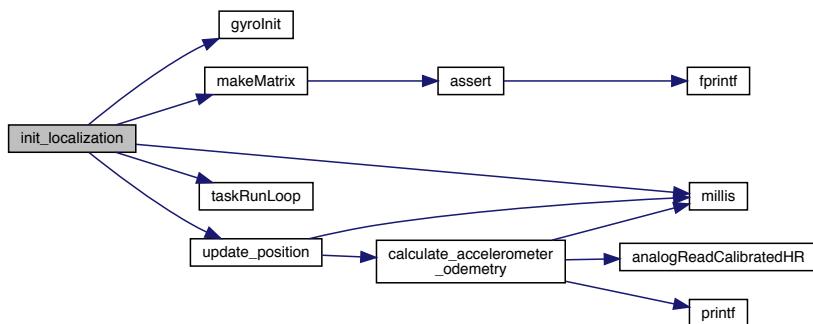
Definition at line 89 of file localization.c.

References g1, gyroInit(), last_call, localization_task, LOCALIZATION_UPDATE_FREQUENCY, makeMatrix(), millis(), taskRunLoop(), and update_position().

```

89
90     g1 = gyroInit(gyro1, multiplier);
91     //init state matrix
92
93     //one dimensional vector with x, y, theta, acceleration in x and y
94     state_matrix = makeMatrix(1, 5);
95     localization_task = taskRunLoop(update_position,
96                                     LOCALIZATION_UPDATE_FREQUENCY * 1000);
96     last_call = millis();
97     return true;
98 }
```

Here is the call graph for this function:

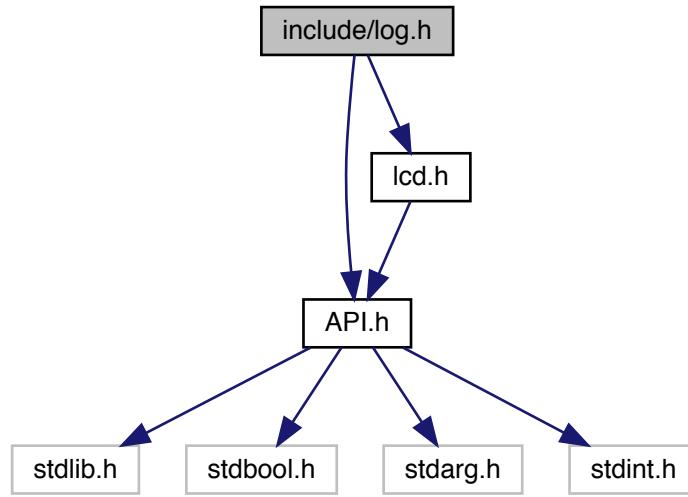


7.11 include/log.h File Reference

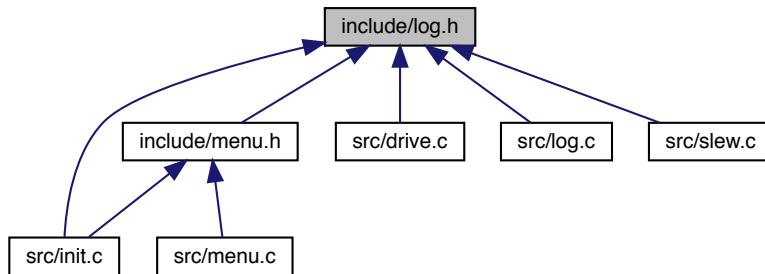
Contains logging functions.

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

Include dependency graph for log.h:



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



Macros

- #define DEBUG 4
logging only info debug. most verbose level
- #define ERROR 1
logging only errors. Also displays error to lcd
- #define INFO 3
logging only info messages and higher.
- #define NONE 0
No logging. Should be used in competition to reduce serial communication.
- #define WARNING 2
logs errors and warnings. Also displays error to lcd

Functions

- void **debug** (const char *debug_message)
prints a info message
- void **error** (const char *error_message)
prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE
- void **info** (const char *info_message)
prints a info message
- void **init_error** (bool use_lcd, FILE *lcd)
Initializes the error lcd system Only required if using lcd.
- void **warning** (const char *warning_message)
prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

7.11.1 Detailed Description

Contains logging functions.

Author

Chris Jerrett

Date

9/16/2017

7.11.2 Macro Definition Documentation

7.11.2.1 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.

7.11.2.2 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().

7.11.2.3 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.

7.11.2.4 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().

7.11.2.5 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().

7.11.3 Function Documentation

7.11.3.1 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](#)

Parameters

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

Definition at line 37 of file log.c.

References ERROR, log_level, and printf().

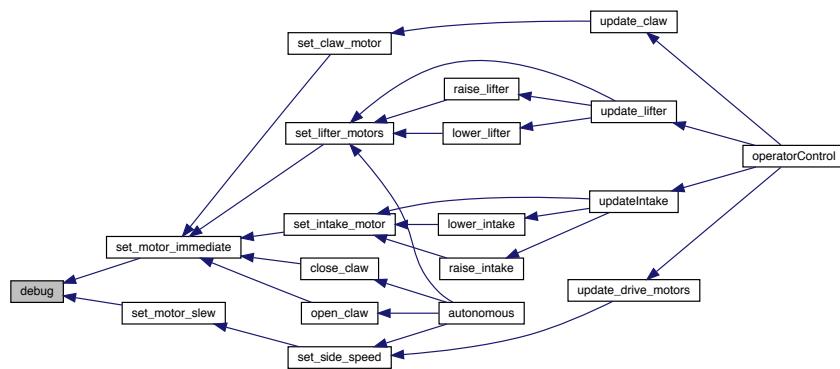
Referenced by set_motor_immediate(), and set_motor_slew().

```
37
38     if(log_level>ERROR) {
39         printf("[INFO]: %s\n", debug_message);
40     }
41 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.2 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](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

<code>error_message</code>	the message
----------------------------	-------------

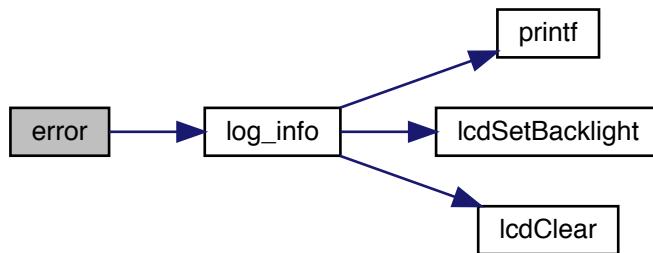
Definition at line 21 of file log.c.

References log_info(), log_level, and NONE.

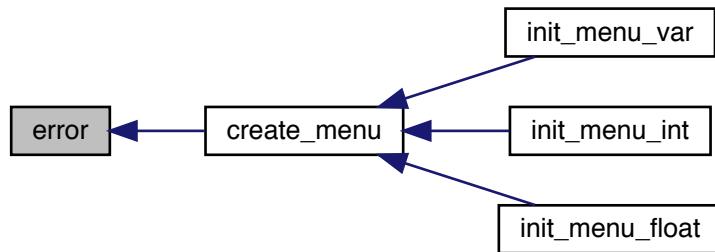
Referenced by create_menu().

```
21
22     if(log_level>NONE)
23         log_info("ERROR", error_message);
24 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.3 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](#)

Parameters

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

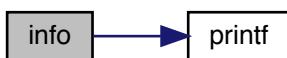
Definition at line 31 of file log.c.

References ERROR, log_level, and printf().

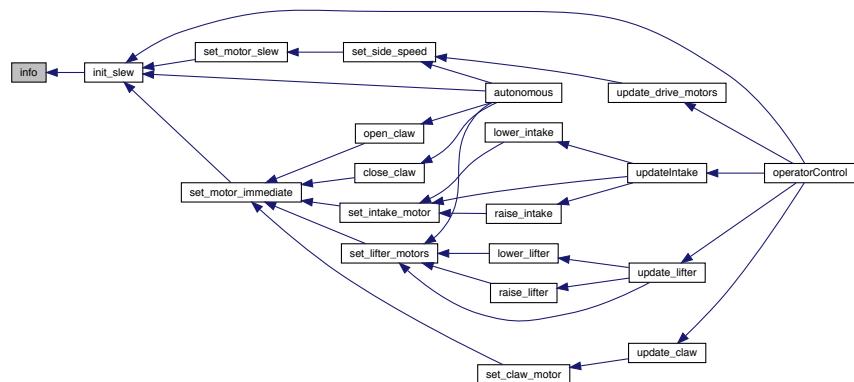
Referenced by init_slew().

```
31
32     if(log_level>ERROR) {
33         printf("[INFO]: %s\n", info_message);
34     }
35 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.11.3.4 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

<code>use_lcd</code>	whether to use the lcd
<code>lcd</code>	the lcd

Definition at line 6 of file log.c.

References `lcdInit()`, and `log_lcd`.

```
6
7     if(use_lcd) {
8         lcdInit(lcd);
9         log_lcd = lcd;
10    }
11 }
```

Here is the call graph for this function:



7.11.3.5 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](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

<i>warning_message</i>	the message
------------------------	-------------

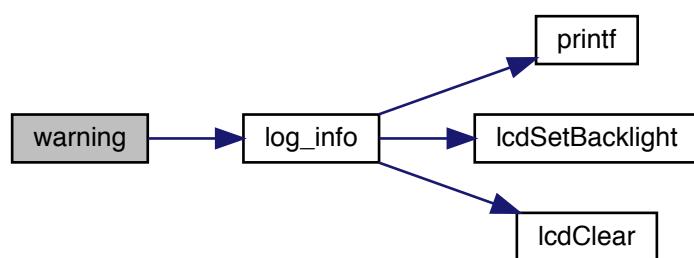
Definition at line 26 of file log.c.

References [log_info\(\)](#), [log_level](#), and [WARNING](#).

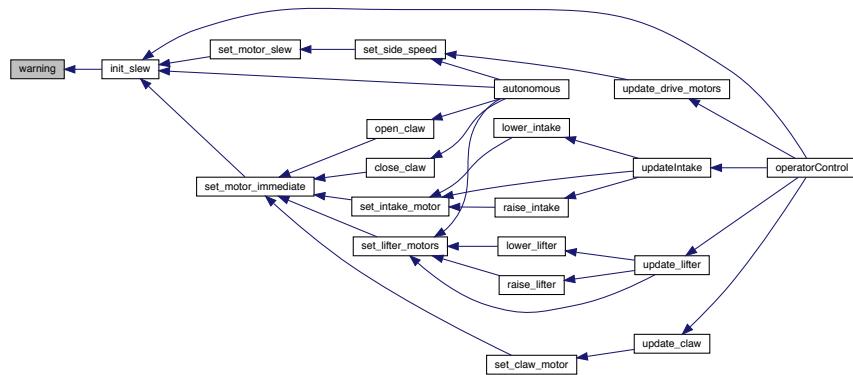
Referenced by [init_slew\(\)](#).

```
26
27     if(log_level>WARNING)
28         log_info("WARNING", warning_message);
29 }
```

Here is the call graph for this function:



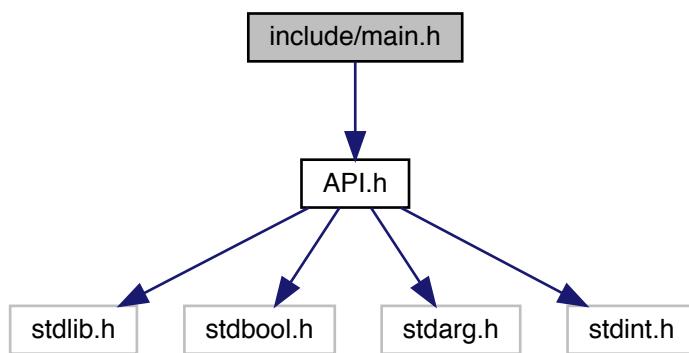
Here is the caller graph for this function:



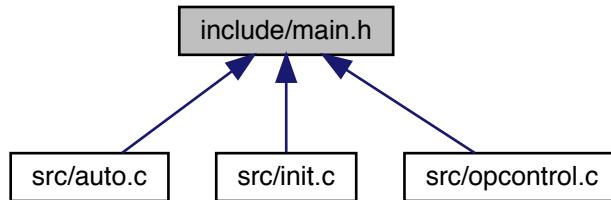
7.12 include/main.h File Reference

Header file for global functions.

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



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



Functions

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

7.12.1 Detailed Description

Header file for global functions.

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

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

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

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

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

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

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

7.12.2 Function Documentation

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

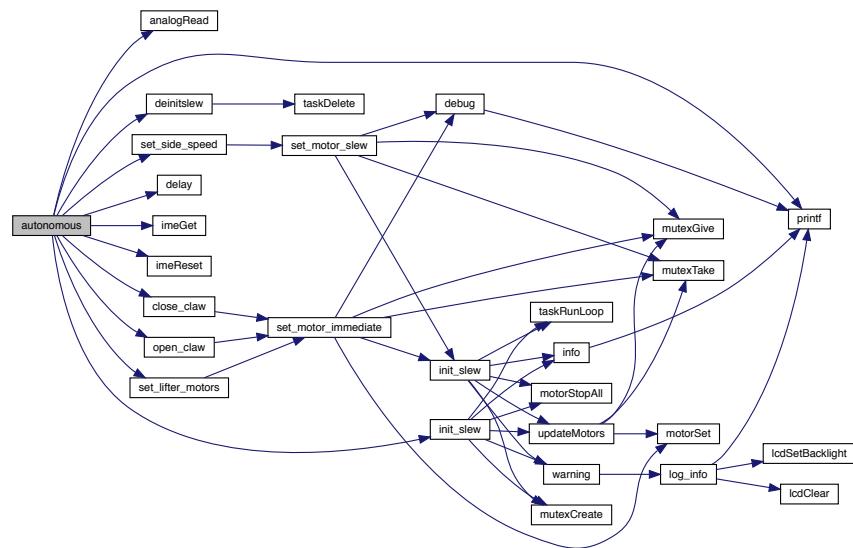
Definition at line 30 of file auto.c.

References `analogRead()`, `BOTH`, `close_claw()`, `deinitSlew()`, `delay()`, `GOAL_HEIGHT`, `imeGet()`, `imeReset()`, `init_slew()`, `LIFTER`, `MID_LEFT_DRIVE`, `MID_RIGHT_DRIVE`, `open_claw()`, `printf()`, `set_lifter_motors()`, and `set_side_speed()`.

```

30
31     init_slew();
32
33     delay(10);
34     printf("auto\n");
35     int counts_drive_left;
36     int counts_drive_right;
37     int counts_drive;
38     imeReset(MID_LEFT_DRIVE);
39     imeReset(MID_RIGHT_DRIVE);
40     imeGet(MID_LEFT_DRIVE, &counts_drive_left);
41     imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
42     counts_drive = counts_drive_left + counts_drive_right;
43     counts_drive /= 2;
44     close_claw();
45
46     delay(300);
47     //unsigned long time = millis();
48     while(analogRead(LIFTER) < GOAL_HEIGHT) {
49         set_lifter_motors(-127);
50     }
51     set_lifter_motors(0);
52     while(counts_drive_left < 530) {
53         set_side_speed(BOTH, 127);
54         imeGet(MID_LEFT_DRIVE, &counts_drive_left);
55         imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
56         counts_drive = counts_drive_left + counts_drive_right;
57         counts_drive /= 2;
58         //if(millis() - time > 1000) break;
59     }
60     set_side_speed(BOTH, 0);
61     delay(1000);
62     open_claw();
63     delay(1000);
64     deinitSlew();
65 }
```

Here is the call graph for this function:



7.12.2.2 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.

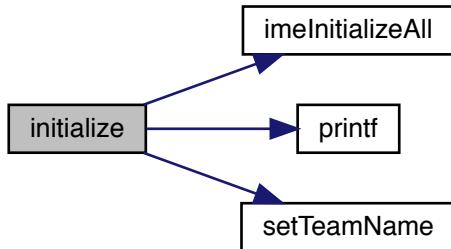
This function must exit relatively promptly, or the `operatorControl()` and `autonomous()` tasks will not start. An autonomous mode selection menu like the `pre_auton()` in other environments can be implemented in this task if desired.

Definition at line 47 of file init.c.

References `imlInitializeAll()`, `printf()`, and `setTeamName()`.

```
47     {  
48     int c = imeInitializeAll();  
49     setTeamName("9228A");  
50     printf("Counts : %d\\n", c);  
51 }
```

Here is the call graph for this function:



7.12.2.3 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 ([uartOpen\(\)](#)) but cannot set up an LCD ([lcdInit\(\)](#)).

Definition at line 30 of file init.c.

References [watchdogInit\(\)](#).

```
30
31     watchdogInit ();
32 }
```

Here is the call graph for this function:



7.12.2.4 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.

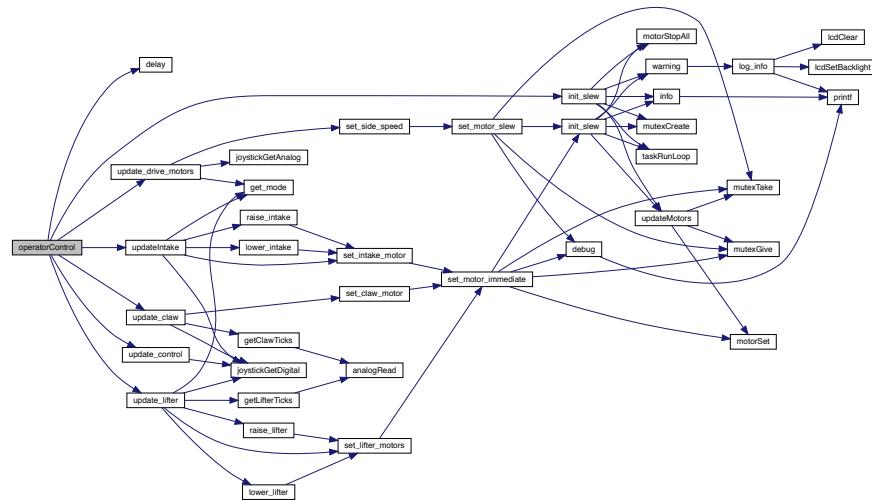
This task should never exit; it should end with some kind of infinite loop, even if empty.

Definition at line 40 of file opcontrol.c.

References `delay()`, `init_slew()`, `update_claw()`, `update_control()`, `update_drive_motors()`, `update_lifter()`, and `updateIntake()`.

```
40
41     init_slew();
42     delay(10);
43     while (1) {
44         update_drive_motors();
45         update_lifter();
46         update_claw();
47         updateIntake();
48         update_control();
49         delay(25);
50     }
51 }
```

Here is the call graph for this function:

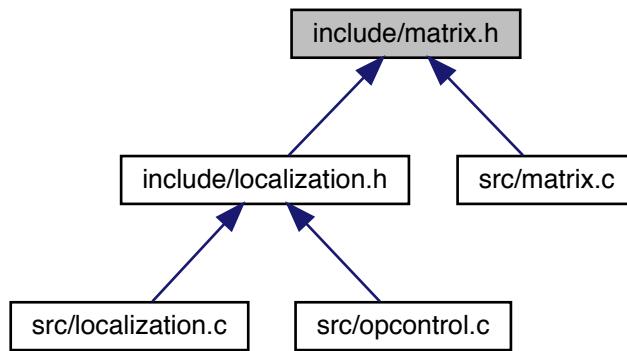


7.13 include/matrix.h File Reference

Various Matrix operations.

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

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



Data Structures

- struct [_matrix](#)

TypeDefs

- [typedef struct _matrix matrix](#)

Functions

- void [assert](#) (int assertion, char *message)
- [matrix * copyMatrix \(matrix *m\)](#)
- [matrix * covarianceMatrix \(matrix *m\)](#)
- [matrix * dotDiagonalMatrix \(matrix *a, matrix *b\)](#)
- [matrix * dotProductMatrix \(matrix *a, matrix *b\)](#)
- [matrix * eyeMatrix \(int n\)](#)
- void [freeMatrix \(matrix *m\)](#)
- [matrix * L2_distance \(matrix *a, matrix *b\)](#)
- [matrix * makeMatrix \(int width, int height\)](#)
- [matrix * meanMatrix \(matrix *m\)](#)
- [matrix * multiplyMatrix \(matrix *a, matrix *b\)](#)
- void [printMatrix \(matrix *m\)](#)
- [matrix * readMatrix \(char *filename\)](#)
- void [rowSwap \(matrix *a, int p, int q\)](#)
- [matrix * scaleMatrix \(matrix *m, double value\)](#)
- double [traceMatrix \(matrix *m\)](#)
- [matrix * transposeMatrix \(matrix *m\)](#)
- void [writeMatrix \(matrix *m, char *filename\)](#)

7.13.1 Detailed Description

Various Matrix operations.

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

7.13.2 Typedef Documentation

7.13.2.1 matrix

```
typedef struct __matrix matrix
```

7.13.3 Function Documentation

7.13.3.1 assert()

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

assert 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 meant to act like Python's assert keyword.

Definition at line 13 of file matrix.c.

References `fprintf()`.

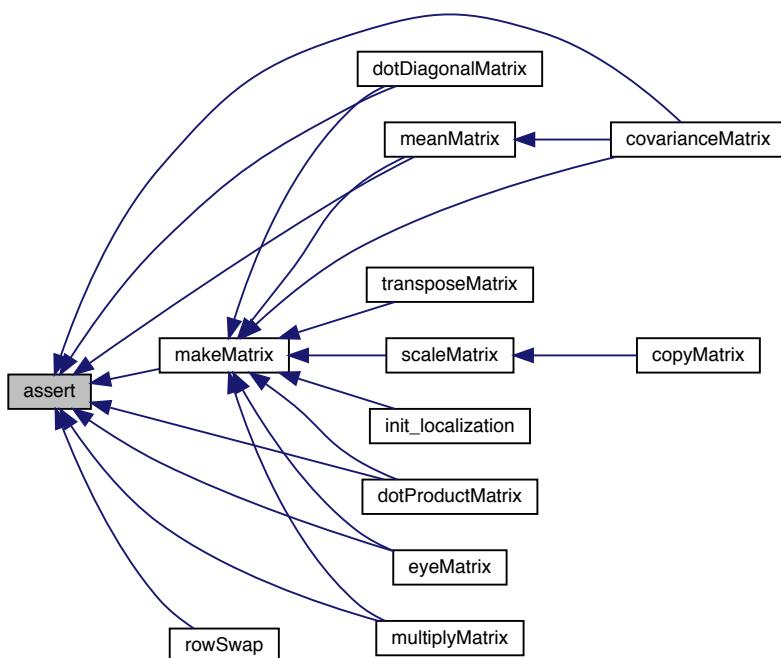
Referenced by `covarianceMatrix()`, `dotDiagonalMatrix()`, `dotProductMatrix()`, `eyeMatrix()`, `makeMatrix()`, `meanMatrix()`, `multiplyMatrix()`, and `rowSwap()`.

```
13
14     if (assertion == 0) {
15         fprintf(stderr, "%s\n", message);
16         exit(1);
17     }
18 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.2 copyMatrix()

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

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

Definition at line 47 of file matrix.c.

References scaleMatrix().

```

47
48     return scaleMatrix(m, 1);
49 }

```

Here is the call graph for this function:



7.13.3.3 covarianceMatrix()

```

matrix* covarianceMatrix (
    matrix * m )

```

covarianceMatrix Given an "m rows by n columns" matrix, it returns a matrix with n row and n columns, where each element represents covariance of 2 columns.

Definition at line 164 of file matrix.c.

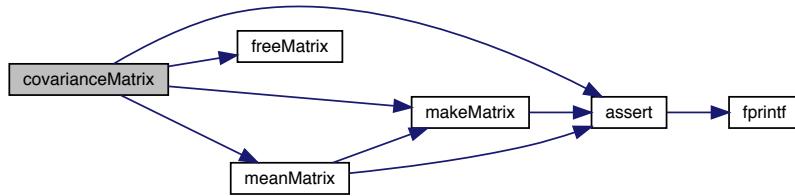
References assert(), _matrix::data, freeMatrix(), _matrix::height, makeMatrix(), meanMatrix(), and _matrix::width.

```

164
165     int i, j, k, L = 0;
166     matrix* out;
167     matrix* mean;
168     double* ptrA;
169     double* ptrB;
170     double* ptrOut;
171
172     assert(m->height > 1, "Height of matrix cannot be zero or one.");
173
174     mean = meanMatrix(m);
175     out = makeMatrix(m->width, m->width);
176     ptrOut = out->data;
177
178     for (i = 0; i < m->width; i++) {
179         for (j = 0; j < m->width; j++) {
180             ptrA = &m->data[i];
181             ptrB = &m->data[j];
182             *ptrOut = 0.0;
183             for (k = 0; k < m->height; k++) {
184                 *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
185                 ptrA += m->width;
186                 ptrB += m->width;
187             }
188             *ptrOut /= m->height - 1;
189             ptrOut++;
190         }
191     }
192
193     freeMatrix(mean);
194     return out;
195 }

```

Here is the call graph for this function:



7.13.3.4 dotDiagonalMatrix()

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

matrixDotDiagonal 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 parameter is NULL, it is assumed that we are performing a cross product with itself.

Definition at line 367 of file matrix.c.

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

```

367
368     matrix* out;
369     double* ptrOut;
370     double* ptrA;
371     double* ptrB;
372     int i, j;
373
374     if (b != NULL) {
375         assert(a->width == b->width && a->height == b->
376 height, "Matrices must be of the same dimensionality.");
377     }
378
379     // Are we computing the sum of squares of the same matrix?
380     if (a == b || b == NULL) {
381         b = a; // May not appear safe, but we can do this without risk of losing b.
382     }
383
384     out = makeMatrix(1, a->height);
385     ptrOut = out->data;
386     ptrA = a->data;
387     ptrB = b->data;
388
389     for (i = 0; i < a->height; i++) {
390         *ptrOut = 0;
391         for (j = 0; j < a->width; j++) {
392             *ptrOut += *ptrA * *ptrB;
393             ptrA++;
394             ptrB++;
395         }
396         ptrOut++;
397     }
398     return out;
399 }
```

Here is the call graph for this function:



7.13.3.5 dotProductMatrix()

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

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

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

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

Definition at line 318 of file matrix.c.

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

```

318
319     matrix* out;
320     double* ptrOut;
321     double* ptrA;
322     double* ptrB;
323     int i, j, k;
324
325     if (b != NULL) {
326         assert(a->width == b->width, "Matrices must be of the same dimensionality.");
327     }
328
329     // Are we computing the sum of squares of the same matrix?
330     if (a == b || b == NULL) {
331         b = a; // May not appear safe, but we can do this without risk of losing b.
332     }
333
334     out = makeMatrix(b->height, a->height);
335     ptrOut = out->data;
336
337     for (i = 0; i < a->height; i++) {
338         ptrB = b->data;
339
340         for (j = 0; j < b->height; j++) {
341             ptrA = &a->data[i * a->width];
342
343             *ptrOut = 0;
344             for (k = 0; k < a->width; k++) {
345                 *ptrOut += *ptrA * *ptrB;
346                 ptrA++;
347                 ptrB++;
348             }
349             ptrOut++;
350         }
351     }
352
353     return out;
354 }
```

Here is the call graph for this function:



7.13.3.6 eyeMatrix()

```
matrix* eyeMatrix (
    int n )
```

`eyeMatrix` Returns an identity matrix of size n by n , where n is the input parameter.

Definition at line 90 of file `matrix.c`.

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

```

90
91     int i;
92     matrix *out;
93     double* ptr;
94
95     assert(n > 0, "Identity matrix must have value greater than zero.");
96
97     out = makeMatrix(n, n);
98     ptr = out->data;
99     for (i = 0; i < n; i++) {
100         *ptr = 1.0;
101         ptr += n + 1;
102     }
103
104     return out;
105 }
```

Here is the call graph for this function:



7.13.3.7 freeMatrix()

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

freeMatrix Frees the resources of a matrix

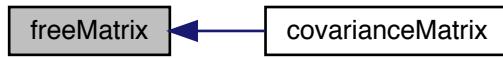
Definition at line 55 of file matrix.c.

References `_matrix::data`.

Referenced by `covarianceMatrix()`.

```
55     if (m != NULL) {
56         if (m->data != NULL) {
57             free(m->data);
58             m->data = NULL;
59         }
60     }
61     free(m);
62     m = NULL;
63 }
64
65 return;
66 }
```

Here is the caller graph for this function:



7.13.3.8 L2_distance()

```
matrix* L2_distance (
    matrix * a,
    matrix * b )
```

7.13.3.9 makeMatrix()

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

makeMatrix Makes a matrix with a width and height parameters.

Definition at line 24 of file matrix.c.

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

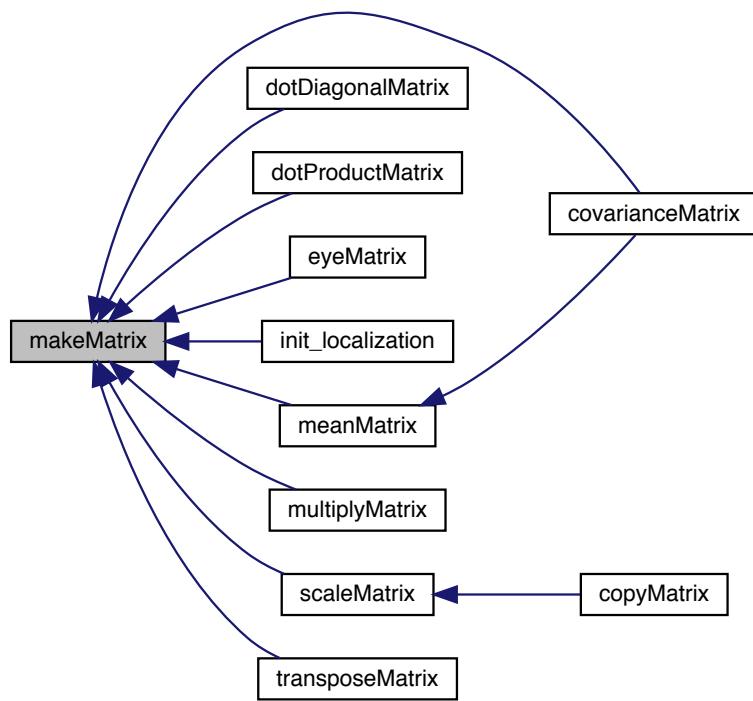
Referenced by covarianceMatrix(), dotDiagonalMatrix(), dotProductMatrix(), eyeMatrix(), init_localization(), mean←Matrix(), multiplyMatrix(), scaleMatrix(), and transposeMatrix().

```
24
25     matrix* out;
26     assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
27     out = (matrix*) malloc(sizeof(matrix));
28
29     assert(out != NULL, "Out of memory.");
30
31     out->width = width;
32     out->height = height;
33     out->data = (double*) malloc(sizeof(double) * width * height);
34
35     assert(out->data != NULL, "Out of memory.");
36
37     memset(out->data, 0.0, width * height * sizeof(double));
38
39     return out;
40 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.10 meanMatrix()

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

`meanMatrix` Given an "m rows by n columns" matrix, it returns a matrix with 1 row and n columns, where each element represents the mean of that full column.

Definition at line 138 of file matrix.c.

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

Referenced by `covarianceMatrix()`.

```

138
139     int i, j;
140     double* ptr;
141     matrix* out;
142
143     assert(m->height > 0, "Height of matrix cannot be zero.");
144
145     out = makeMatrix(m->width, 1);
146
147     for (i = 0; i < m->width; i++) {
148         out->data[i] = 0.0;
149         ptr = &m->data[i];

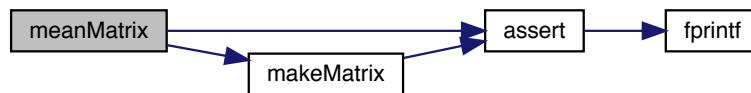
```

```

150     for (j = 0; j < m->height; j++) {
151         out->data[i] += *ptr;
152         ptr += out->width;
153     }
154     out->data[i] /= (double) m->height;
155 }
156 return out;
157 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.11 multiplyMatrix()

```

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

```

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

Definition at line 223 of file matrix.c.

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

```

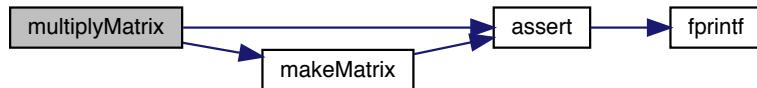
223
224     int i, j, k;
225     matrix* out;
226     double* ptrOut;
227     double* ptrA;
228     double* ptrB;
229
230     assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");
231
232     out = makeMatrix(b->width, a->height);
233     ptrOut = out->data;

```

```

234     for (i = 0; i < a->height; i++) {
235         for (j = 0; j < b->width; j++) {
236             ptrA = &a->data[ i * a->width ];
237             ptrB = &b->data[ j ];
238
239             *ptrOut = 0;
240             for (k = 0; k < a->width; k++) {
241                 *ptrOut += *ptrA * *ptrB;
242                 ptrA++;
243                 ptrB += b->width;
244             }
245             ptrOut++;
246         }
247     }
248     return out;
249 }
250
251 }
```

Here is the call graph for this function:



7.13.3.12 printMatrix()

```

void printMatrix (
    matrix * m )

```

`printMatrix` Prints a matrix. Great for debugging.

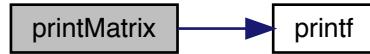
Definition at line 72 of file `matrix.c`.

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

```

72
73     int i, j;
74     double* ptr = m->data;
75     printf("%d %d\n", m->width, m->height);
76     for (i = 0; i < m->height; i++) {
77         for (j = 0; j < m->width; j++) {
78             printf(" %9.6f", *(ptr++));
79         }
80         printf("\n");
81     }
82     return;
83 }
```

Here is the call graph for this function:



7.13.3.13 readMatrix()

```
matrix* readMatrix (
    char * filename )
```

7.13.3.14 rowSwap()

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

rowSwap 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.

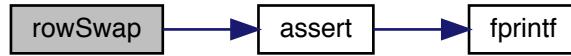
Definition at line 278 of file matrix.c.

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

```

278
279     int i;
280     double temp;
281     double* pRow;
282     double* qRow;
283
284     assert(a->height > 2, "Matrix must have at least two rows to swap.");
285     assert(p < a->height && q < a->height, "Values p and q must be less than the height of the
286     matrix.");
287
288     // If p and q are equal, do nothing.
289     if (p == q) {
290         return;
291     }
292
293     pRow = a->data + (p * a->width);
294     qRow = a->data + (q * a->width);
295
296     // Swap!
297     for (i = 0; i < a->width; i++) {
298         temp = *pRow;
299         *pRow = *qRow;
300         *qRow = temp;
301         pRow++;
302         qRow++;
303     }
304
305 }
```

Here is the call graph for this function:



7.13.3.15 scaleMatrix()

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

scaleMatrix Given a matrix and a double value, this returns a new matrix where each element in the input matrix is multiplied by the double value

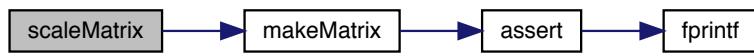
Definition at line 259 of file matrix.c.

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

Referenced by `copyMatrix()`.

```
259
260     int i, elements = m->width * m->height;
261     matrix* out = makeMatrix(m->width, m->height);
262     double* ptrM = m->data;
263     double* ptrOut = out->data;
264
265     for (i = 0; i < elements; i++) {
266         *(ptrOut++) = *(ptrM++) * value;
267     }
268
269     return out;
270 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.13.3.16 traceMatrix()

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

traceMatrix Given an "m rows by n columns" matrix, it returns the sum of the elements along the diagonal. This is known as the matrix 'trace'.

Definition at line 112 of file matrix.c.

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

```

112
113     int i;
114     int size;
115     double* ptr = m->data;
116     double sum = 0.0;
117
118     if (m->height < m->width) {
119         size = m->height;
120     }
121     else {
122         size = m->width;
123     }
124
125     for (i = 0; i < size; i++) {
126         sum += *ptr;
127         ptr += m->width + 1;
128     }
129
130     return sum;
131 }
```

7.13.3.17 transposeMatrix()

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

transposeMatrix Given an matrix, returns the transpose.

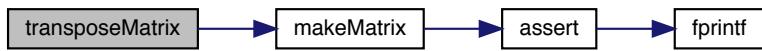
Definition at line 201 of file matrix.c.

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

```

201     matrix* out = makeMatrix(m->height, m->width);
202     double* ptrOut;
203     double* ptrM = m->data;
204     int i, j;
205
206     for (i = 0; i < m->height; i++) {
207         ptrOut = &out->data[i];
208         for (j = 0; j < m->width; j++) {
209             *ptrOut = *ptrM;
210             ptrM++;
211             ptrOut += out->width;
212         }
213     }
214
215
216     return out;
217 }
```

Here is the call graph for this function:



7.13.3.18 writeMatrix()

```

void writeMatrix (
    matrix * m,
    char * filename )
```

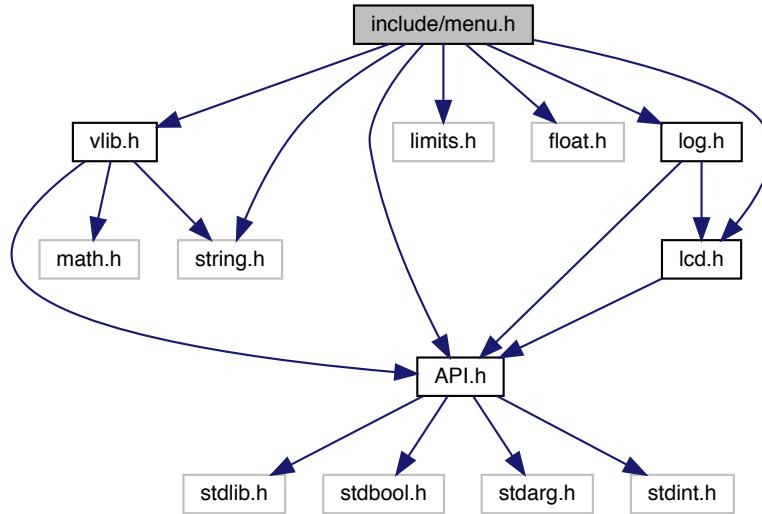
7.14 include/menu.h File Reference

Contains menu functionality and abstraction.

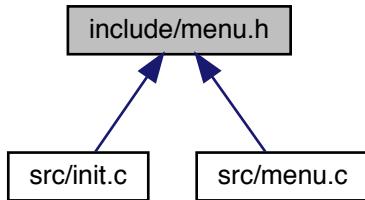
```

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

```
#include "log.h"
Include dependency graph for menu.h:
```



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



Data Structures

- struct [menu_t](#)

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

TypeDefs

- typedef struct [menu_t](#) `menu_t`

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

Enumerations

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

Represents the different types of menus.

Functions

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

Static function that calculates the string from menu.

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

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

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

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

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

Displays a menu context, but does not display. 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, unsigned int nums, const char *prompt, char *options,...)

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

7.14.1 Detailed Description

Contains menu functionality and abstraction.

Author

Chris Jerrett

Date

9/9/2017

7.14.2 Typedef Documentation

7.14.2.1 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](#)
[menu_t](#)
[create_menu](#)
[init_menu](#)
[display_menu](#)
[menu_type](#)
[denint_menu](#)

7.14.3 Enumeration Type Documentation

7.14.3.1 menu_type

```
enum menu_type
```

Represents the different types of menus.

Author

Chris Jerrett

Date

9/8/17

See also

[menu.h](#)
[menu_t](#)
[create_menu](#)
[init_menu](#)
[display_menu](#)
[menu_type](#)

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 29 of file menu.h.

```
29
36     INT_TYPE,
43     FLOAT_TYPE,
49     STRING_TYPE
50 };
```

7.14.4 Function Documentation

7.14.4.1 calculate_current_display()

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

Static function that calculates the string from menu.

Parameters

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

Author

Chris Jerrett

Date

9/8/17

7.14.4.2 create_menu()

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

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

Author

Chris Jerrett

Date

9/8/17

7.14.4.3 denint_menu()

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

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

Parameters

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

See also

menu

Author

Chris Jerrett

Date

9/8/17

Definition at line 101 of file menu.c.

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

```
101
102     free(menu->prompt);
103     if(menu->options != NULL) free(menu->options);
104     free(menu);
105 }
```

7.14.4.4 display_menu()

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

Displays a menu context, but does not display. *Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.*

Parameters

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

See also

[menu_type](#)

Author

Chris Jerrett

Date

9/8/17

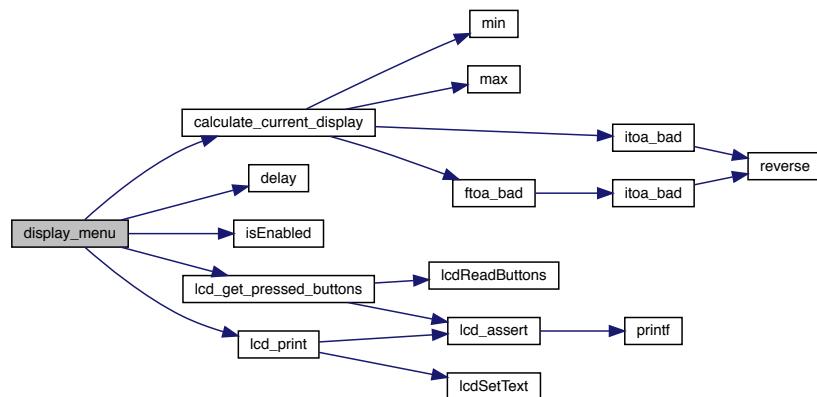
Definition at line 83 of file menu.c.

References `calculate_current_display()`, `menu_t::current`, `delay()`, `isEnabled()`, `lcd_get_pressed_buttons()`, `lcd←print()`, `PRESSED`, `menu_t::prompt`, `RELEASED`, and `TOP_ROW`.

```

83     {
84     lcd_print(TOP_ROW, menu->prompt);
85     //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
86     while(lcd_get_pressed_buttons().middle == RELEASED && !
87         isEnabled()) {
88         char val[16];
89         calculate_current_display(val, menu);
90         if(lcd_get_pressed_buttons().right == PRESSED) {
91             menu->current += 1;
92         }
93         if(lcd_get_pressed_buttons().left == PRESSED) {
94             menu->current -= 1;
95         }
96         delay(500);
97     }
98     return menu->current;
99 }
```

Here is the call graph for this function:



7.14.4.5 init_menu_float()

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

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

Parameters

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

See also

[menu_type](#)

Parameters

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

Author

Chris Jerrett

Date

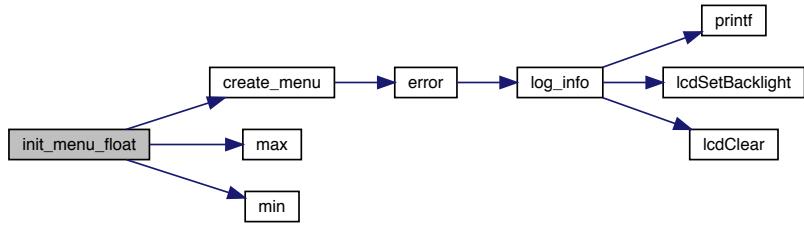
9/8/17

Definition at line 48 of file menu.c.

References [create_menu\(\)](#), [max\(\)](#), [menu_t::max_f](#), [min\(\)](#), [menu_t::min_f](#), and [menu_t::step_f](#).

```
48
49     menu_t* menu = create\_menu(type, prompt);
50     menu->min\_f = min;
51     menu->max\_f = max;
52     menu->step\_f = step;
53     return menu;
54 }
```

Here is the call graph for this function:



7.14.4.6 init_menu_int()

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

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

Parameters

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

See also

[menu_type](#)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

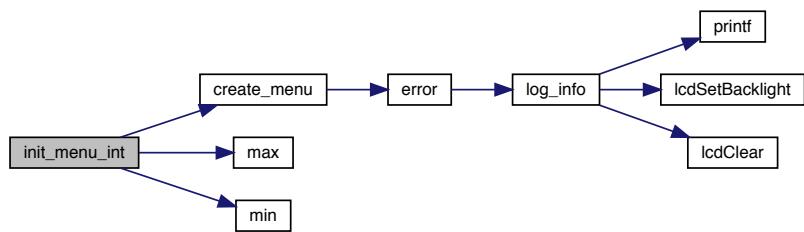
Definition at line 40 of file menu.c.

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

```

40
41     menu_t* menu = create\_menu(type, prompt);
42     menu->min = min;
43     menu->max = max;
44     menu->step = step;
45     return menu;
46 }
```

Here is the call graph for this function:



7.14.4.7 `init_menu_var()`

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

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

Parameters

<code>type</code>	the type of menu
-------------------	------------------

See also

[menu_type](#)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

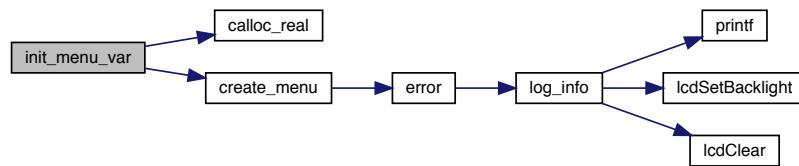
Definition at line 26 of file menu.c.

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

```

26
27 menu_t* menu = create_menu(type, prompt);
28 va_list values;
29 char **options_array = (char**)calloc_real(sizeof(char*), nums);
30 va_start(values, options);
31 for(unsigned int i = 0; i < nums; i++){
32     options_array[i] = va_arg(values, char*);
33 }
34 va_end(values);
35 menu->options = options_array;
36 menu->length = nums;
37 return menu;
38 }
```

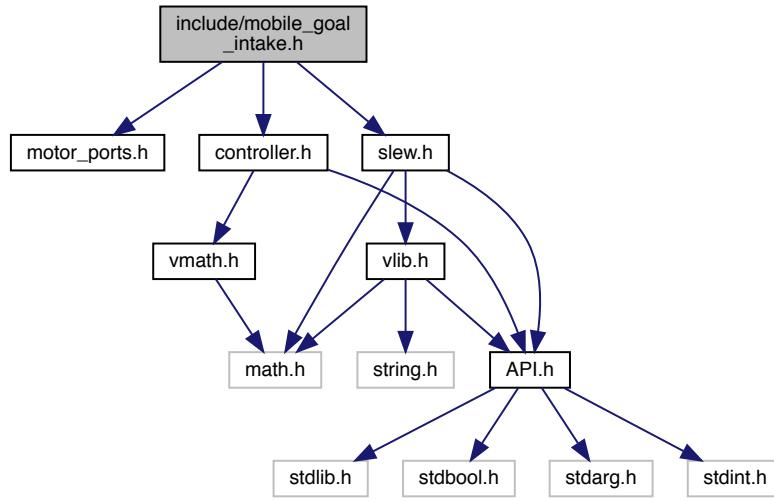
Here is the call graph for this function:



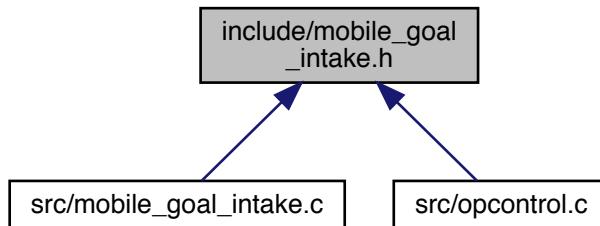
7.15 include/mobile_goal_intake.h File Reference

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

Include dependency graph for mobile_goal_intake.h:



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



Functions

- void [updateIntake \(\)](#)

7.15.1 Function Documentation

7.15.1.1 updateIntake()

```
void updateIntake( )
```

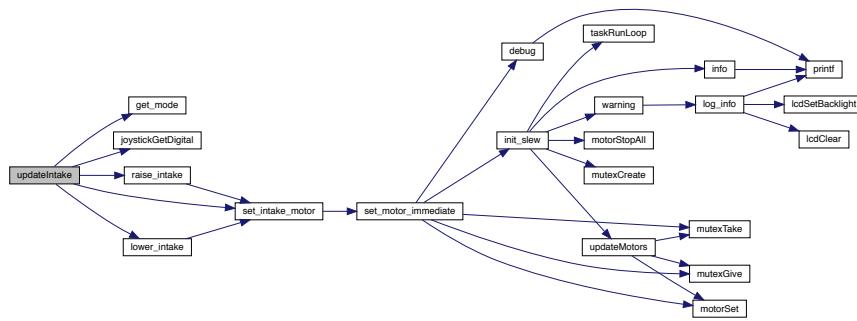
Definition at line 16 of file mobile_goal_intake.c.

References get_mode(), JOY_DOWN, JOY_UP, joystickGetDigital(), lower_intake(), MAIN_CONTROLLER_MODE, MASTER, PARTNER, PARTNER_CONTROLLER_MODE, raise_intake(), and set_intake_motor().

Referenced by operatorControl().

```
16     {
17     if(joystickGetDigital(MASTER, 7, JOY_UP) && (
18         get_mode() == MAIN_CONTROLLER_MODE)
19     || joystickGetDigital(PARTNER, 6, JOY_UP) &&
20         get_mode() == PARTNER_CONTROLLER_MODE) {
21         raise_intake();
22     }
23     else if(joystickGetDigital(MASTER, 7, JOY_DOWN) && (
24         get_mode() == MAIN_CONTROLLER_MODE)
25     || joystickGetDigital(PARTNER, 6, JOY_DOWN) &&
26         get_mode() == PARTNER_CONTROLLER_MODE) {
27         lower_intake();
28     }
29     else set_intake_motor(0);
30 }
```

Here is the call graph for this function:

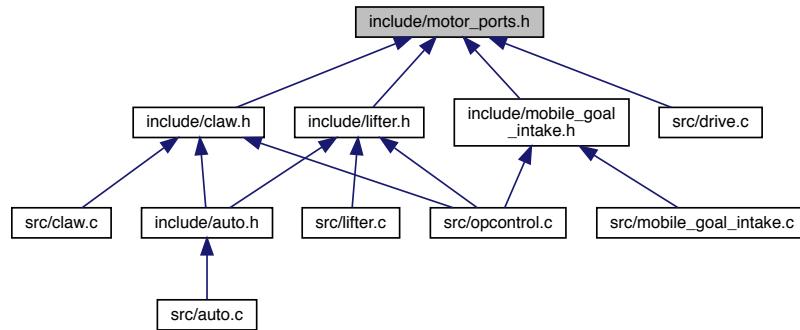


Here is the caller graph for this function:



7.16 include/motor_ports.h File Reference

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



Macros

- #define _MOTOR_PORTS_H_
- #define CLAW_MOTOR 10
- #define INTAKE_MOTOR 1
- #define MAX_SPEED 127
- #define MIN_SPEED -127
- #define MOTOR_BACK_LEFT 5

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

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

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

Front right drive motor of robot base.
- #define MOTOR_LIFT_BOTTOM_LEFT 9
- #define MOTOR_LIFT_BOTTOM_RIGHT 8
- #define MOTOR_LIFT_TOP_LEFT 9
- #define MOTOR_LIFT_TOP_RIGHT 8
- #define MOTOR_MIDDLE_LEFT 6

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

Middle right drive motor of robot base.

7.16.1 Macro Definition Documentation

7.16.1.1 _MOTOR_PORTS_H_

```
#define _MOTOR_PORTS_H_
```

Definition at line 2 of file motor_ports.h.

7.16.1.2 CLAW_MOTOR

```
#define CLAW_MOTOR 10
```

Definition at line 53 of file motor_ports.h.

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

7.16.1.3 INTAKE_MOTOR

```
#define INTAKE_MOTOR 1
```

Definition at line 54 of file motor_ports.h.

Referenced by set_intake_motor().

7.16.1.4 MAX_SPEED

```
#define MAX_SPEED 127
```

Definition at line 4 of file motor_ports.h.

Referenced by raise_lifter().

7.16.1.5 MIN_SPEED

```
#define MIN_SPEED -127
```

Definition at line 5 of file motor_ports.h.

Referenced by lower_lifter().

7.16.1.6 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 46 of file motor_ports.h.

Referenced by set_side_speed().

7.16.1.7 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 40 of file motor_ports.h.

Referenced by set_side_speed().

7.16.1.8 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 19 of file motor_ports.h.

Referenced by set_side_speed().

7.16.1.9 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 12 of file motor_ports.h.

Referenced by `set_side_speed()`.

7.16.1.10 MOTOR_LIFT_BOTTOM_LEFT

```
#define MOTOR_LIFT_BOTTOM_LEFT 9
```

Definition at line 49 of file motor_ports.h.

7.16.1.11 MOTOR_LIFT_BOTTOM_RIGHT

```
#define MOTOR_LIFT_BOTTOM_RIGHT 8
```

Definition at line 48 of file motor_ports.h.

7.16.1.12 MOTOR_LIFT_TOP_LEFT

```
#define MOTOR_LIFT_TOP_LEFT 9
```

Definition at line 51 of file motor_ports.h.

Referenced by `set_lifter_motors()`.

7.16.1.13 MOTOR_LIFT_TOP_RIGHT

```
#define MOTOR_LIFT_TOP_RIGHT 8
```

Definition at line 50 of file motor_ports.h.

Referenced by set_lifter_motors().

7.16.1.14 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 33 of file motor_ports.h.

Referenced by set_side_speed().

7.16.1.15 MOTOR_MIDDLE_RIGHT

```
#define MOTOR_MIDDLE_RIGHT 3
```

Middle right drive motor of robot base.

Author

Christian Desimone

Date

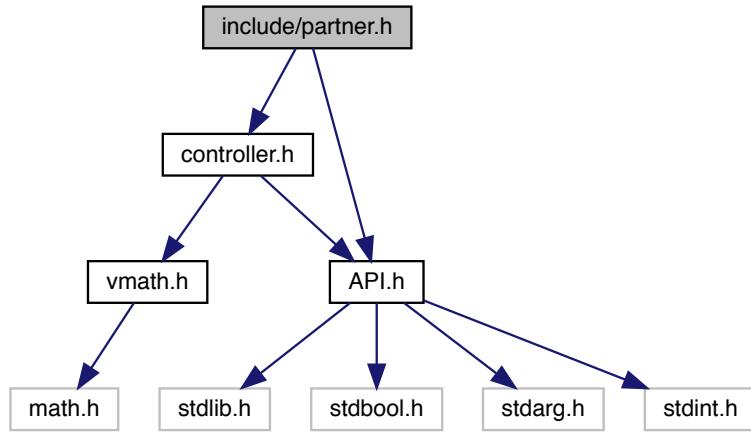
9/7/2017

Definition at line 26 of file motor_ports.h.

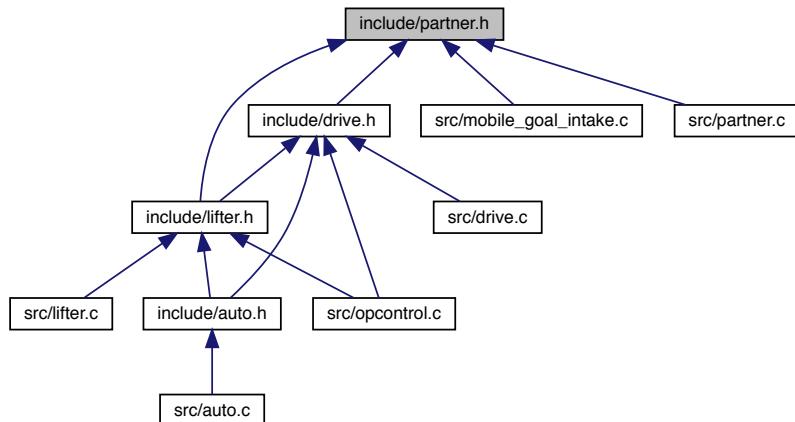
Referenced by set_side_speed().

7.17 include/partner.h File Reference

```
#include "controller.h"
#include "API.h"
Include dependency graph for partner.h:
```



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



Enumerations

- enum `CONTROL_MODE` { `MAIN_CONTROLLER_MODE`, `PARTNER_CONTROLLER_MODE` }

Functions

- enum `CONTROL_MODE` `get_mode ()`
- void `update_control ()`

7.17.1 Enumeration Type Documentation

7.17.1.1 CONTROL_MODE

enum `CONTROL_MODE`

Enumerator

<code>MAIN_CONTROLLER_MODE</code>	
<code>PARTNER_CONTROLLER_MODE</code>	

Definition at line 7 of file partner.h.

```
7           {
8   MAIN_CONTROLLER_MODE,
9   PARTNER_CONTROLLER_MODE
10 };
```

7.17.2 Function Documentation

7.17.2.1 get_mode()

enum `CONTROL_MODE` `get_mode ()`

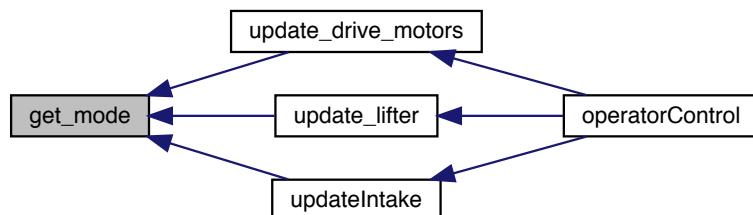
Definition at line 5 of file partner.c.

References mode.

Referenced by `update_drive_motors()`, `update_lifter()`, and `updateIntake()`.

```
5           {
6   return mode;
7 }
```

Here is the caller graph for this function:



7.17.2.2 update_control()

```
void update_control ( )
```

Definition at line 9 of file partner.c.

References JOY_LEFT, JOY_RIGHT, joystickGetDigital(), MAIN_CONTROLLER_MODE, mode, PARTNER, and PARTNER_CONTROLLER_MODE.

Referenced by operatorControl().

```
9      {
10     if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
11       mode = MAIN_CONTROLLER_MODE;
12     } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
13       mode = PARTNER_CONTROLLER_MODE;
14     }
15 }
```

Here is the call graph for this function:

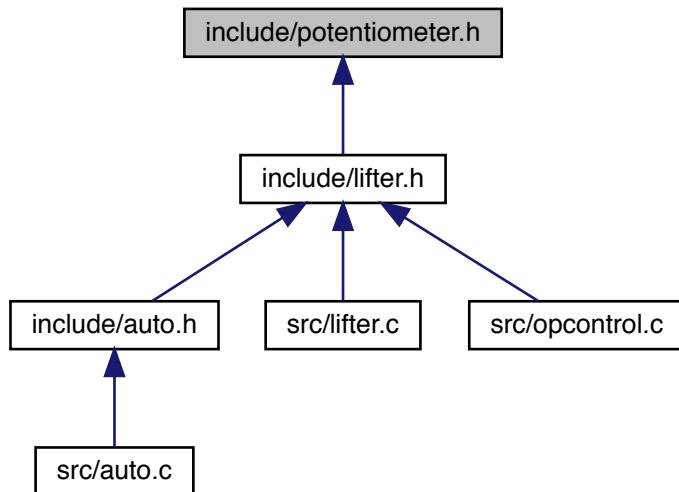


Here is the caller graph for this function:



7.18 include/potentiometer.h File Reference

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



Macros

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

7.18.1 Macro Definition Documentation

7.18.1.1 DEG_MAX

```
#define DEG_MAX 250.0
```

Definition at line 5 of file potentiometer.h.

Referenced by lifterPotentiometerToDegree().

7.18.1.2 TICK_MAX

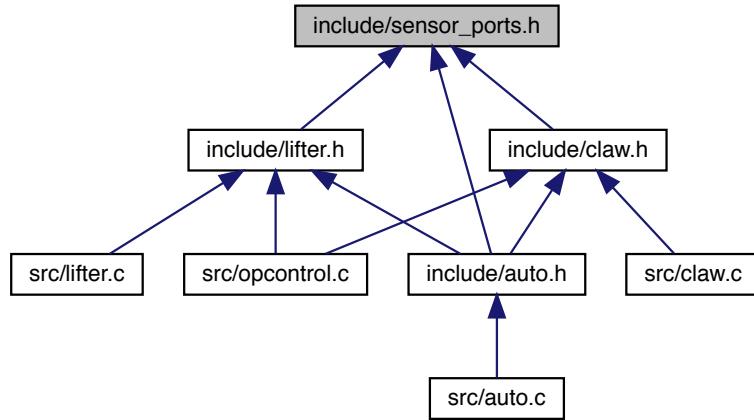
```
#define TICK_MAX 4095.0
```

Definition at line 4 of file potentiometer.h.

Referenced by lifterPotentiometerToDegree().

7.19 include/sensor_ports.h File Reference

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



Macros

- `#define CLAW_POT 1`
- `#define IME_FRONT_RIGHT 0`
Number of integrated motor encoders Used when checking to see if all imes are plugged in.
- `#define LIFTER 2`

7.19.1 Macro Definition Documentation

7.19.1.1 CLAW_POT

```
#define CLAW_POT 1
```

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

Referenced by `getClawTicks()`.

7.19.1.2 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](#)

Author

Christian Desimone

Date

9/7/2017

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

7.19.1.3 LIFTER

```
#define LIFTER 2
```

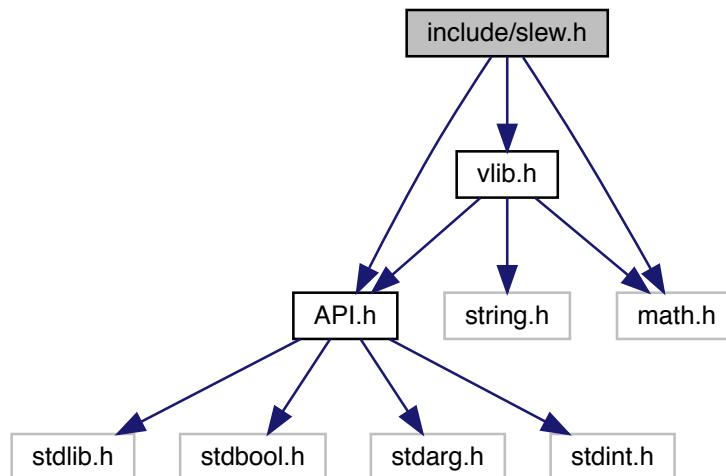
Definition at line 20 of file `sensor_ports.h`.

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

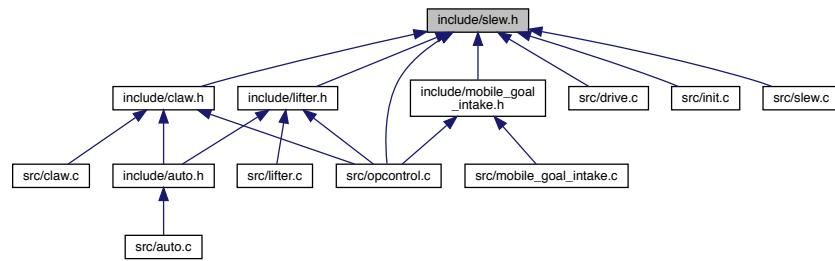
7.20 include/slew.h File Reference

Contains the slew rate controller wrapper for the motors.

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



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



Macros

- `#define MOTOR_PORTS 12`
The number of motor ports on the robot.
- `#define RAMP_PROPORTION 1`
proportion defining how quickly the motor should converge on the correct value. higher value leads to slower convergence
- `#define UPDATE_PERIOD_MS 25`
How frequently to update the motors, in milliseconds.

Functions

- `void deinitSlew ()`
Deinitializes the slew rate controller and frees memory.
- `void init_slew ()`
Initializes the slew rate controller.
- `void set_motor_immediate (int motor, int speed)`
- `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.

7.20.1 Detailed Description

Contains the slew rate controller wrapper for the motors.

Author

Chris Jerrett

Date

9/14/17

7.20.2 Macro Definition Documentation

7.20.2.1 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.

7.20.2.2 RAMP_PROPORTION

```
#define RAMP_PROPORTION 1
```

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

Author

Chris Jerrett

Date

9/14/17

Definition at line 34 of file slew.h.

7.20.2.3 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.

7.20.3 Function Documentation

7.20.3.1 deinitsllew()

```
void deinitsllew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line 43 of file slew.c.

References initialized, motors_curr_speeds, motors_set_speeds, slew, and taskDelete().

Referenced by autonomous().

```
43     {
44     taskDelete(slew);
45     memset(motors_set_speeds, 0, sizeof(int) * 10);
46     memset(motors_curr_speeds, 0, sizeof(int) * 10);
47     initialized = false;
48 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.3.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

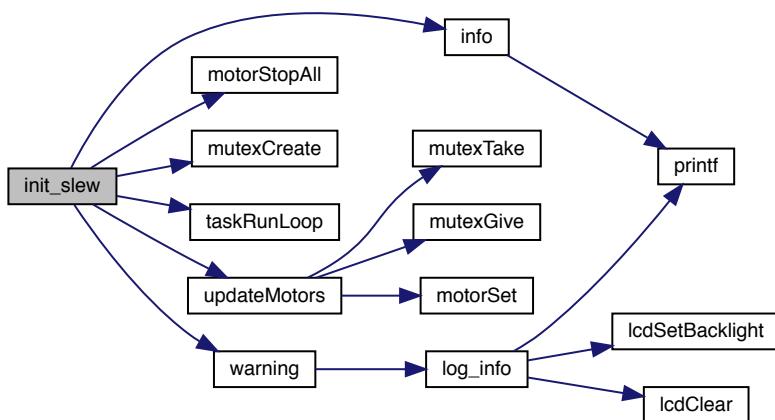
Definition at line 30 of file slew.c.

References info(), initialized, motors_curr_speeds, motors_set_speeds, motorStopAll(), mutexCreate(), slew, speeds_mutex, taskRunLoop(), updateMotors(), and warning().

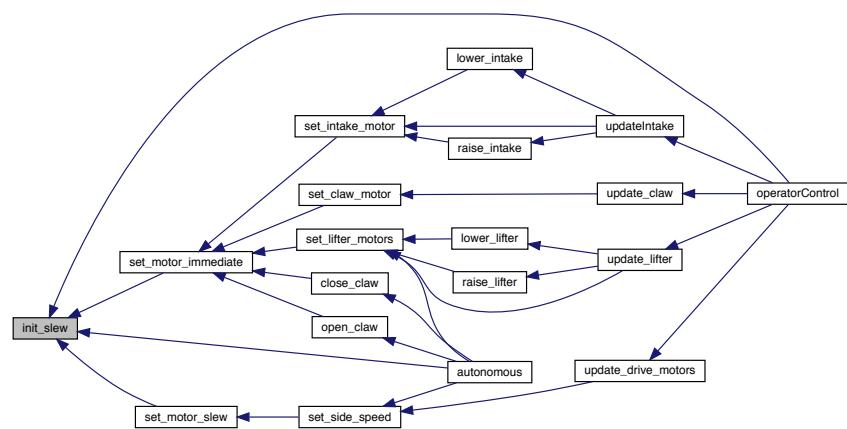
Referenced by autonomous(), operatorControl(), set_motor_immediate(), and set_motor_slew().

```
30      {
31      if(initialized) {
32          warning("Trying to init already init slew");
33      }
34      memset(motors_set_speeds, 0, sizeof(int) * 10);
35      memset(motors_curr_speeds, 0, sizeof(int) * 10);
36      motorStopAll();
37      info("Did Init Slew");
38      speeds_mutex = mutexCreate();
39      slew = taskRunLoop(updateMotors, 100);
40      initialized = true;
41 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.3.3 set_motor_immediate()

```
void set_motor_immediate (
```

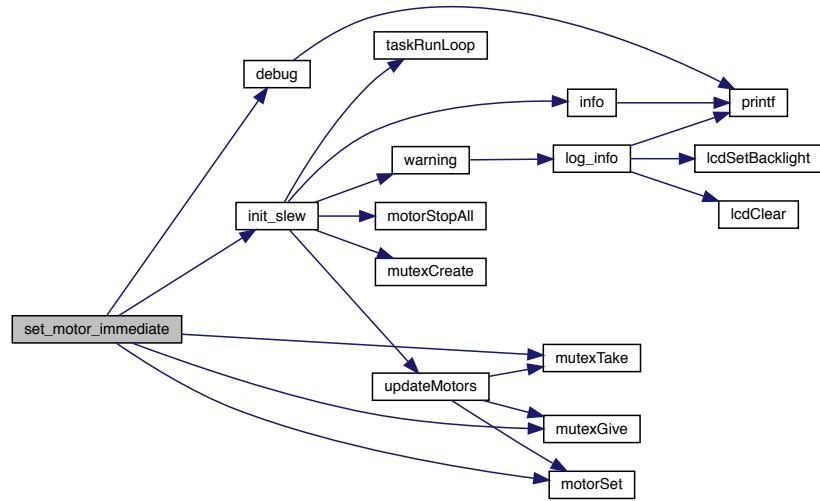
Definition at line 60 of file slew.c.

References debug(), init_slew(), initialized, motors_curr_speeds, motors_set_speeds, motorSet(), mutexGive(), mutexTake(), and speeds_mutex.

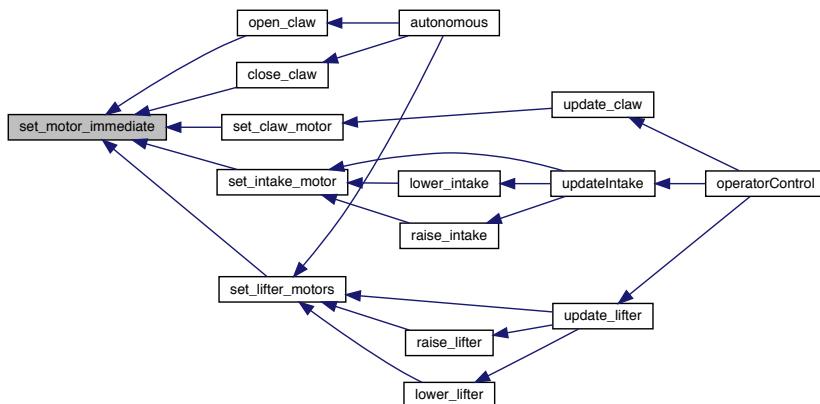
Referenced by `close_claw()`, `open_claw()`, `set_claw_motor()`, `set_intake_motor()`, and `set_lifter_motors()`.

```
60
61     if(!initialized) {
62         debug("Slew Not Initialized! Initializing");
63         init_slew();
64     }
65     motorSet(motor, speed);
66     mutexTake(speeds_mutex, 10);
67     motors_curr_speeds[motor-1] = speed;
68     motors_set_speeds[motor-1] = speed;
69     mutexGive(speeds_mutex);
70 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.3.4 set_motor_slew()

```

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

Sets motor speed wrapped inside the slew rate controller.

Parameters

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

Author

Chris Jerrett

Date

9/14/17

Definition at line 50 of file slew.c.

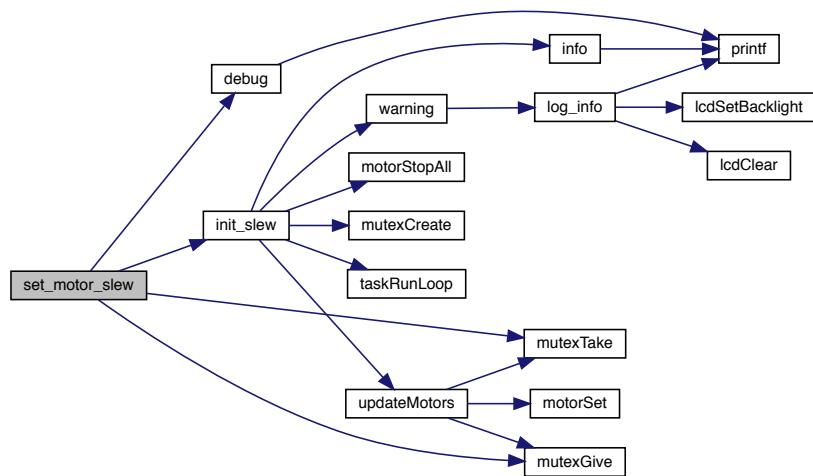
References debug(), init_slew(), initialized, motors_set_speeds, mutexGive(), mutexTake(), and speeds_mutex.

Referenced by set_side_speed().

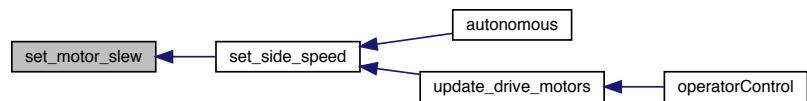
```

50
51 if(!initialized) {
52   debug("Slew Not Initialized! Initializing");
53   init_slew();
54 }
55 mutexTake(speeds_mutex, 10);
56 motors_set_speeds[motor-1] = speed;
57 mutexGive(speeds_mutex);
58 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.20.3.5 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

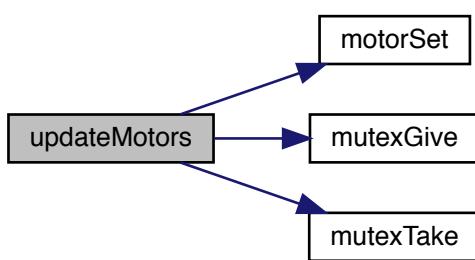
Definition at line 13 of file slew.c.

References motors_curr_speeds, motors_set_speeds, motorSet(), mutexGive(), mutexTake(), and speeds_mutex.

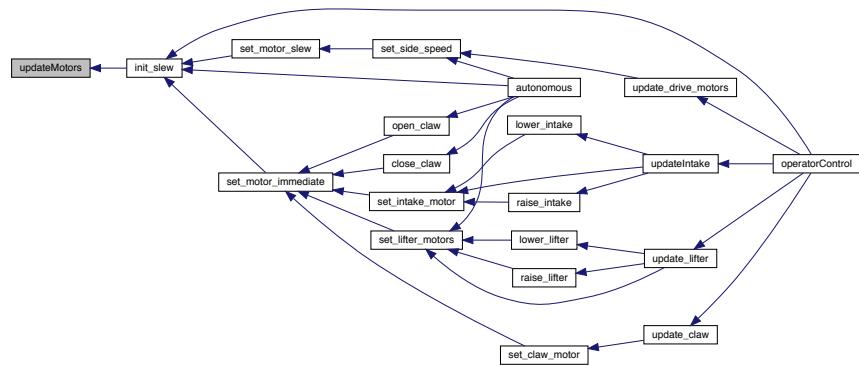
Referenced by init_slew().

```
13
14     {
15     //Take back half approach
16     //Not linear but equal to setSpeed(1-(1/2)^x)
17     for(unsigned int i = 0; i < 9; i++) {
18         if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
19         mutexTake(speeds_mutex, 10);
20         int set_speed = (motors_set_speeds[i]);
21         int curr_speed = motors_curr_speeds[i];
22         mutexGive(speeds_mutex);
23         int diff = set_speed - curr_speed;
24         int offset = diff;
25         int n = curr_speed + offset;
26         motors_curr_speeds[i] = n;
27         motorSet(i+1, n);
28     }
```

Here is the call graph for this function:

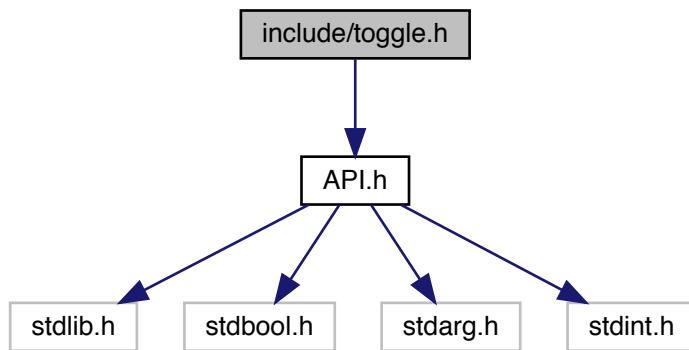


Here is the caller graph for this function:

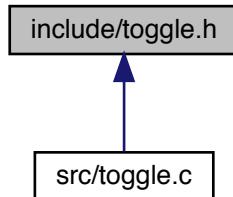


7.21 include/toggle.h File Reference

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



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



Enumerations

- enum `button_t` {

JOY1_5D = 0, JOY1_5U = 1, JOY1_6D = 2, JOY1_6U = 3,

JOY1_7U = 4, JOY1_7L = 5, JOY1_7R = 6, JOY1_7D = 7,

JOY1_8U = 8, JOY1_8L = 9, JOY1_8R = 10, JOY1_8D = 11,

JOY2_5D = 12, JOY2_5U = 13, JOY2_6D = 14, JOY2_6U = 15,

JOY2_7U = 16, JOY2_7L = 17, JOY2_7R = 18, JOY2_7D = 19,

JOY2_8U = 20, JOY2_8L = 21, JOY2_8R = 22, JOY2_8D = 23,

`LCD_LEFT` = 24, `LCD_CENT` = 25, `LCD_RIGHT` = 26 }

Functions

- bool `buttonGetState (button_t)`

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

Initializes the buttons.
- bool `buttonIsNewPress (button_t)`

Detects if button is a new press from most recent check by comparing previous value to current value.

7.21.1 Enumeration Type Documentation

7.21.1.1 `button_t`

`enum button_t`

Renames the input channels

Enumerator

<code>JOY1_5D</code>	
<code>JOY1_5U</code>	

Enumerator

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 20 of file toggle.h.

```

20      {
21      JOY1_5D = 0,
22      JOY1_5U = 1,
23      JOY1_6D = 2,
24      JOY1_6U = 3,
25      JOY1_7U = 4,
26      JOY1_7L = 5,
27      JOY1_7R = 6,
28      JOY1_7D = 7,
29      JOY1_8U = 8,
30      JOY1_8L = 9,
31      JOY1_8R = 10,
32      JOY1_8D = 11,
33
34      JOY2_5D = 12,
35      JOY2_5U = 13,
36      JOY2_6D = 14,
37      JOY2_6U = 15,
38      JOY2_7U = 16,
39      JOY2_7L = 17,
40      JOY2_7R = 18,
41      JOY2_7D = 19,
42      JOY2_8U = 20,
43      JOY2_8L = 21,
44      JOY2_8R = 22,
45      JOY2_8D = 23,
46
47      LCD_LEFT = 24,
48      LCD_CENT = 25,
49      LCD_RIGHT = 26
50 } button_t;

```

7.21.2 Function Documentation

7.21.2.1 buttonGetState()

```
bool buttonGetState (
    button_t    )
```

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

Parameters

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

Returns

true (pressed) or false (not pressed)

Definition at line 25 of file toggle.c.

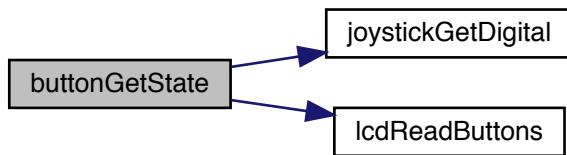
References JOY_DOWN, JOY_LEFT, JOY_RIGHT, JOY_UP, joystickGetDigital(), LCD_BTN_CENTER, LCD_BT↔N_LEFT, LCD_BTN_RIGHT, LCD_CENT, LCD_LEFT, LCD_RIGHT, lcdReadButtons(), and uart1.

Referenced by buttonIsNewPress().

```
25
26     bool currentButton = false;
27
28     // Determine how to get the current button value (from what function) and where it
29     // is, then get it.
30     if (button < LCD_LEFT) {
31         // button is a joystick button
32         unsigned char joystick;
33         unsigned char buttonGroup;
34         unsigned char buttonLocation;
35
36         button_t newButton;
37         if (button <= 11) {
38             // button is on joystick 1
39             joystick = 1;
40             newButton = button;
41         }
42         else {
43             // button is on joystick 2
44             joystick = 2;
45             // shift button down to joystick 1 buttons in order to
46             // detect which button on joystick is queried
47             newButton = (button_t)(button - 12);
48         }
49
50         switch (newButton) {
51             case 0:
52                 buttonGroup = 5;
53                 buttonLocation = JOY_DOWN;
54                 break;
55             case 1:
56                 buttonGroup = 5;
57                 buttonLocation = JOY_UP;
58                 break;
59             case 2:
60                 buttonGroup = 6;
61                 buttonLocation = JOY_DOWN;
62                 break;
63             case 3:
64                 buttonGroup = 6;
65                 buttonLocation = JOY_UP;
```

```
66         break;
67     case 4:
68         buttonGroup = 7;
69         buttonLocation = JOY_UP;
70         break;
71     case 5:
72         buttonGroup = 7;
73         buttonLocation = JOY_LEFT;
74         break;
75     case 6:
76         buttonGroup = 7;
77         buttonLocation = JOY_RIGHT;
78         break;
79     case 7:
80         buttonGroup = 7;
81         buttonLocation = JOY_DOWN;
82         break;
83     case 8:
84         buttonGroup = 8;
85         buttonLocation = JOY_UP;
86         break;
87     case 9:
88         buttonGroup = 8;
89         buttonLocation = JOY_LEFT;
90         break;
91     case 10:
92         buttonGroup = 8;
93         buttonLocation = JOY_RIGHT;
94         break;
95     case 11:
96         buttonGroup = 8;
97         buttonLocation = JOY_DOWN;
98         break;
99     default:
100         break;
101     }
102     currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
103 }
104 else {
105     // button is on LCD
106     if (button == LCD_LEFT)
107         currentButton = (lcdReadButtons(uart1) ==
LCD_BTN_LEFT);
108     if (button == LCD_CENT)
109         currentButton = (lcdReadButtons(uart1) ==
LCD_BTN_CENTER);
110     if (button == LCD_RIGHT)
111         currentButton = (lcdReadButtons(uart1) ==
LCD_BTN_RIGHT);
112 }
113 return currentButton;
114 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.21.2.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons.

Initializes the buttons.

Definition at line 20 of file toggle.c.

References buttonPressed.

```
20      {
21      for (int i = 0; i < 27; i++)
22          buttonPressed[i] = false;
23 }
```

7.21.2.3 buttonIsNewPress()

```
bool buttonIsNewPress (
    button_t button )
```

Detects if button is a new press from most recent check by comparing previous value to current value.

Parameters

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

Returns

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

Parameters

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

Returns

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

Example code:

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

Definition at line 135 of file toggle.c.

References buttonGetState(), and buttonPressed.

```
135     bool currentButton = buttonGetState(button);
136
137     if (!currentButton) // buttons is not currently pressed
138         buttonPressed[button] = false;
140
141     if (currentButton && !buttonPressed[button]) {
142         // button is currently pressed and was not detected as being pressed during last check
143         buttonPressed[button] = true;
144         return true;
145     }
146     else return false; // button is not pressed or was already detected
147 }
```

Here is the call graph for this function:

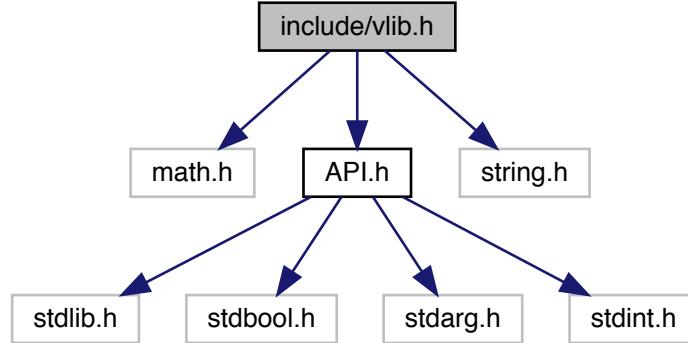


7.22 include/vlib.h File Reference

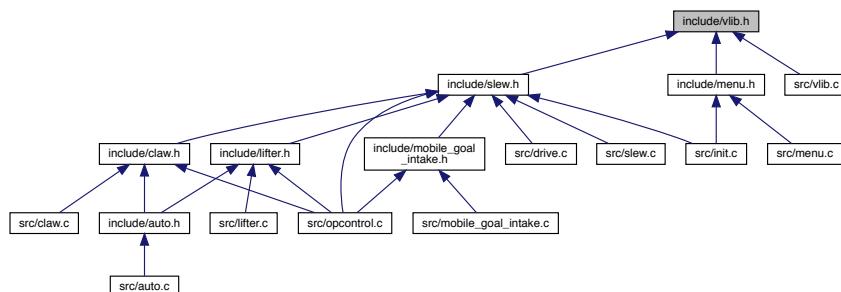
Contains misc helpful functions.

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

```
#include <string.h>
Include dependency graph for vlib.h:
```



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



Functions

- void * `calloc_real` (size_t elements, size_t size)
- void `ftoa_bad` (float a, char *buffer, int precision)
 - converts a float to string.*
- int `itoa_bad` (int a, char *buffer, int digits)
 - converts a int to string.*
- void `reverse` (char *str, int len)
 - reverses a string 'str' of length 'len'*

7.22.1 Detailed Description

Contains misc helpful functions.

Author

Chris Jerrett

Date

9/9/2017

7.22.2 Function Documentation

7.22.2.1 calloc_real()

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

Referenced by init_menu_var().

Here is the caller graph for this function:



7.22.2.2 ftoa_bad()

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

converts a float to string.

Parameters

<i>a</i>	the float
<i>buffer</i>	the string the float will be written to.
<i>precision</i>	digits after the decimal to write

Author

Christian DeSimone

Date

9/26/2017

Definition at line 30 of file vlib.c.

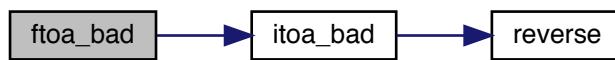
References `itoa_bad()`.

Referenced by `calculate_current_display()`.

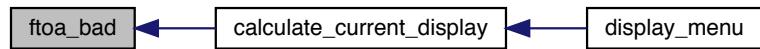
```

30
31 // Extract integer part
32 int ipart = (int)a;
33
34 // Extract floating part
35 float fpart = a - (float)ipart;
36
37 // convert integer part to string
38 int i = itoa_bad(ipart, buffer, 0);
39
40 // check for display option after point
41 if(precision != 0) {
42     buffer[i] = '.';
43     // add dot
44     // Get the value of fraction part up to given num.
45     // of points after dot. The third parameter is needed
46     // to handle cases like 233.007
47     fpart = fpart * pow(10, precision);
48
49     itoa_bad((int)fpart, buffer + i + 1, precision);
50 }
51 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.22.2.3 itoa_bad()

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

converts a int to string.

Parameters

<i>a</i>	the integer
<i>buffer</i>	the string the int will be written to.
<i>digits</i>	the number of digits to be written

Returns

the digits

Author

Chris Jerrett, Christian DeSimone

Date

9/9/2017

Definition at line 13 of file vlib.c.

References reverse().

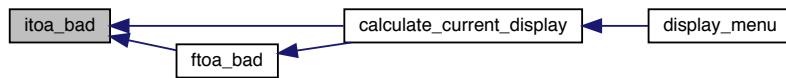
Referenced by calculate_current_display(), and ftoa_bad().

```
13     int i = 0;
14     while (a) {
15         buffer[i++] = (a%10) + '0';
16         a = a/10;
17     }
18
19
20 // If number of digits required is more, then
21 // add 0s at the beginning
22 while (i < digits)
23     buffer[i++] = '0';
24
25 reverse(buffer, i);
26 buffer[i] = '\0';
27 return i;
28 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.22.2.4 reverse()

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

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

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

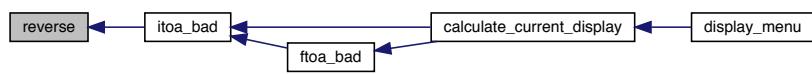
Definition at line 3 of file vlib.c.

Referenced by itoa_bad().

```

3
4     int i=0, j=len-1, temp;
5     while (i<j) {
6         temp = str[i];
7         str[i] = str[j];
8         str[j] = temp;
9         i++; j--;
10    }
11 }
```

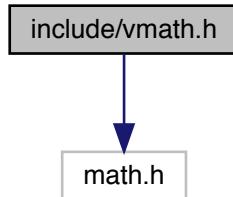
Here is the caller graph for this function:



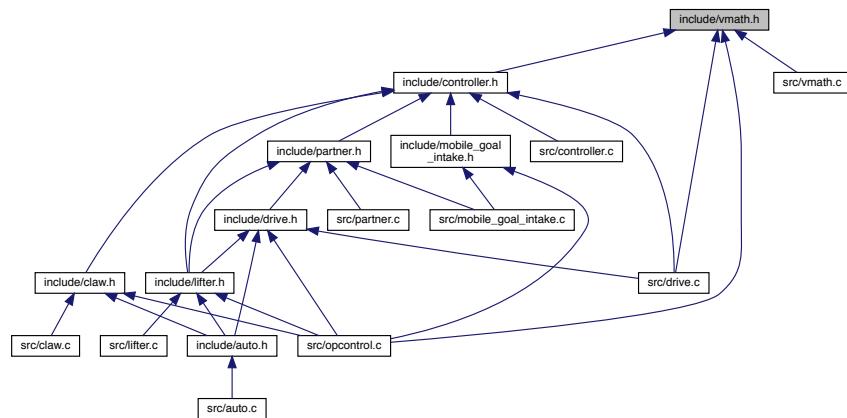
7.23 include/vmath.h File Reference

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

```
#include <math.h>
Include dependency graph for vmath.h:
```



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



Data Structures

- struct `cord`
A struct that contains cartesian coordinates.
- struct `polar_cord`
A struct that contains polar coordinates.

Macros

- #define `M_PI` 3.14159265358979323846

Functions

- struct `polar_cord cartesian_cord_to_polar` (struct `cord` cords)
Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.
- struct `polar_cord cartesian_to_polar` (float x, float y)
Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- int `max` (int a, int b)
- int `min` (int a, int b)
- double `sind` (double angle)

7.23.1 Detailed Description

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

Author

Christian Desimone
Chris Jerrett

Date

9/9/2017

7.23.2 Macro Definition Documentation

7.23.2.1 M_PI

```
#define M_PI 3.14159265358979323846
```

Definition at line 13 of file vmath.h.

Referenced by sind().

7.23.3 Function Documentation

7.23.3.1 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

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

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)
[cord](#)

Definition at line 33 of file vmath.c.

References [cartesian_to_polar\(\)](#).

```
33
34     return cartesian_to_polar(cords.x, cords.y);
35 }
```

Here is the call graph for this function:



7.23.3.2 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

<code>x</code>	float value of the x cartesian coordinate.
<code>y</code>	float value of the y cartesian coordinate.

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)

Definition at line 3 of file vmath.c.

References `polar_cord::angle`, and `polar_cord::magnitue`.

Referenced by `cartesian_cord_to_polar()`.

```

3
4   float degree = 0;
5   double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
6
7   if(x < 0){
8       degree += 180.0;
9   }
10  else if(x > 0 && y < 0){
11      degree += 360.0;
12  }
13
14  if(x != 0 && y != 0){
15      degree += atan((float)y / (float)x);
16  }
17  else if(x == 0 && y > 0){
18      degree = 90.0;
19  }
20  else if(y == 0 && x < 0){
21      degree = 180.0;
22  }
23  else if(x == 0 && y < 0){
24      degree = 270.0;
25  }
26
27  struct polar_cord p;
28  p.angle = degree;
29  p.magnitue = magnitude;
30  return p;
31 }
```

Here is the caller graph for this function:



7.23.3.3 max()

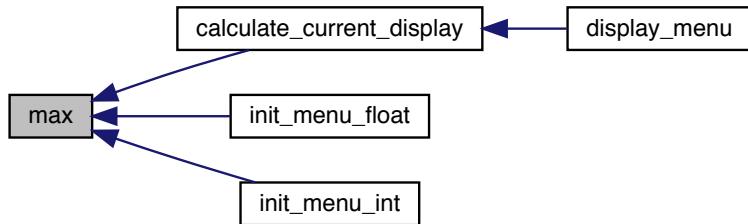
```
int max (
    int a,
    int b )
```

Definition at line 48 of file vmath.c.

Referenced by calculate_current_display(), init_menu_float(), and init_menu_int().

```
48
49     if(a > b)  return a;
50     return b;
51 }
```

Here is the caller graph for this function:



7.23.3.4 min()

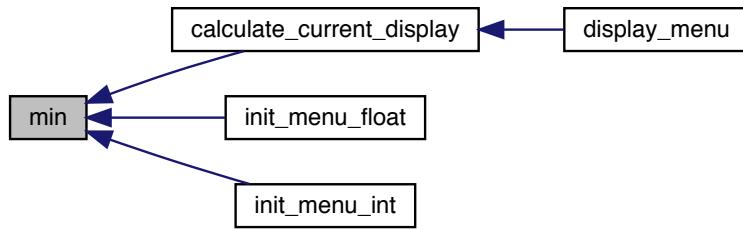
```
int min (
    int a,
    int b )
```

Definition at line 42 of file vmath.c.

Referenced by calculate_current_display(), init_menu_float(), and init_menu_int().

```
42
43     if(a < b)  return a;
44     return b;
45 }
```

Here is the caller graph for this function:



7.23.3.5 sind()

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

Definition at line 37 of file vmath.c.

References M_PI.

```
37      {
38      double angleradians = angle * M_PI / 180.0f;
39      return sin(angleradians);
40 }
```

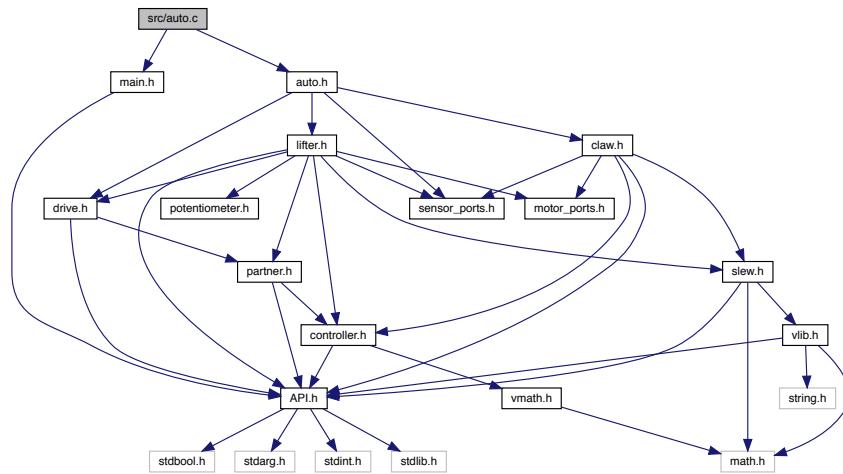
7.24 README.md File Reference

7.25 src/auto.c File Reference

File for autonomous code.

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

Include dependency graph for auto.c:



Functions

- void **autonomous()**

7.25.1 Detailed Description

File for autonomous code.

This file should contain the user [autonomous\(\)](#) 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.

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

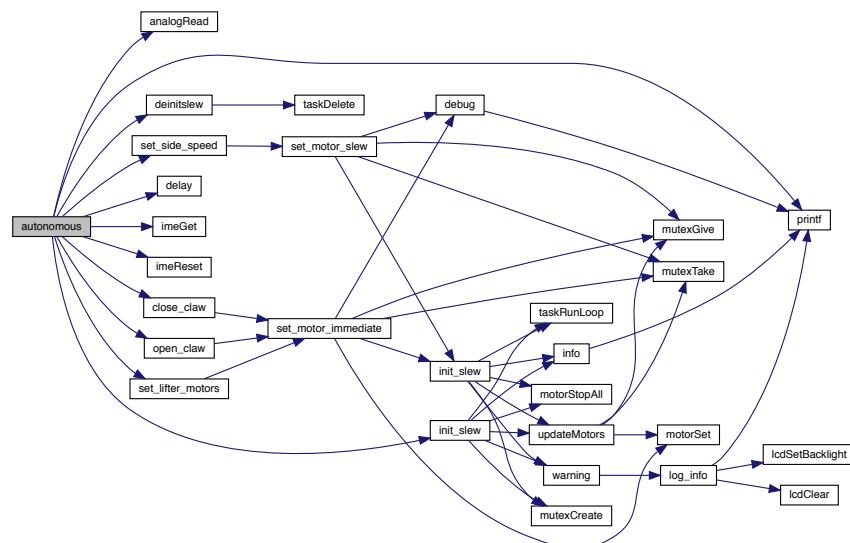
Definition at line 30 of file auto.c.

References `analogRead()`, `BOTH`, `close_claw()`, `deinitSlew()`, `delay()`, `GOAL_HEIGHT`, `imeGet()`, `imeReset()`, `initSlew()`, `LIFTER`, `MID_LEFT_DRIVE`, `MID_RIGHT_DRIVE`, `open_claw()`, `printf()`, `set_lifter_motors()`, and `set_sideSpeed()`.

```

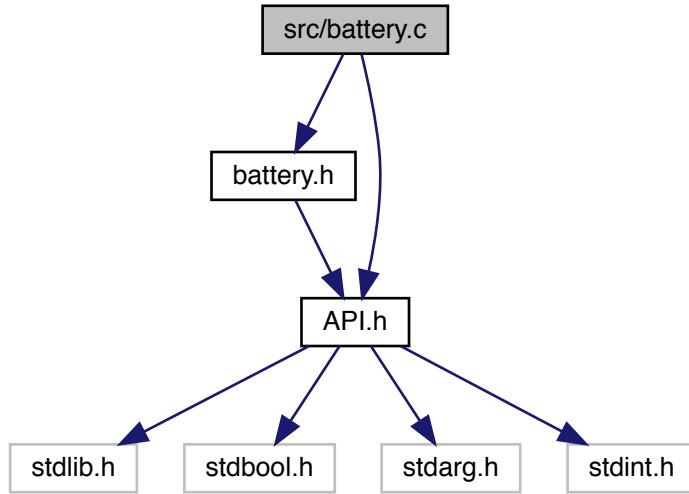
30
31     init_slew();
32
33     delay(10);
34     printf("auto\n");
35     int counts_drive_left;
36     int counts_drive_right;
37     int counts_drive;
38     imeReset(MID_LEFT_DRIVE);
39     imeReset(MID_RIGHT_DRIVE);
40     imeGet(MID_LEFT_DRIVE, &counts_drive_left);
41     imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
42     counts_drive = counts_drive_left + counts_drive_right;
43     counts_drive /= 2;
44     close_claw();
45
46     delay(300);
47     //unsigned long time = millis();
48     while(analogRead(LIFTER) < GOAL_HEIGHT) {
49         set_lifter_motors(-127);
50     }
51     set_lifter_motors(0);
52     while(counts_drive_left < 530){
53         set_side_speed(BOTH, 127);
54         imeGet(MID_LEFT_DRIVE, &counts_drive_left);
55         imeGet(MID_RIGHT_DRIVE, &counts_drive_right);
56         counts_drive = counts_drive_left + counts_drive_right;
57         counts_drive /= 2;
58         //if(millis() - time > 1000) break;
59     }
60     set_side_speed(BOTH, 0);
61     delay(1000);
62     open_claw();
63     delay(1000);
64     deinitSlew();
65 }
```

Here is the call graph for this function:



7.26 src/battery.c File Reference

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



Functions

- double [backup_battery_voltage \(\)](#)
gets the backup battery voltage
- bool [battery_level_acceptable \(\)](#)
returns if the batteries are acceptable
- double [main_battery_voltage \(\)](#)
gets the main battery voltage

7.26.1 Function Documentation

7.26.1.1 [backup_battery_voltage\(\)](#)

```
double backup_battery_voltage ( )
```

gets the backup battery voltage

Author

Chris Jerrett

Definition at line 8 of file battery.c.

References powerLevelBackup().

Referenced by battery_level_acceptable().

```
8          {  
9     return powerLevelBackup() / 1000.0;  
10 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.26.1.2 battery_level_acceptable()

```
bool battery_level_acceptable ( )
```

returns if the batteries are acceptable

See also

[MIN_MAIN_VOLTAGE](#)
[MIN_BACKUP_VOLTAGE](#)

Author

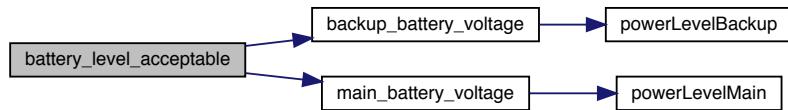
Chris Jerrett

Definition at line 12 of file battery.c.

References backup_battery_voltage(), main_battery_voltage(), MIN_BACKUP_VOLTAGE, and MIN_MAIN_VOLTAGE.

```
12             {
13     if(main_battery_voltage() < MIN_MAIN_VOLTAGE) return false;
14     if(backup_battery_voltage() < MIN_BACKUP_VOLTAGE) return false;
15     return true;
16 }
```

Here is the call graph for this function:



7.26.1.3 main_battery_voltage()

```
double main_battery_voltage ( )
```

gets the main battery voltage

Author

Chris Jerrett

Definition at line 4 of file battery.c.

References powerLevelMain().

Referenced by battery_level_acceptable().

```
4             {
5     return powerLevelMain() / 1000.0;
6 }
```

Here is the call graph for this function:

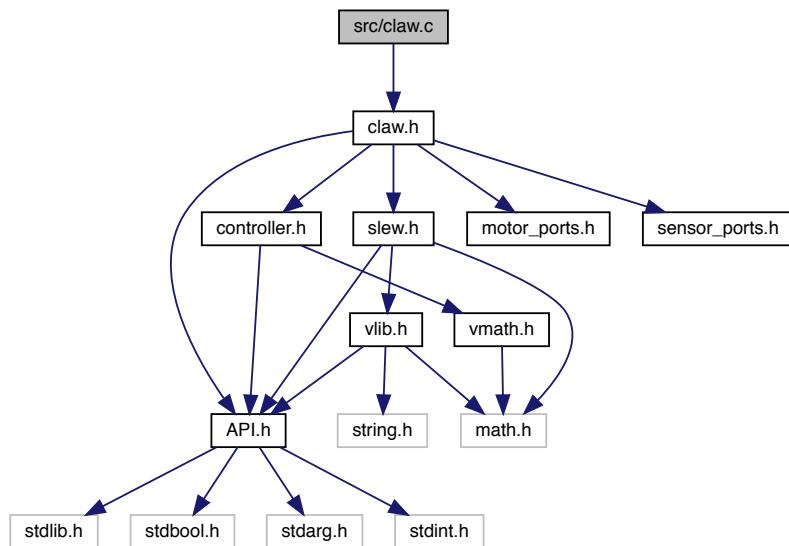


Here is the caller graph for this function:



7.27 src/claw.c File Reference

```
#include "claw.h"
Include dependency graph for claw.c:
```



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.

7.27.1 Function Documentation

7.27.1.1 **close_claw()**

```
void close_claw ( )
```

Drives the motors to close the claw.

Author

Chris Jerrett

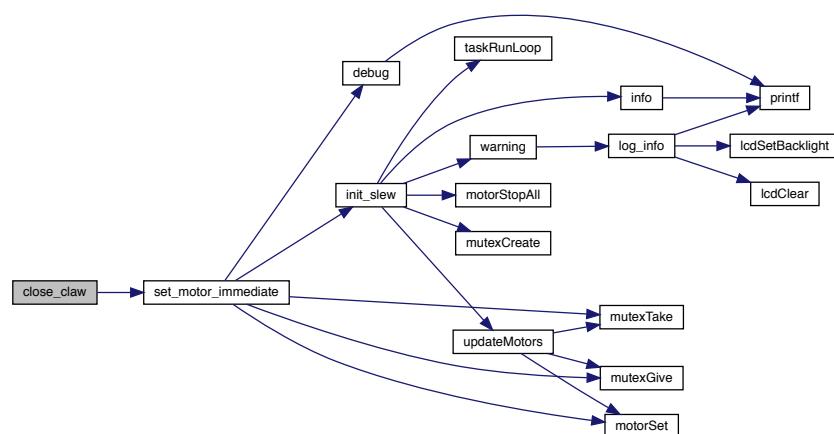
Definition at line 37 of file claw.c.

References CLAW_MOTOR, MIN_CLAW_SPEED, and set_motor_immediate().

Referenced by autonomous().

```
37 {
38     set_motor_immediate(CLAW_MOTOR, MIN_CLAW_SPEED);
39 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.1.2 getClawTicks()

```
unsigned int getClawTicks ( )
```

Gets the claw position in potentiometer ticks.

Author

Chris Jerrett

Definition at line 29 of file claw.c.

References analogRead(), and CLAW_POT.

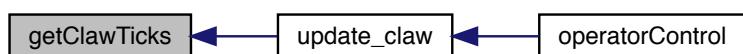
Referenced by update_claw().

```
29 {  
30     return analogRead(CLAW_POT);  
31 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.1.3 open_claw()

```
void open_claw ( )
```

Drives the motors to open the claw.

Author

Chris Jerrett

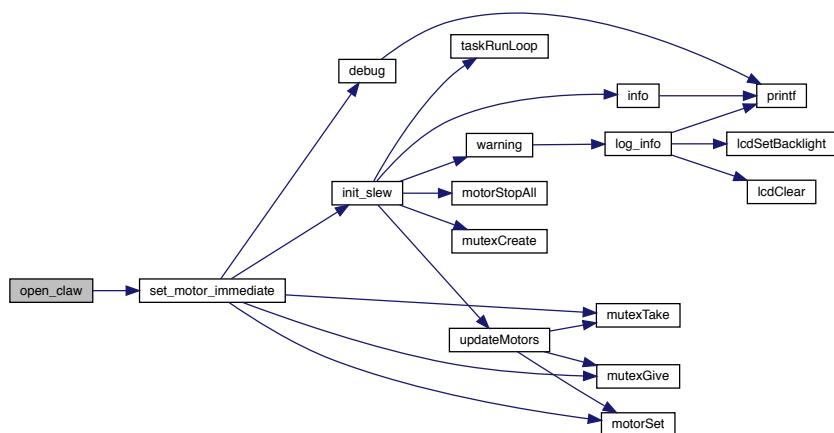
Definition at line 33 of file claw.c.

References CLAW_MOTOR, MAX_CLAW_SPEED, and set_motor_immediate().

Referenced by autonomous().

```
33          {  
34     set_motor_immediate(CLAW_MOTOR, MAX_CLAW_SPEED);  
35 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.1.4 set_claw_motor()

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

sets the claw motor speed

Author

Chris Jerrett

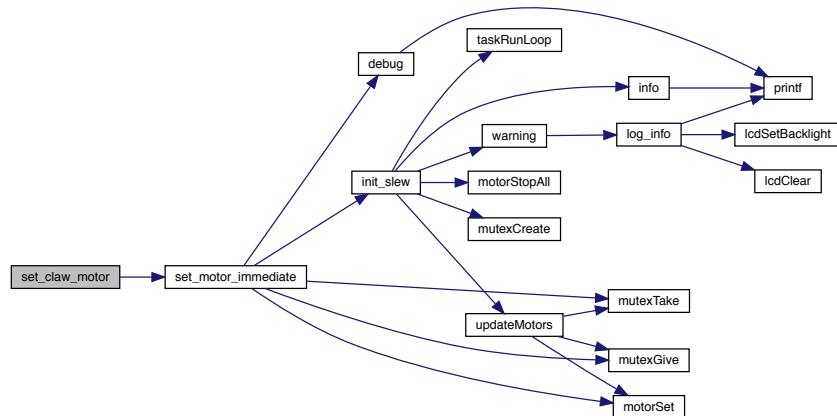
Definition at line 25 of file claw.c.

References CLAW_MOTOR, and set_motor_immediate().

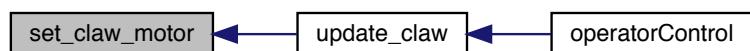
Referenced by update_claw().

```
25  {
26  set_motor_immediate(CLAW_MOTOR, v);
27 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.27.1.5 update_claw()

```
void update_claw ( )
```

Updates the claw motor values.

Author

Chris Jerrett

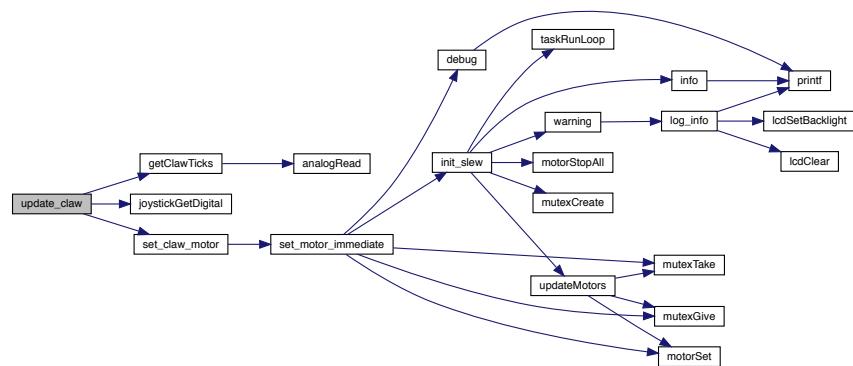
Definition at line 3 of file claw.c.

References CLAW_CLOSE, CLAW_CLOSE_STATE, CLAW_CLOSE_VAL, CLAW_OPEN, CLAW_OPEN_STATE, CLAW_OPEN_VAL, CLAW_P, getClawTicks(), joystickGetDigital(), and set_claw_motor().

Referenced by operatorControl().

```
3             {
4     static int last_error = 0;
5     static enum claw_state state = CLAW_OPEN_STATE;
6     if(joystickGetDigital(CLAW_CLOSE)) {
7         state = CLAW_CLOSE_STATE;
8     }
9     else if(joystickGetDigital(CLAW_OPEN)) {
10        state = CLAW_OPEN_STATE;
11    } else {
12        int p = 0;
13        if(state == CLAW_OPEN_STATE) {
14            p = getClawTicks() - CLAW_OPEN_VAL;
15        } else {
16            p = getClawTicks() - CLAW_CLOSE_VAL;
17        }
18        int d = (p - last_error);
19        int motor = CLAW_P * p;
20        set_claw_motor(motor);
21    }
22 }
23 }
```

Here is the call graph for this function:

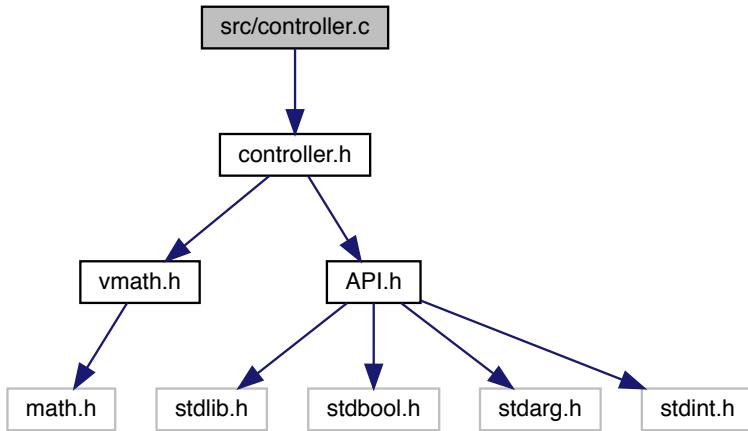


Here is the caller graph for this function:



7.28 src/controller.c File Reference

```
#include "controller.h"
Include dependency graph for controller.c:
```



Functions

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

7.28.1 Function Documentation

7.28.1.1 `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 3 of file controller.c.

References `joystickGetAnalog()`, `LEFT_JOY_X`, `LEFT_JOY_Y`, `RIGHT_JOY`, `RIGHT_JOY_X`, `RIGHT_JOY_Y`, `cord::x`, and `cord::y`.

```

3
4     int x;
5     int y;
6     if(side == RIGHT_JOY) {
7         y = joystickGetAnalog(controller, RIGHT_JOY_X);
8         x = joystickGetAnalog(controller, RIGHT_JOY_Y);
9     } else {
10        y = joystickGetAnalog(controller, LEFT_JOY_X);
11        x = joystickGetAnalog(controller, LEFT_JOY_Y);
12    }
13    struct cord c;
14    c.x = x;
15    c.y = y;
16    return c;
17 }
```

Here is the call graph for this function:

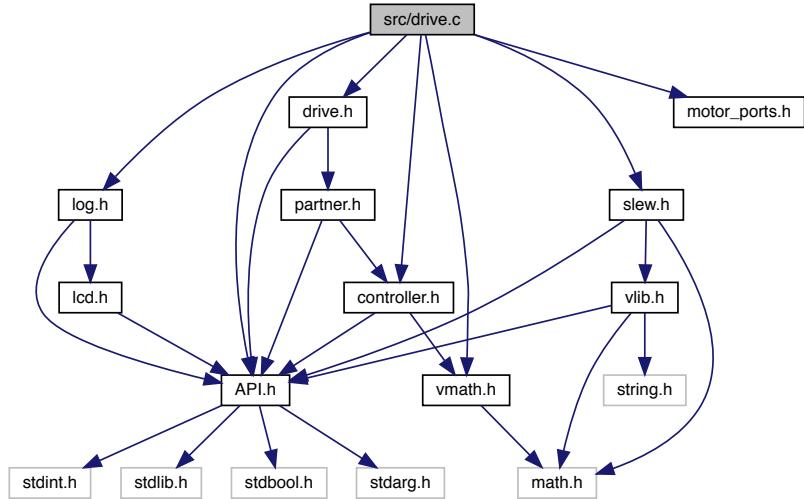


7.29 src/drive.c File Reference

```

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

Include dependency graph for drive.c:



Functions

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

Variables

- static int [thresh = 30](#)

7.29.1 Function Documentation

7.29.1.1 [getThresh\(\)](#)

```
int getThresh ( )
```

Definition at line 13 of file `drive.c`.

References `thresh`.

```
13 {
14     return thresh;
15 }
```

7.29.1.2 [joystickExp\(\)](#)

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

Applies exponential scale to a joystick value.

Author

Christian DeSimone, Chris Jerrett

Parameters

<code>joystickVal</code>	the analog value from the joystick
--------------------------	------------------------------------

Date

9/21/2017

Definition at line 69 of file drive.c.

References THRESHOLD.

```

69
70     //make the offset negative if moving backwards
71     if (abs(joystickVal) < THRESHOLD) {
72         return 0;
73     }
74
75     int offset;
76     if (joystickVal < 0) {
77         offset = - (THRESHOLD);
78     } else {
79         offset = THRESHOLD;
80     }
81
82     return (pow(joystickVal/10 , 3) / 18 + offset) * 0.8;
83 }
```

7.29.1.3 set_side_speed()

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

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

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

Definition at line 50 of file drive.c.

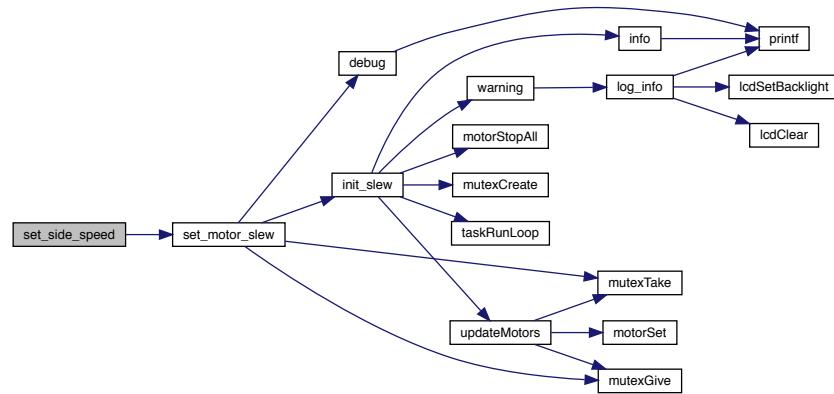
References BOTH, LEFT, MOTOR_BACK_LEFT, MOTOR_BACK_RIGHT, MOTOR_FRONT_LEFT, MOTOR_FRONT_RIGHT, MOTOR_MIDDLE_LEFT, MOTOR_MIDDLE_RIGHT, RIGHT, and set_motor_slew().

Referenced by autonomous(), and update_drive_motors().

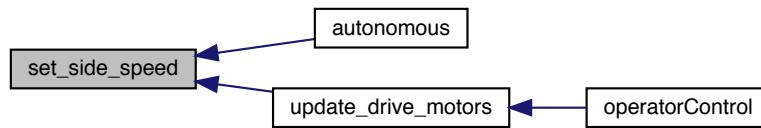
```

50
51     if(side == RIGHT || side == BOTH){
52         set_motor_slew(MOTOR_BACK_RIGHT , -speed);
53         set_motor_slew(MOTOR_FRONT_RIGHT, -speed);
54         set_motor_slew(MOTOR_MIDDLE_RIGHT, -speed);
55     }
56     if(side == LEFT || side == BOTH){
57         set_motor_slew(MOTOR_BACK_LEFT, speed);
58         set_motor_slew(MOTOR_MIDDLE_LEFT, speed);
59         set_motor_slew(MOTOR_FRONT_LEFT, speed);
60     }
61 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.29.1.4 setThresh()

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

Sets the deadzone threshhold on the drive.

Author

Chris Jerrett

Definition at line 17 of file `drive.c`.

References `thresh`.

```
17
18     thresh = t;
19 }
```

7.29.1.5 update_drive_motors()

```
void update_drive_motors( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

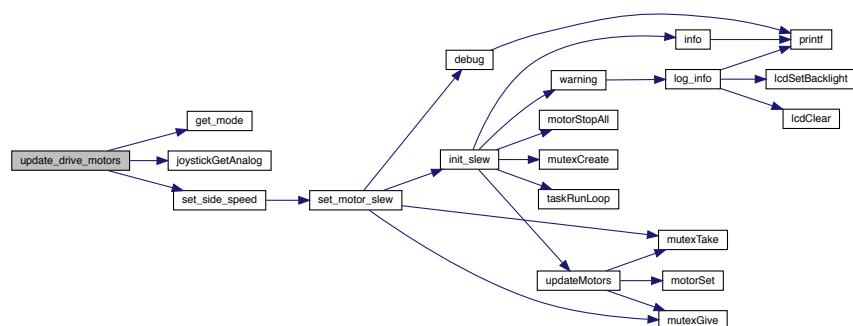
Definition at line 21 of file drive.c.

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

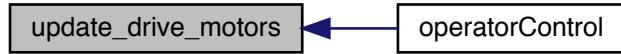
Referenced by operatorControl().

```
21
22
23 int x = 0;
24 int y = 0;
25 if(get_mode() == PARTNER_CONTROLLER_MODE) {
26     x = (joystickGetAnalog(PARTNER, 3));
27     y = (joystickGetAnalog(PARTNER, 1));
28 } else {
29     x = -(joystickGetAnalog(MASTER, 3));
30     y = (joystickGetAnalog(MASTER, 1));
31 }
32
33 //x = joystickExp(x);
34 //y = joystickExp(y);
35 if(x < thresh && x > -thresh) {
36     x = 0;
37 }
38 if(y < thresh && y > -thresh) {
39     y = 0;
40 }
41
42 int r = (x + y);
43 int l = -(x - y);
44
45 set_side_speed(LEFT, l);
46 set_side_speed(RIGHT, -r);
47
48 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.29.2 Variable Documentation

7.29.2.1 thresh

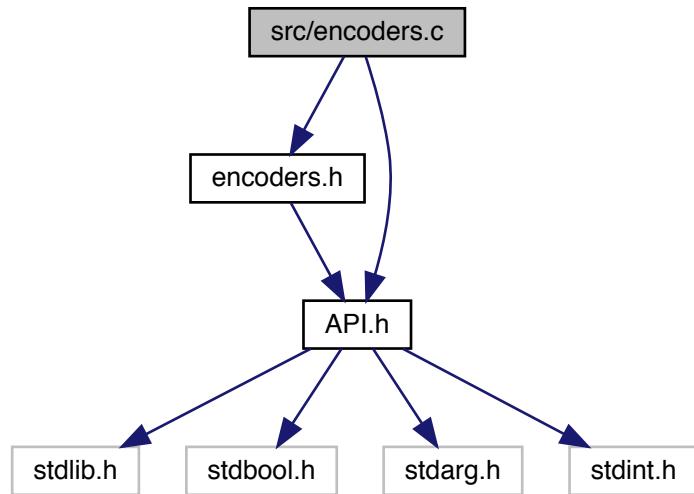
```
int thresh = 30 [static]
```

Definition at line 11 of file drive.c.

Referenced by getThresh(), setThresh(), and update_drive_motors().

7.30 src/encoders.c File Reference

```
#include "encoders.h"
#include <API.h>
Include dependency graph for encoders.c:
```



Functions

- int [get_encoder_ticks](#) (unsigned char address)
Gets the encoder ticks since last reset.
- int [get_encoder_velocity](#) (unsigned char address)
Gets the encoder reads.
- bool [init_encoders](#) ()
Initializes all motor encoders.

7.30.1 Function Documentation

7.30.1.1 [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 12 of file encoders.c.

References [imeGet\(\)](#).

```
12
13     int i = 0;
14     imeGet(address, &i);
15     return i;
16 }
```

Here is the call graph for this function:



7.30.1.2 get_encoder_velocity()

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

Gets the encoder reads.

Author

Chris Jerrett

Date

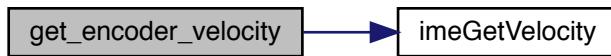
9/15/2017

Definition at line 18 of file encoders.c.

References `imeGetVelocity()`.

```
18
19     int i = 0;
20     imeGetVelocity(address, &i);
21     return i;
22 }
```

Here is the call graph for this function:



7.30.1.3 init_encoders()

```
bool init_encoders ( )
```

Initializes all motor encoders.

Author

Chris Jerrett

Date

9/9/2017

See also

[IME_NUMBER](#)

Definition at line 4 of file encoders.c.

References IME_NUMBER, and imelInitializeAll().

```

4
5 #ifdef IME_NUMBER
6 return imelInitializeAll() == IME_NUMBER;
7 #else
8 return imelInitializeAll();
9#endif
10 }

```

Here is the call graph for this function:

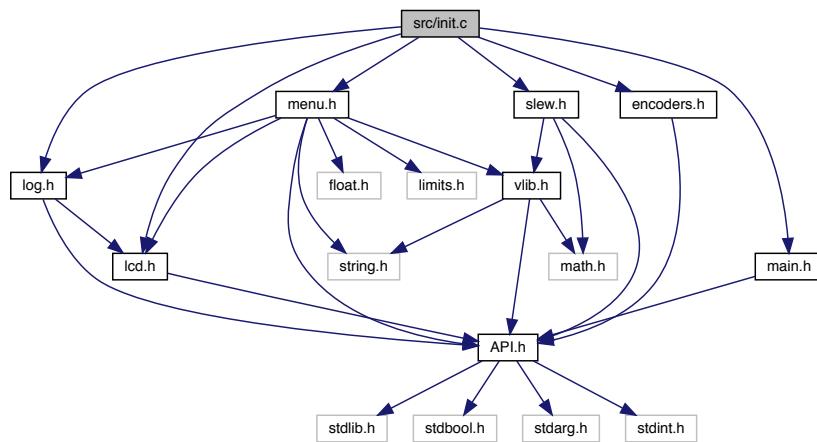


7.31 src/init.c File Reference

File for initialization code.

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

Include dependency graph for init.c:



Functions

- void `initialize ()`
- void `initializeIO ()`

7.31.1 Detailed Description

File for initialization code.

This file should contain the user `initialize()` 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.

7.31.2 Function Documentation

7.31.2.1 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.

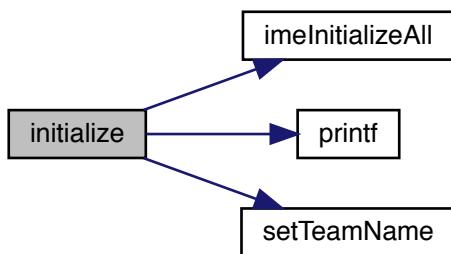
This function must exit relatively promptly, or the `operatorControl()` and `autonomous()` tasks will not start. An autonomous mode selection menu like the `pre_auton()` in other environments can be implemented in this task if desired.

Definition at line 47 of file init.c.

References `imeInitializeAll()`, `printf()`, and `setTeamName()`.

```
47
48     int c = imeInitializeAll();
49     setTeamName("9228A");
50     printf("Counts : %d\n", c);
51 }
```

Here is the call graph for this function:



7.31.2.2 initializeIO()

```
void initializeIO ( )
```

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

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

Definition at line 30 of file init.c.

References [watchdogInit\(\)](#).

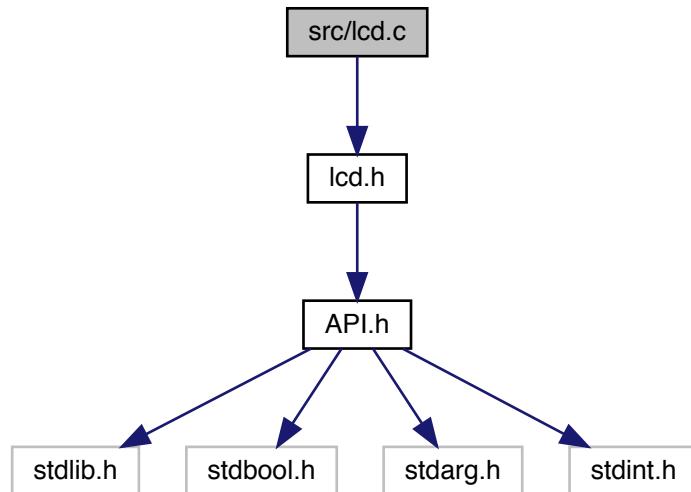
```
30
31     watchdogInit () ;
32 }
```

Here is the call graph for this function:



7.32 src/lcd.c File Reference

```
#include "lcd.h"
Include dependency graph for lcd.c:
```



Functions

- void [init_main_lcd \(FILE *lcd\)](#)
Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- static void [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 [prompt_confirmation \(const char *confirm_text\)](#)
Prompts the user to confirm a string. User must press middle button to confirm. Function is not thread safe and will stall a thread.

Variables

- static FILE * [lcd_port](#) = NULL

7.32.1 Function Documentation

7.32.1.1 init_main_lcd()

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

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

Parameters

<i>lcd</i>	the urart port of the lcd screen
------------	----------------------------------

See also

[uart1](#)
[uart2](#)

Author

Chris Jerrett

Date

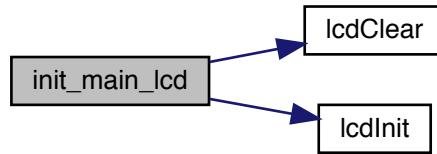
9/9/2017

Definition at line 39 of file lcd.c.

References `lcd_port`, `lcdClear()`, and `lcdInit()`.

```
39      lcdInit(lcd);
40      lcdClear(lcd);
41      lcd_port = lcd;
42  }
```

Here is the call graph for this function:



7.32.1.2 `lcd_assert()`

```
static void 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`, and `printf()`.

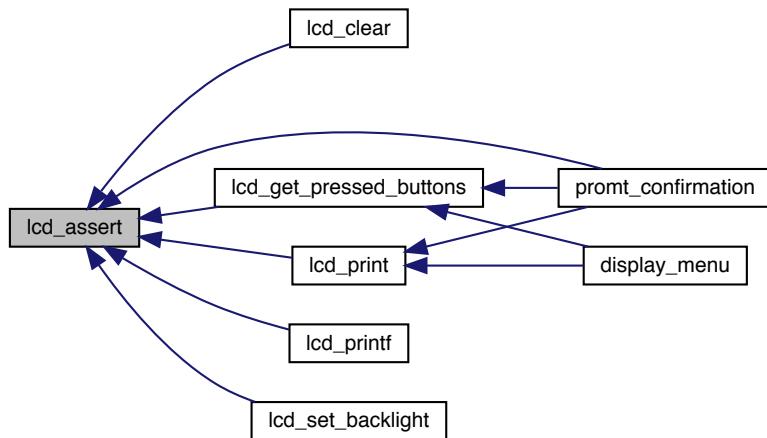
Referenced by `lcd_clear()`, `lcd_get_pressed_buttons()`, `lcd_print()`, `lcd_printf()`, `lcd_set_backlight()`, and `prompt_confirmation()`.

```
13     if(lcd_port != NULL) {
14         printf("LCD NULL!");
15         exit(1);
16     }
17 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.1.3 `lcd_clear()`

```
void lcd_clear( )
```

Clears the lcd.

Author

Chris Jerrett

Date

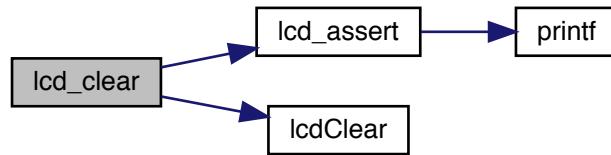
9/9/2017

Definition at line 34 of file lcd.c.

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

```
34      {
35     lcd_assert();
36     lcdClear(lcd_port);
37 }
```

Here is the call graph for this function:



7.32.1.4 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](#)

Definition at line 20 of file lcd.c.

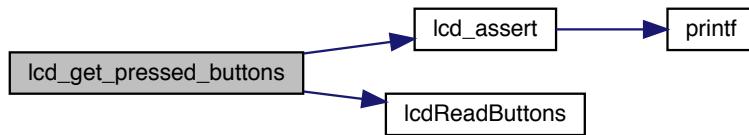
References `lcd_assert()`, `lcd_port`, `lcdReadButtons()`, `lcd_buttons::left`, `lcd_buttons::middle`, `PRESSED`, `RELEASED`, and `lcd_buttons::right`.

Referenced by `display_menu()`, and `prompt_confirmation()`.

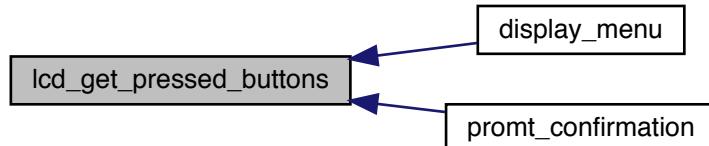
```

20
21     lcd_assert();
22     unsigned int btn_binary = lcdReadButtons(lcd_port);
23     bool left = btn_binary & 0x1;
24     bool middle = btn_binary & 0x2;
25     bool right = btn_binary & 0x4;
26     lcd_buttons btns;
27     btns.left = left ? PRESSED : RELEASED;
28     btns.middle = middle ? PRESSED : RELEASED;
29     btns.right = right ? PRESSED : RELEASED;
30
31     return btns;
32 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.1.5 lcd_print()

```

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

prints a string to a line on the lcd

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

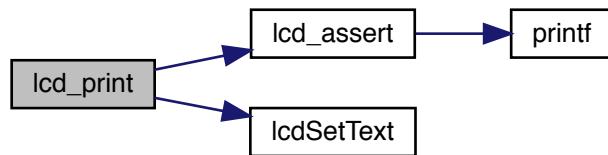
Definition at line 45 of file lcd.c.

References `lcd_assert()`, `lcd_port`, and `lcdSetText()`.

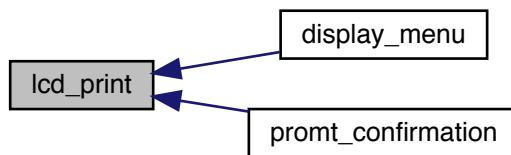
Referenced by `display_menu()`, and `prompt_confirmation()`.

```
45
46     lcd_assert();
47     lcdSetText(lcd_port, line, str);
48 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.32.1.6 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

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

Author

Chris Jerrett

Date

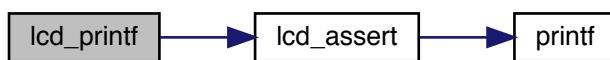
9/9/2017

Definition at line 50 of file lcd.c.

References lcd_assert(), and lcd_port.

```
50
51     lcd_assert ();
52     lcdPrint(lcd_port, line, format_str);
53 }
```

Here is the call graph for this function:



7.32.1.7 lcd_set_backlight()

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

sets the backlight of the lcd

Parameters

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

Author

Chris Jerrett

Date

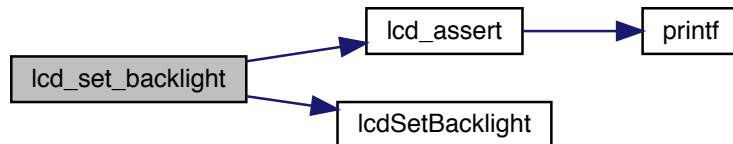
9/9/2017

Definition at line 55 of file lcd.c.

References `lcd_assert()`, `lcd_port`, and `lcdSetBacklight()`.

```
55
56     lcd_assert();
57     lcdSetBacklight(lcd_port, state);
58 }
```

Here is the call graph for this function:



7.32.1.8 prompt_confirmation()

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

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

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

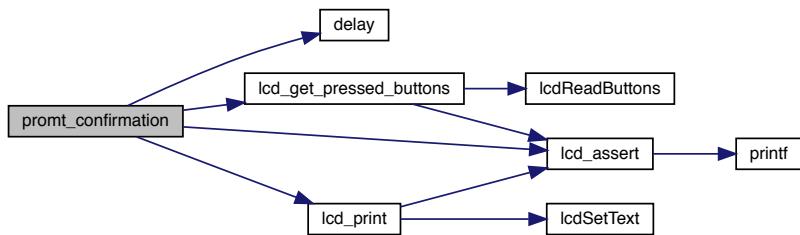
Definition at line 60 of file lcd.c.

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

```

60      lcd_assert();
61      lcd_print(1, confirm_text);
62      while(lcd_get_pressed_buttons().middle != PRESSED) {
63          delay(200);
64      }
65  }
```

Here is the call graph for this function:



7.32.2 Variable Documentation

7.32.2.1 lcd_port

```
FILE* lcd_port = NULL [static]
```

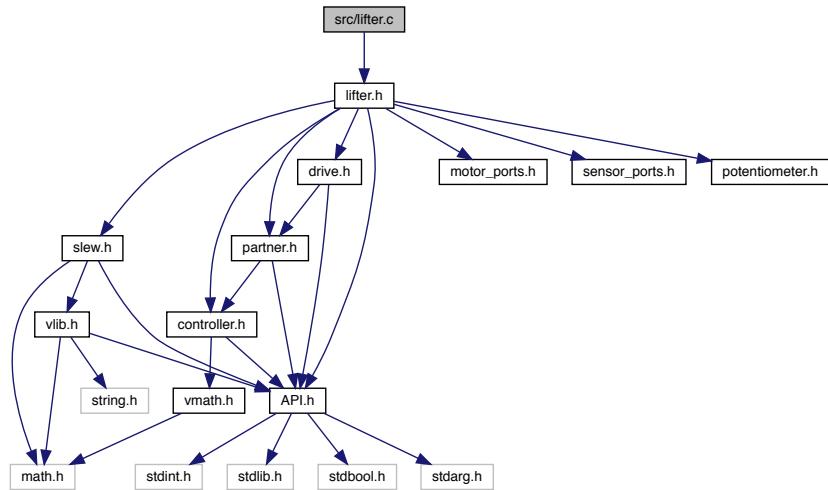
The port of the initialized lcd

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().

7.33 src/lifter.c File Reference

```
#include "lifter.h"
Include dependency graph for lifter.c:
```



Functions

- `double getLifterHeight ()`
- `int getLifterTicks ()`
- `float lifterPotentiometerToDegree (int x)`
- `void lower_lifter ()`
- `void raise_lifter ()`
- `void set_lifter_motors (const int v)`
- `void set_lifter_pos (int pos)`
- `void update_lifter ()`

7.33.1 Function Documentation

7.33.1.1 `getLifterHeight()`

```
double getLifterHeight ( )
```

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

References `getLifterTicks()`.

```

86
87     unsigned int ticks = getLifterTicks();
88     return (-2 * pow(10, (-9 * ticks)) + 6 * (pow(10, (-6 * ticks * ticks)))) + 0.0198 * ticks + 2.3033;
89 }
```

Here is the call graph for this function:



7.33.1.2 getLifterTicks()

int getLifterTicks ()

Definition at line 81 of file lifter.c.

References analogRead(), and LIFTER.

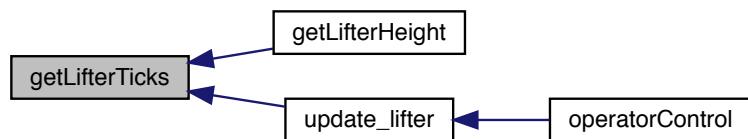
Referenced by getLifterHeight(), and update_lifter().

```
81 {  
82     return analogRead(LIFTER);  
83 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.1.3 lifterPotentiometerToDegree()

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

Definition at line 77 of file lifter.c.

References DEG_MAX, TICK_DIFF, and TICK_MAX.

```
77     {
78     return (x - TICK_DIFF) / TICK_MAX * DEG_MAX;
79 }
```

7.33.1.4 lower_lifter()

```
void lower_lifter ( )
```

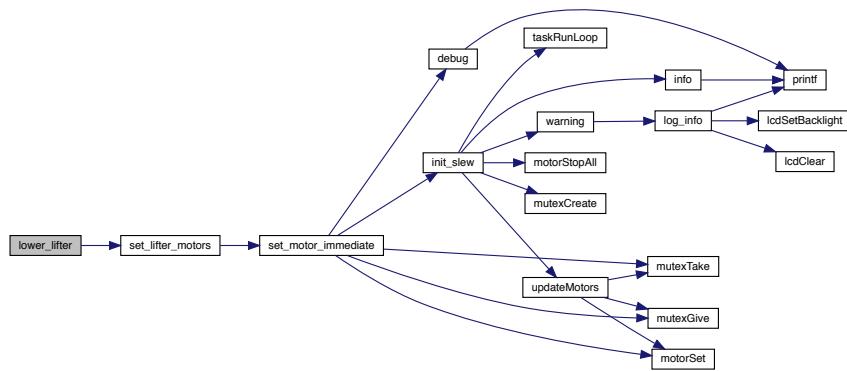
Definition at line 16 of file lifter.c.

References MIN_SPEED, and set_lifter_motors().

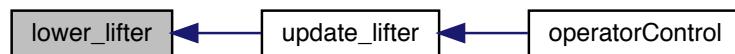
Referenced by update_lifter().

```
16     {
17     set_lifter_motors(MIN_SPEED);
18 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.1.5 raise_lifter()

```
void raise_lifter ( )
```

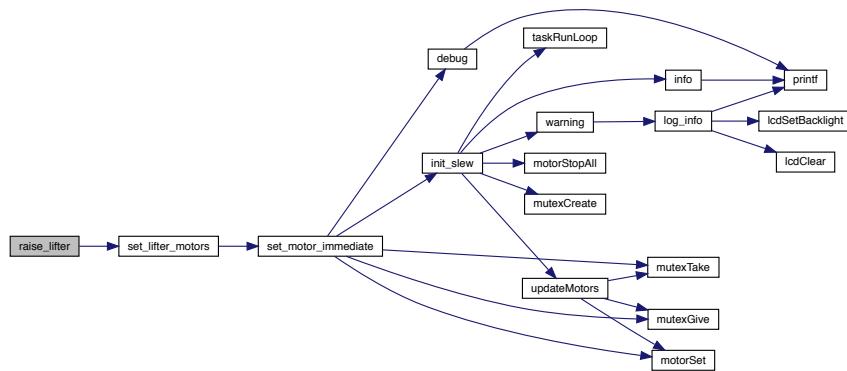
Definition at line 12 of file lifter.c.

References MAX_SPEED, and set_lifter_motors().

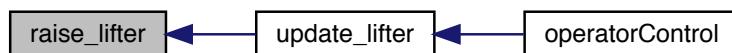
Referenced by update_lifter().

```
12      {
13     set_lifter_motors(MAX_SPEED);
14 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.1.6 set_lifter_motors()

```
void set_lifter_motors (
    const int v )
```

Definition at line 3 of file lifter.c.

References MOTOR_LIFT_TOP_LEFT, MOTOR_LIFT_TOP_RIGHT, and set_motor_immediate().

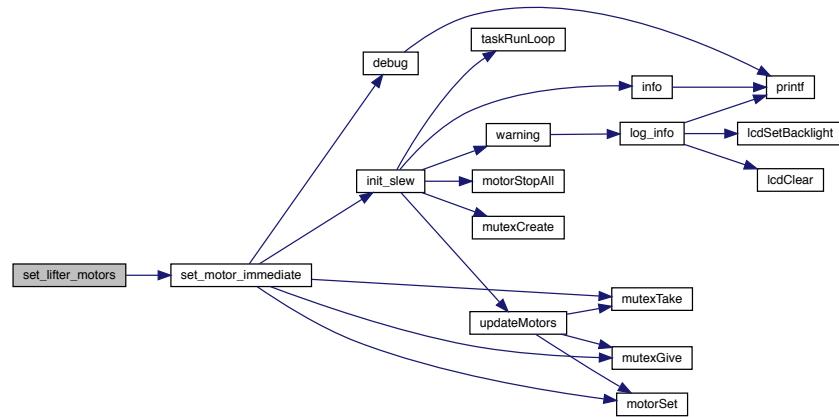
Referenced by autonomous(), lower_lifter(), raise_lifter(), and update_lifter().

```

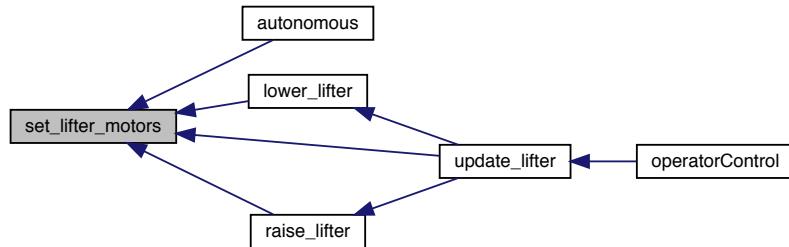
3
4     set_motor_immediate(MOTOR_LIFT_TOP_RIGHT, -v);
5     set_motor_immediate(MOTOR_LIFT_TOP_LEFT, -v);
6 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.33.1.7 set_lifter_pos()

```

void set_lifter_pos (
    int pos )

```

Definition at line 8 of file lifter.c.

```

8
9
10 }

```

7.33.1.8 update_lifter()

```
void update_lifter ( )
```

Definition at line 20 of file lifter.c.

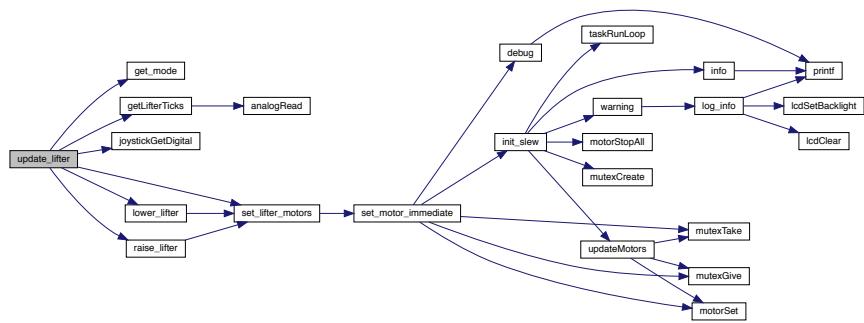
References get_mode(), getLifterTicks(), joystickGetDigital(), LIFTER_D, LIFTER_DOWN, LIFTER_DOWN_PARTNER, LIFTER_DRIVER_LOAD, LIFTER_I, LIFTER_P, LIFTER_UP, LIFTER_UP_PARTNER, lower_lifter(), MAIN_CONTROLLER_MODE, PARTNER_CONTROLLER_MODE, raise_lifter(), and set_lifter_motors().

Referenced by operatorControl().

```

20             {
21     static bool changed = true;
22     static unsigned int target = 0;
23     static bool first_run = true;
24     if(first_run) {
25         target = getLifterTicks();
26         first_run = false;
27     }
28     static int last_error = 0;
29     static long long i = 0;
30     if((joystickGetDigital(LIFTER_UP) && get_mode() == MAIN_CONTROLLER_MODE)
31         || (joystickGetDigital(LIFTER_UP_PARTNER) &&
32             get_mode() == PARTNER_CONTROLLER_MODE)) {
33         changed = true;
34         i = 0;
35         target = getLifterTicks() + 200;
36         lower_lifter();
37     }
38     else if((joystickGetDigital(LIFTER_DOWN) &&
39             get_mode() == MAIN_CONTROLLER_MODE)
40         || (joystickGetDigital(LIFTER_DOWN_PARTNER) &&
41             get_mode() == PARTNER_CONTROLLER_MODE)) {
42         changed = true;
43         i = 0;
44         target = getLifterTicks();
45         raise_lifter();
46     }
47     else if(joystickGetDigital(LIFTER_DRIVER_LOAD) &&
48             get_mode() == MAIN_CONTROLLER_MODE) {
49         changed = true;
50         i = 0;
51         int k = 0;
52         if(getLifterTicks() < 1270) {
53             lower_lifter();
54         }
55         if(getLifterTicks() > 1230) {
56             raise_lifter();
57             //k = 100;
58         }
59     }
60     else {
61         if(get_mode() == PARTNER_CONTROLLER_MODE) {
62             set_lifter_motors(0);
63             return;
64         }
65         int p = getLifterTicks() - target;
66         int d = p - last_error;
67         i += p;
68         int motor = LIFTER_P * p + LIFTER_D * d + LIFTER_I * i;
69         if (motor < 10) {
70             set_lifter_motors(0);
71         } else {
72             set_lifter_motors(motor);
73         }
74     }
75 }
```

Here is the call graph for this function:



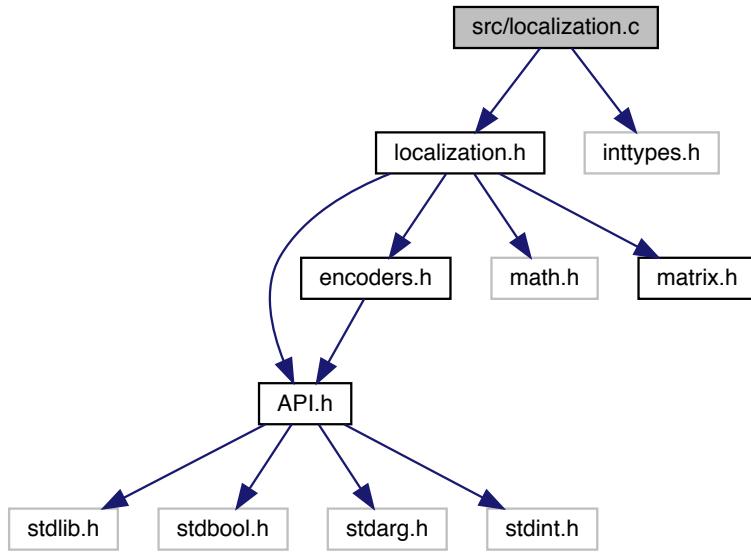
Here is the caller graph for this function:



7.34 src/localization.c File Reference

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

Include dependency graph for localization.c:



Data Structures

- struct `accelerometer_odometry`
- struct `encoder_odometry`

Functions

- static struct `accelerometer_odometry calculate_accelerometer_odometry ()`
- static double `calculate_angle ()`
- static double `calculate_gryo_angular_velocity ()`
- struct `location get_position ()`
- 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 ()`

Variables

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

7.34.1 Function Documentation

7.34.1.1 calculate_accelerometer_odometry()

```
static struct accelerometer_odometry calculate_accelerometer_odometry ( ) [static]
```

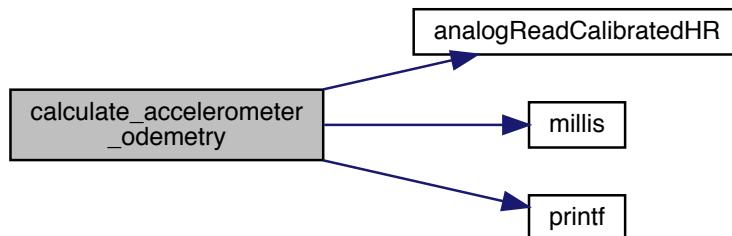
Definition at line 49 of file localization.c.

References analogReadCalibratedHR(), last_call, millis(), and printf().

Referenced by update_position().

```
49
50     static double vel_acumm_x = 0;
51     static double vel_acumm_y = 0;
52
53     int32_t accel_x_rel = (int32_t) analogReadCalibratedHR(2);
54     int32_t accel_y_rel = (int32_t) analogReadCalibratedHR(3);
55
56     //Ignore atom format string errors
57     printf("x: %+" PRId32 " y: %+" PRId32 "\n", accel_x_rel, accel_y_rel);
58
59     double delta_time = ((millis() - last_call)/1000.0);
60     //double accel_x_abs = (accel_x_rel * cos(theta) + accel_y_rel * sin(theta)) * delta_time;
61     //double accel_y_abs = (accel_y_rel * cos(theta) + accel_x_rel * sin(theta)) * delta_time;
62
63     //vel_acumm_x += accel_x_abs;
64     //vel_acumm_y += accel_y_abs;
65
66     //double new_x = x + vel_acumm_x * delta_time;
67     //double new_y = y + vel_acumm_y * delta_time;
68
69     struct accelerometer_odometry od;
70     //od.x = new_x;
71     //od.y = new_y;
72     return od;
73 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.34.1.2 calculate_angle()

```
static double calculate_angle ( ) [static]
```

7.34.1.3 calculate_gryo_angular_velocity()

```
static double calculate_gryo_angular_velocity ( ) [static]
```

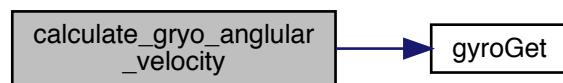
Definition at line 81 of file localization.c.

References g1, gyroGet(), and LOCALIZATION_UPDATE_FREQUENCY.

```

81
82     static int last_gyro = 0;
83     int current = gyroGet(g1);
84     // Calculate w (angular velocity in degrees per second)
85     double w = (current - last_gyro) / (LOCALIZATION_UPDATE_FREQUENCY/1000.0);
86     return w;
87 }
```

Here is the call graph for this function:



7.34.1.4 get_position()

```
struct location get_position ( )
```

Definition at line 25 of file localization.c.

```
25
26
27 }
```

7.34.1.5 init_localization()

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

Starts the localization process.

Parameters

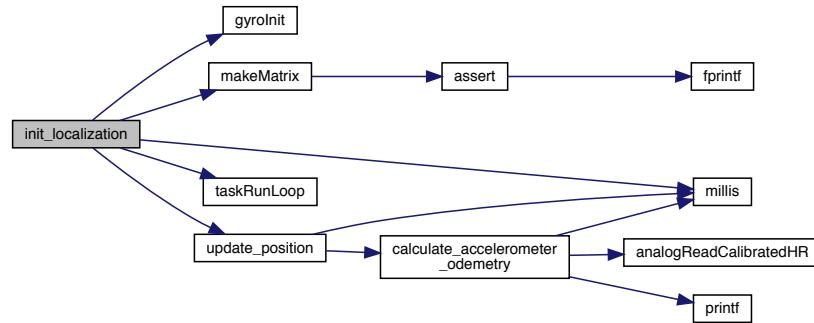
<i>gyro1</i>	The first gyro The multiplier parameter can tune the gyro to adapt to specific sensors. The default value at this time is 196; higher values will increase the number of degrees reported for a fixed actual rotation, while lower values will decrease the number of degrees reported. If your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

Definition at line 89 of file localization.c.

References g1, gyroInit(), last_call, localization_task, LOCALIZATION_UPDATE_FREQUENCY, makeMatrix(), millis(), taskRunLoop(), and update_position().

```
89
{
90     g1 = gyroInit(gyro1, multiplier);
91     //init state matrix
92
93     //one dimensional vector with x, y, theta, acceleration in x and y
94     state_matrix = makeMatrix(1, 5);
95     localization_task = taskRunLoop(update_position,
96                                     LOCALIZATION_UPDATE_FREQUENCY * 1000);
96     last_call = millis();
97     return true;
98 }
```

Here is the call graph for this function:



7.34.1.6 integrate_gyro_w()

```
static double integrate_gyro_w (
    int new_w ) [static]
```

Definition at line 75 of file localization.c.

References LOCALIZATION_UPDATE_FREQUENCY, and encoder_odemtry::theta.

```
75
76     static double theta = 0;
77     double delta_theta = new_w * LOCALIZATION_UPDATE_FREQUENCY;
78     theta += delta_theta;
79 }
```

7.34.1.7 update_position()

```
void update_position ( )
```

Definition at line 29 of file localization.c.

References calculate_accelerometer_odometry(), last_call, and millis().

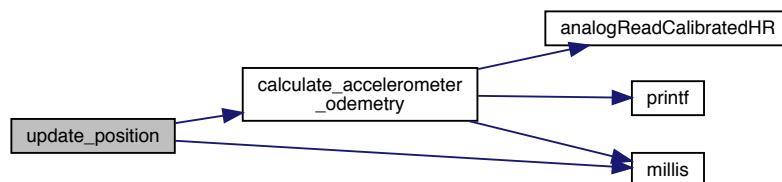
Referenced by init_localization().

```

29
30 //int curr_theta = calculate_angle();
31
32 struct accelerometer_odometry odem = 
    calculate_accelerometer_odometry();
33 //printf("x: %d y: %d T: %d\n", a.x, a.y, 0);
34
35 /*int l = 1;
36 int vr = get_encoder_velocity(1);
37 int vl = get_encoder_velocity(2);
38 int theta_dot = (vr - vl) / l;
39 int curr_theta = theta + theta_dot;
40 double dt = LOCALIZATION_UPDATE_FREQUENCY;
41 double v_tot = (vr+vl)/2.0;
42 int x_curr = x - v_tot*dt*sin(curr_theta);
43 int y_curr = y + v_tot*dt*cos(curr_theta);
44 x = x_curr;
45 y = y_curr; */
46 last_call = millis();
47 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.34.2 Variable Documentation

7.34.2.1 g1

`Gyro g1 [static]`

Definition at line 4 of file localization.c.

Referenced by `calculate_gryo-angular_velocity()`, and `init_localization()`.

7.34.2.2 last_call

```
int last_call = 0 [static]
```

Definition at line 7 of file localization.c.

Referenced by calculate_accelerometer_odometry(), init_localization(), and update_position().

7.34.2.3 localization_task

```
TaskHandle localization_task [static]
```

Definition at line 5 of file localization.c.

Referenced by init_localization().

7.34.2.4 state_matrix

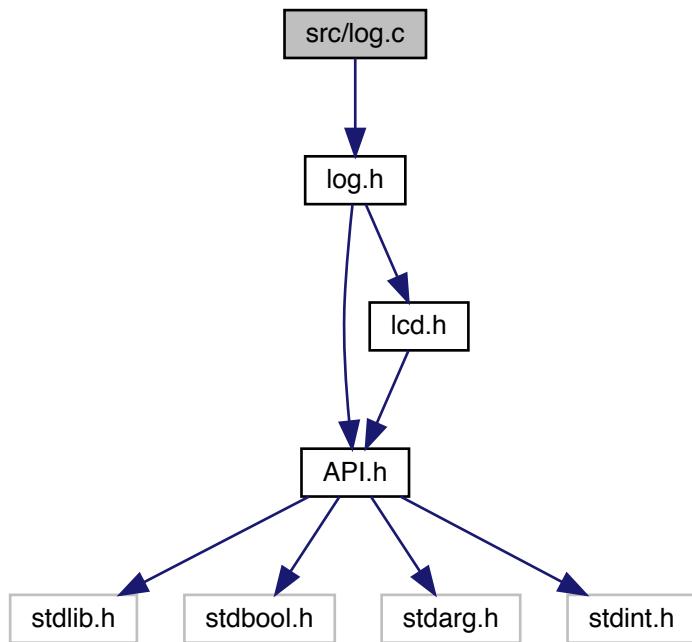
```
matrix* state_matrix
```

Definition at line 9 of file localization.c.

7.35 src/log.c File Reference

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

Include dependency graph for log.c:



Functions

- void **debug** (const char *debug_message)
prints a info message
- void **error** (const char *error_message)
prints a error message and displays on lcd. Only will print and display if log_level is greater than NONE
- void **info** (const char *info_message)
prints a info message
- void **init_error** (bool use_lcd, FILE *lcd)
Initializes the error lcd system Only required if using lcd.
- static void **log_info** (const char *s, const char *mess)
- void **warning** (const char *warning_message)
prints a warning message and displays on lcd. Only will print and display if log_level is greater than NONE

Variables

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

7.35.1 Function Documentation

7.35.1.1 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](#)

Parameters

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

Definition at line 37 of file log.c.

References ERROR, log_level, and printf().

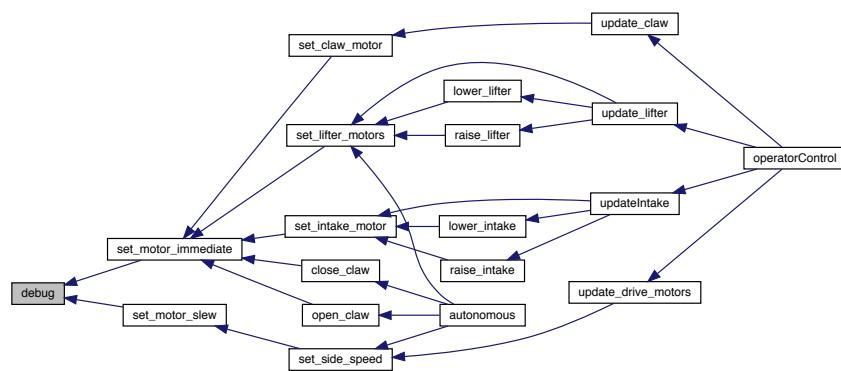
Referenced by set_motor_immediate(), and set_motor_slew().

```
37
38     if(log_level>ERROR) {
39         printf("[INFO]: %s\n", debug_message);
40     }
41 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.35.1.2 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](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

<code>error_message</code>	the message
----------------------------	-------------

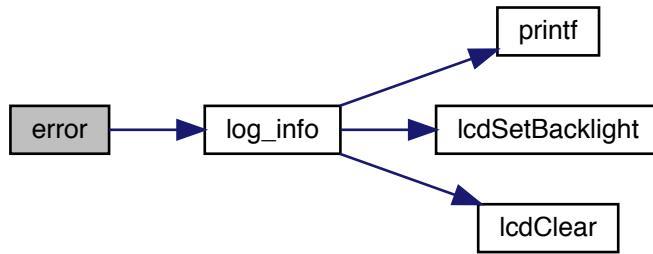
Definition at line 21 of file log.c.

References log_info(), log_level, and NONE.

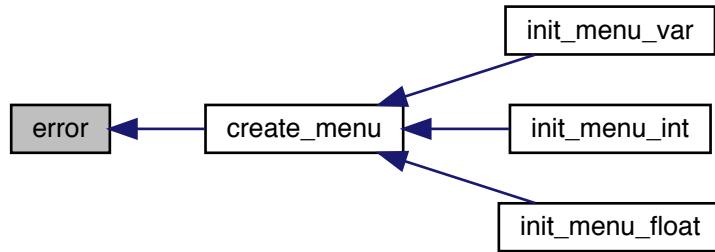
Referenced by create_menu().

```
21
22     if(log_level>NONE)
23         log_info("ERROR", error_message);
24 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.35.1.3 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](#)

Parameters

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

Definition at line 31 of file log.c.

References ERROR, log_level, and printf().

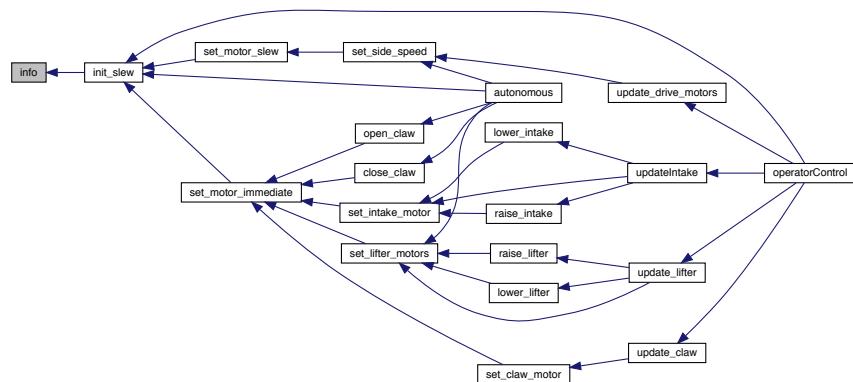
Referenced by init_slew().

```
31
32     if(log_level>ERROR) {
33         printf("[INFO]: %s\n", info_message);
34     }
35 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.35.1.4 init_error()

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

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

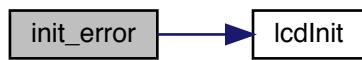
<i>use_lcd</i>	whether to use the lcd
<i>lcd</i>	the lcd

Definition at line 6 of file log.c.

References lcdInit(), and log_lcd.

```
6
7     if(use_lcd) {
8         lcdInit(lcd);
9         log_lcd = lcd;
10    }
11 }
```

Here is the call graph for this function:



7.35.1.5 log_info()

```
static void log_info (
    const char * s,
    const char * mess ) [static]
```

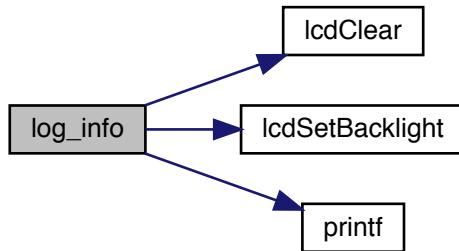
Definition at line 13 of file log.c.

References BOTTOM_ROW, lcdClear(), lcdSetBacklight(), log_lcd, printf(), and TOP_ROW.

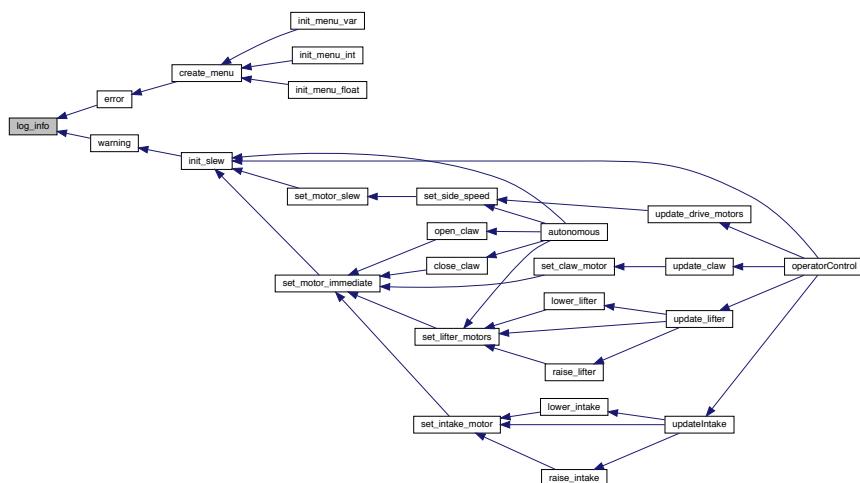
Referenced by error(), and warning().

```
13
14     printf("[%s]: %s\n", s, mess);
15     lcdSetBacklight(log_lcd, true);
16     lcdClear(log_lcd);
17     lcdPrint(log_lcd, TOP_ROW, s);
18     lcdPrint(log_lcd, BOTTOM_ROW, mess);
19 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.35.1.6 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](#)

Author

Chris Jerrett

Date

9/10/17

Parameters

<i>warning_message</i>	the message
------------------------	-------------

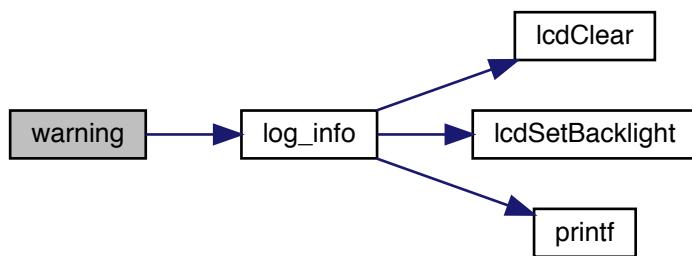
Definition at line 26 of file log.c.

References [log_info\(\)](#), [log_level](#), and [WARNING](#).

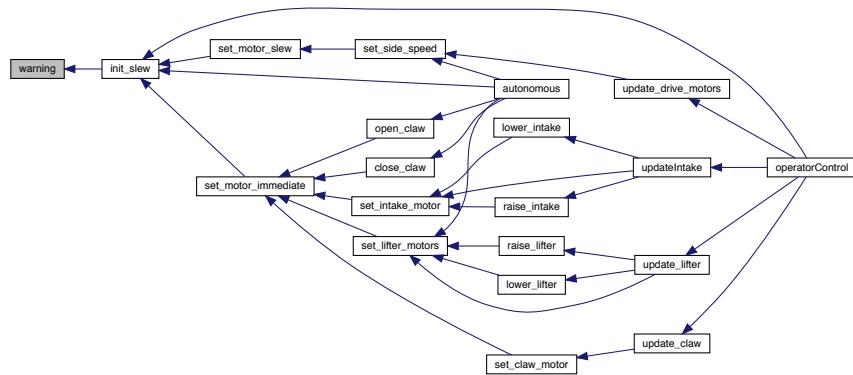
Referenced by [init_slew\(\)](#).

```
26
27     if(log_level>WARNING)
28         log_info("WARNING", warning_message);
29 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.35.2 Variable Documentation

7.35.2.1 log_lcd

```
FILE* log_lcd = NULL [static]
```

Definition at line 4 of file log.c.

Referenced by init_error(), and log_info().

7.35.2.2 log_level

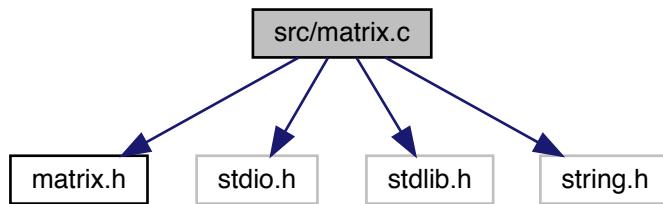
```
unsigned int log_level = INFO
```

Definition at line 3 of file log.c.

Referenced by debug(), error(), info(), and warning().

7.36 src/matrix.c File Reference

```
#include "matrix.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Include dependency graph for matrix.c:
```



Functions

- void [assert](#) (int assertion, char *message)
- matrix * [copyMatrix](#) (matrix *m)
- matrix * [covarianceMatrix](#) (matrix *m)
- matrix * [dotDiagonalMatrix](#) (matrix *a, matrix *b)
- matrix * [dotProductMatrix](#) (matrix *a, matrix *b)
- matrix * [eyeMatrix](#) (int n)
- void [freeMatrix](#) (matrix *m)
- matrix * [makeMatrix](#) (int width, int height)
- matrix * [meanMatrix](#) (matrix *m)
- matrix * [multiplyMatrix](#) (matrix *a, matrix *b)
- void [printMatrix](#) (matrix *m)
- void [rowSwap](#) (matrix *a, int p, int q)
- matrix * [scaleMatrix](#) (matrix *m, double value)
- double [traceMatrix](#) (matrix *m)
- matrix * [transposeMatrix](#) (matrix *m)

7.36.1 Function Documentation

7.36.1.1 assert()

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

assert 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 13 of file matrix.c.

References fprintf().

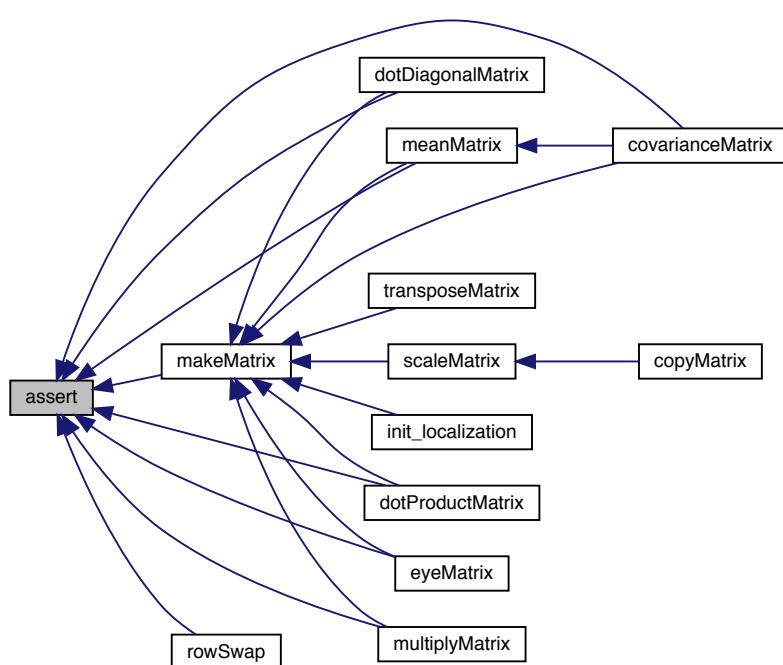
Referenced by covarianceMatrix(), dotDiagonalMatrix(), dotProductMatrix(), eyeMatrix(), makeMatrix(), meanMatrix(), multiplyMatrix(), and rowSwap().

```
13
14     if (assertion == 0) {
15         fprintf(stderr, "%s\n", message);
16         exit(1);
17     }
18 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.36.1.2 copyMatrix()

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

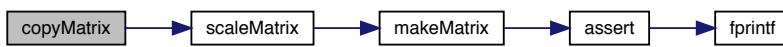
copyMatrix Copies a matrix. This function uses scaleMatrix, because scaling matrix by 1 is the same as a copy.

Definition at line 47 of file matrix.c.

References scaleMatrix().

```
47             {
48     return scaleMatrix(m, 1);
49 }
```

Here is the call graph for this function:



7.36.1.3 covarianceMatrix()

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

covarianceMatrix Given an "m rows by n columns" matrix, it returns a matrix with n row and n columns, where each element represents covariance of 2 columns.

Definition at line 164 of file matrix.c.

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

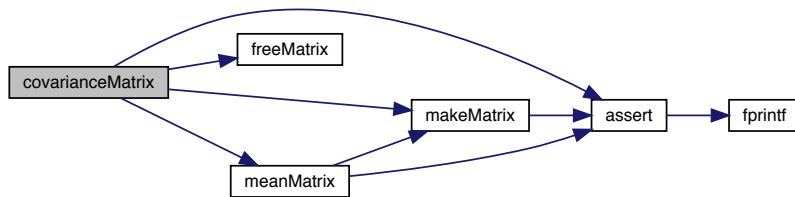
```
164
165     int i, j, k, l = 0;
166     matrix* out;
167     matrix* mean;
168     double* ptrA;
169     double* ptrB;
170     double* ptrOut;
171
172     assert(m->height > 1, "Height of matrix cannot be zero or one.");
173
174     mean = meanMatrix(m);
175     out = makeMatrix(m->width, m->width);
176     ptrOut = out->data;
177
178     for (i = 0; i < m->width; i++) {
179         for (j = 0; j < m->width; j++) {
180             ptrA = &m->data[i];
181             ptrB = &m->data[j];
182             *ptrOut = 0.0;
```

```

183         for (k = 0; k < m->height; k++) {
184             *ptrOut += (*ptrA - mean->data[i]) * (*ptrB - mean->data[j]);
185             ptrA += m->width;
186             ptrB += m->width;
187         }
188         *ptrOut /= m->height - 1;
189         ptrOut++;
190     }
191 }
192
193 freeMatrix(mean);
194 return out;
195 }

```

Here is the call graph for this function:



7.36.1.4 dotDiagonalMatrix()

```

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

```

`matrixDotDiagonal` 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 parameter is NULL, it is assumed that we are performing a cross product with itself.

Definition at line 367 of file matrix.c.

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

```

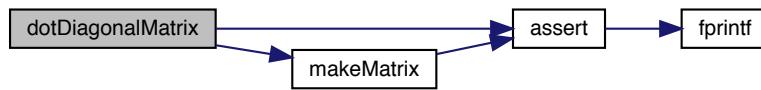
367
368     matrix* out;
369     double* ptrOut;
370     double* ptrA;
371     double* ptrB;
372     int i, j;
373
374     if (b != NULL) {
375         assert(a->width == b->width && a->height == b->
376 height, "Matrices must be of the same dimensionality.");
377     }
378
379     // Are we computing the sum of squares of the same matrix?
380     if (a == b || b == NULL) {
381         b = a; // May not appear safe, but we can do this without risk of losing b.
382     }

```

```

383     out = makeMatrix(1, a->height);
384     ptrOut = out->data;
385     ptrA = a->data;
386     ptrB = b->data;
387
388     for (i = 0; i < a->height; height; i++) {
389         *ptrOut = 0;
390         for (j = 0; j < a->width; j++) {
391             *ptrOut += *ptrA * *ptrB;
392             ptrA++;
393             ptrB++;
394         }
395         ptrOut++;
396     }
397
398     return out;
399 }
```

Here is the call graph for this function:



7.36.1.5 dotProductMatrix()

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

dotProductMatrix 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 parameter is NULL, it is assumed that we are performing a cross product with itself.

Definition at line 318 of file matrix.c.

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

```

318
319     matrix* out;
320     double* ptrOut;
321     double* ptrA;
322     double* ptrB;
323     int i, j, k;
324
325     if (b != NULL) {
326         assert(a->width == b->width, "Matrices must be of the same dimensionality.");
327     }
328
329     // Are we computing the sum of squares of the same matrix?
330     if (a == b || b == NULL) {
331         b = a; // May not appear safe, but we can do this without risk of losing b.
332     }
333 }
```

```

334     out = makeMatrix(b->height, a->height);
335     ptrOut = out->data;
336
337     for (i = 0; i < a->height; i++) {
338         ptrB = b->data;
339
340         for (j = 0; j < b->height; j++) {
341             ptrA = &a->data[ i * a->width ];
342
343             *ptrOut = 0;
344             for (k = 0; k < a->width; k++) {
345                 *ptrOut += *ptrA * *ptrB;
346                 ptrA++;
347                 ptrB++;
348             }
349             ptrOut++;
350         }
351     }
352
353     return out;
354 }
```

Here is the call graph for this function:



7.36.1.6 eyeMatrix()

```
matrix* eyeMatrix (
    int n )
```

eyeMatrix Returns an identity matrix of size n by n, where n is the input parameter.

Definition at line 90 of file matrix.c.

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

```

90
91     int i;
92     matrix *out;
93     double* ptr;
94
95     assert(n > 0, "Identity matrix must have value greater than zero.");
96
97     out = makeMatrix(n, n);
98     ptr = out->data;
99     for (i = 0; i < n; i++) {
100         *ptr = 1.0;
101         ptr += n + 1;
102     }
103
104     return out;
105 }
```

Here is the call graph for this function:



7.36.1.7 freeMatrix()

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

freeMatrix Frees the resources of a matrix

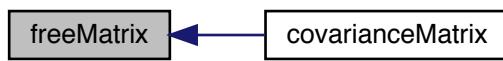
Definition at line 55 of file matrix.c.

References `_matrix::data`.

Referenced by covarianceMatrix().

```
55
56     if (m != NULL) {
57         if (m->data != NULL) {
58             free(m->data);
59             m->data = NULL;
60         }
61
62         free(m);
63         m = NULL;
64     }
65     return;
66 }
```

Here is the caller graph for this function:



7.36.1.8 makeMatrix()

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

`makeMatrix` Makes a matrix with a width and height parameters.

Definition at line 24 of file `matrix.c`.

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

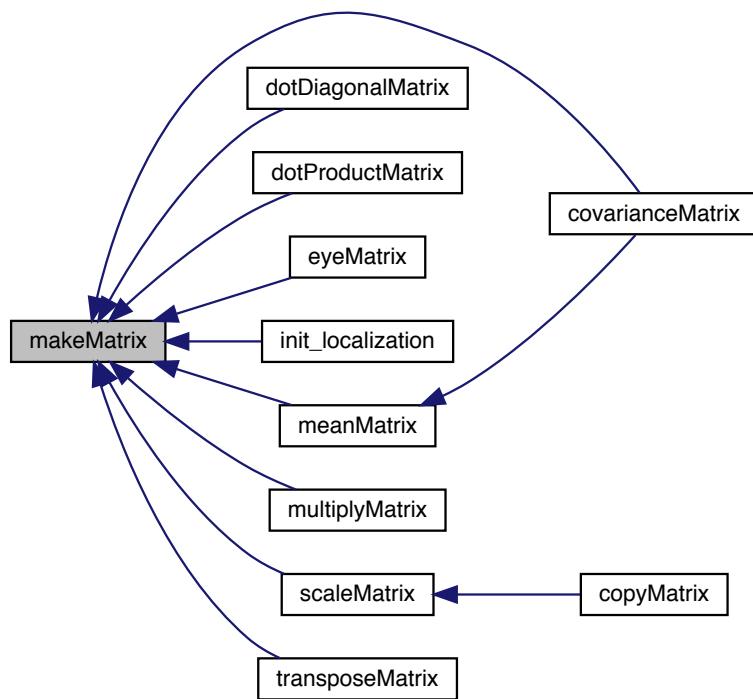
Referenced by `covarianceMatrix()`, `dotDiagonalMatrix()`, `dotProductMatrix()`, `eyeMatrix()`, `init_localization()`, `mean←Matrix()`, `multiplyMatrix()`, `scaleMatrix()`, and `transposeMatrix()`.

```
24
25     matrix* out;
26     assert(width > 0 && height > 0, "New matrix must be at least a 1 by 1");
27     out = (matrix*) malloc(sizeof(matrix));
28
29     assert(out != NULL, "Out of memory.");
30
31     out->width = width;
32     out->height = height;
33     out->data = (double*) malloc(sizeof(double) * width * height);
34
35     assert(out->data != NULL, "Out of memory.");
36
37     memset(out->data, 0.0, width * height * sizeof(double));
38
39     return out;
40 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.36.1.9 meanMatrix()

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

meanMatrix Given an "m rows by n columns" matrix, it returns a matrix with 1 row and n columns, where each element represents the mean of that full column.

Definition at line 138 of file matrix.c.

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

Referenced by covarianceMatrix().

```

138      {
139      int i, j;
140      double* ptr;
141      matrix* out;
142
143      assert(m->height > 0, "Height of matrix cannot be zero.");
144
145      out = makeMatrix(m->width, 1);
146
147      for (i = 0; i < m->width; i++) {
148          out->data[i] = 0.0;
149          ptr = &m->data[i];

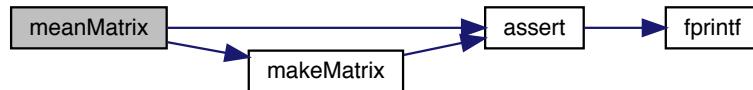
```

```

150     for (j = 0; j < m->height; j++) {
151         out->data[i] += *ptr;
152         ptr += out->width;
153     }
154     out->data[i] /= (double) m->height;
155 }
156 return out;
157 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.36.1.10 multiplyMatrix()

```

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

```

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

Definition at line 223 of file `matrix.c`.

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

```

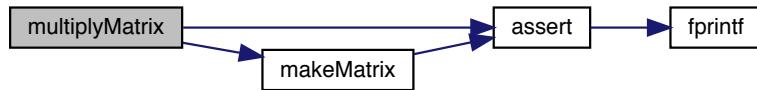
223
224     int i, j, k;
225     matrix* out;
226     double* ptrOut;
227     double* ptrA;
228     double* ptrB;
229
230     assert(a->width == b->height, "Matrices have incorrect dimensions. a->width != b->height");
231
232     out = makeMatrix(b->width, a->height);
233     ptrOut = out->data;

```

```

234     for (i = 0; i < a->height; i++) {
235         for (j = 0; j < b->width; j++) {
236             ptrA = &a->data[ i * a->width ];
237             ptrB = &b->data[ j ];
238
239             *ptrOut = 0;
240             for (k = 0; k < a->width; k++) {
241                 *ptrOut += *ptrA * *ptrB;
242                 ptrA++;
243                 ptrB += b->width;
244             }
245             ptrOut++;
246         }
247     }
248     return out;
249 }
250
251 }
```

Here is the call graph for this function:



7.36.1.11 printMatrix()

```

void printMatrix (
    matrix * m )
  
```

`printMatrix` Prints a matrix. Great for debugging.

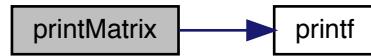
Definition at line 72 of file matrix.c.

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

```

72
73     int i, j;
74     double* ptr = m->data;
75     printf("%d %d\n", m->width, m->height);
76     for (i = 0; i < m->height; i++) {
77         for (j = 0; j < m->width; j++) {
78             printf(" %9.6f", *(ptr++));
79         }
80         printf("\n");
81     }
82     return;
83 }
```

Here is the call graph for this function:



7.36.1.12 rowSwap()

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

rowSwap 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.

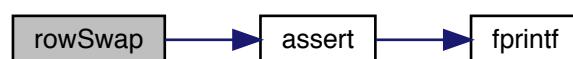
Definition at line 278 of file matrix.c.

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

```

278
279     int i;
280     double temp;
281     double* pRow;
282     double* qRow;
283
284     assert(a->height > 2, "Matrix must have at least two rows to swap.");
285     assert(p < a->height && q < a->height, "Values p and q must be less than the height of the
286     matrix.");
287
288     // If p and q are equal, do nothing.
289     if (p == q) {
290         return;
291     }
292
293     pRow = a->data + (p * a->width);
294     qRow = a->data + (q * a->width);
295
296     // Swap!
297     for (i = 0; i < a->width; i++) {
298         temp = *pRow;
299         *pRow = *qRow;
300         *qRow = temp;
301         pRow++;
302         qRow++;
303     }
304
305 }
```

Here is the call graph for this function:



7.36.1.13 scaleMatrix()

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

scaleMatrix Given a matrix and a double value, this returns a new matrix where each element in the input matrix is multiplied by the double value

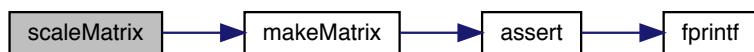
Definition at line 259 of file matrix.c.

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

Referenced by `copyMatrix()`.

```
259     int i, elements = m->width * m->height;
260     matrix* out = makeMatrix(m->width, m->height);
261     double* ptrM = m->data;
262     double* ptrOut = out->data;
263
264     for (i = 0; i < elements; i++) {
265         *(ptrOut++) = *(ptrM++) * value;
266     }
267
268     return out;
269 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.36.1.14 traceMatrix()

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

traceMatrix Given an "m rows by n columns" matrix, it returns the sum of the elements along the diagonal. This is known as the matrix 'trace'.

Definition at line 112 of file matrix.c.

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

```
112
113     int i;
114     int size;
115     double* ptr = m->data;
116     double sum = 0.0;
117
118     if (m->height < m->width) {
119         size = m->height;
120     }
121     else {
122         size = m->width;
123     }
124
125     for (i = 0; i < size; i++) {
126         sum += *ptr;
127         ptr += m->width + 1;
128     }
129
130     return sum;
131 }
```

7.36.1.15 transposeMatrix()

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

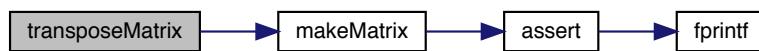
transposeMatrix Given an matrix, returns the transpose.

Definition at line 201 of file matrix.c.

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

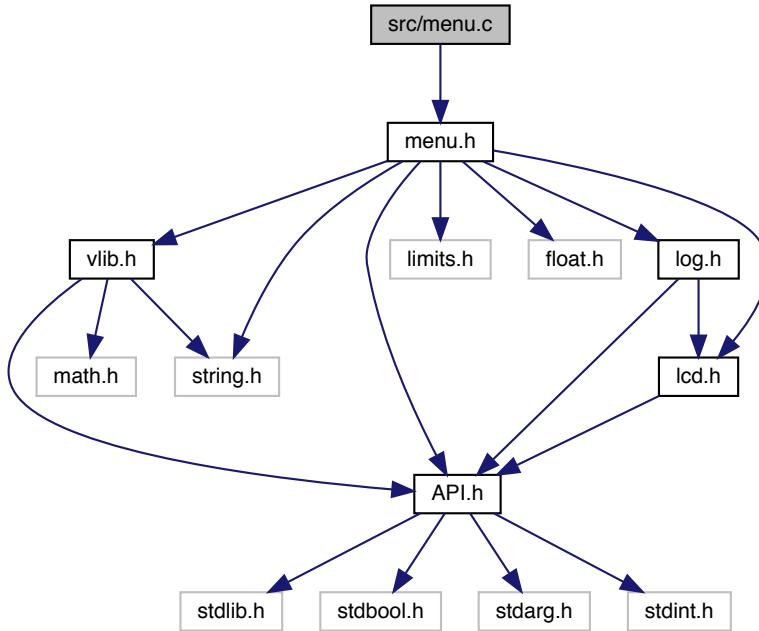
```
201
202     matrix* out = makeMatrix(m->height, m->width);
203     double* ptrOut;
204     double* ptrM = m->data;
205     int i, j;
206
207     for (i = 0; i < m->height; i++) {
208         ptrOut = &out->data[i];
209         for (j = 0; j < m->width; j++) {
210             *ptrOut = *ptrM;
211             ptrM++;
212             ptrOut += out->width;
213         }
214     }
215
216     return out;
217 }
```

Here is the call graph for this function:



7.37 src/menu.c File Reference

```
#include "menu.h"
Include dependency graph for menu.c:
```



Functions

- static void [calculate_current_display](#) (char *rtn, [menu_t](#) *menu)
- static [menu_t](#) * [create_menu](#) (enum [menu_type](#) type, const char *prompt)
- void [denint_menu](#) ([menu_t](#) *menu)

Destroys a menu. Menu must be freed or will cause memory leak
- int [display_menu](#) ([menu_t](#) *menu)

Displays a menu context, but does not display. 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, unsigned int nums, const char *prompt, char *options,...)

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

7.37.1 Function Documentation

7.37.1.1 calculate_current_display()

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

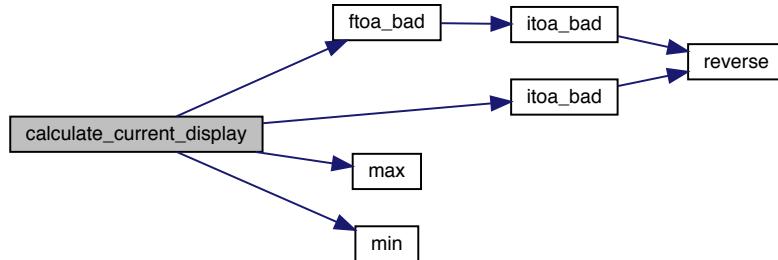
Definition at line 56 of file menu.c.

References menu_t::current, FLOAT_TYPE, ftoa_bad(), INT_TYPE, itoa_bad(), 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().

```
56
57     if(menu->type == STRING_TYPE) {
58         //Ignore warning
59         rtn = (menu->options[menu->current % (menu->length)]);
60     }
61     if(menu->type == INT_TYPE) {
62         int step = (menu->step);
63         int min = (menu->min);
64         int max = (menu->max);
65         int value = menu->current * step;
66         value = value < min ? min : value;
67         value = value > max ? max : value;
68         itoa_bad(value, rtn, 4);
69     }
70     if(menu->type == FLOAT_TYPE) {
71         float step = (menu->step_f);
72         float min = (menu->min_f);
73         float max = (menu->max_f);
74         float value = menu->current * step;
75         value = value < min ? min : value;
76         value = value > max ? max : value;
77
78         ftoa_bad(value, rtn, 5);
79     }
80 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.37.1.2 create_menu()

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

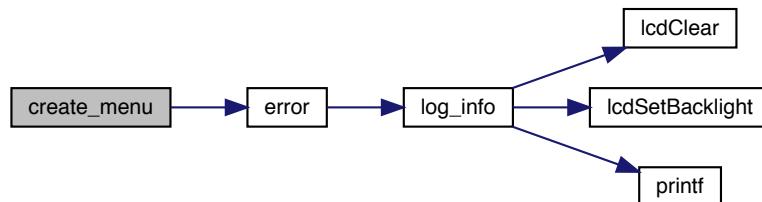
Definition at line 6 of file menu.c.

References 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.

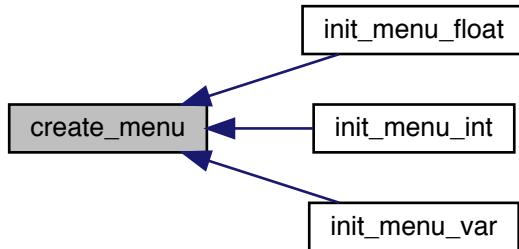
Referenced by init_menu_float(), init_menu_int(), and init_menu_var().

```
6
7  menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
8  if (!menu) {
9      error("Menu Malloc");
10 }
11 menu->type = type;
12 // Add one for null terminator
13 size_t strlength = strlen(prompt) + 1;
14 menu->prompt = (char*) malloc(strlength * sizeof(char));
15 memcpy(menu->prompt, prompt, strlength);
16 menu->max = INT_MAX;
17 menu->min = INT_MIN;
18 menu->step = 1;
19 menu->min_f = FLT_MIN;
20 menu->max_f = FLT_MAX;
21 menu->step_f = 1;
22
23 return menu;
24 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.37.1.3 `denint_menu()`

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

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

Parameters

<code>menu</code>	the menu to free
-------------------	------------------

See also

`menu`

Author

Chris Jerrett

Date

9/8/17

Definition at line 101 of file `menu.c`.

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

```
101
102     free(menu->prompt);
103     if(menu->options != NULL) free(menu->options);
104     free(menu);
105 }
```

7.37.1.4 display_menu()

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

Displays a menu context, but does not display. *Menu must be freed or will cause memory leak! Will exit if robot is enabled. This prevents menu from locking up system in even of a reset.*

Parameters

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

See also

[menu_type](#)

Author

Chris Jerrett

Date

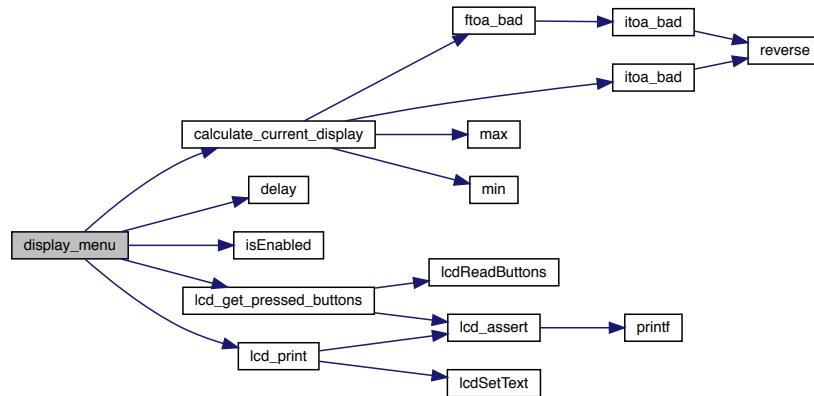
9/8/17

Definition at line 83 of file menu.c.

References calculate_current_display(), menu_t::current, delay(), isEnabled(), lcd_get_pressed_buttons(), lcd←print(), PRESSED, menu_t::prompt, RELEASED, and TOP_ROW.

```
83     {
84     lcd_print(TOP_ROW, menu->prompt);
85     //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
86     while(lcd_get_pressed_buttons().middle == RELEASED && !
87         isEnabled()) {
87     char val[16];
88     calculate_current_display(val, menu);
89
90     if(lcd_get_pressed_buttons().right == PRESSED) {
91         menu->current += 1;
92     }
93     if(lcd_get_pressed_buttons().left == PRESSED) {
94         menu->current -= 1;
95     }
96     delay(500);
97 }
98 return menu->current;
99 }
```

Here is the call graph for this function:



7.37.1.5 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

<code>type</code>	the type of menu
-------------------	------------------

See also

[menu_type](#)

Parameters

<code>min</code>	the minimum value
<code>max</code>	the maximum value
<code>step</code>	the step value
<code>prompt</code>	the prompt to display to user

Author

Chris Jerrett

Date

9/8/17

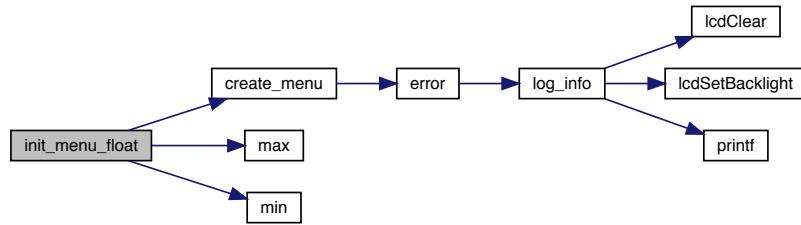
Definition at line 48 of file menu.c.

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

```

48
49     menu_t* menu = create_menu(type, prompt);
50     menu->min_f = min;
51     menu->max_f = max;
52     menu->step_f = step;
53     return menu;
54 }
```

Here is the call graph for this function:



7.37.1.6 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

<code>type</code>	the type of menu
-------------------	------------------

See also

[menu_type](#)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

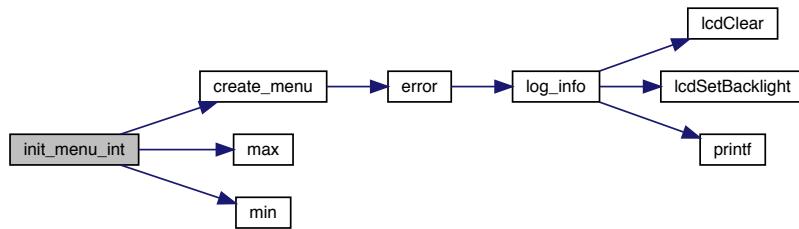
Definition at line 40 of file menu.c.

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

```

40
41     menu_t* menu = create_menu(type, prompt);
42     menu->min = min;
43     menu->max = max;
44     menu->step = step;
45     return menu;
46 }
```

Here is the call graph for this function:

**7.37.1.7 init_menu_var()**

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

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

Parameters

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

See also

[menu_type](#)

Parameters

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

Author

Chris Jerrett

Date

9/8/17

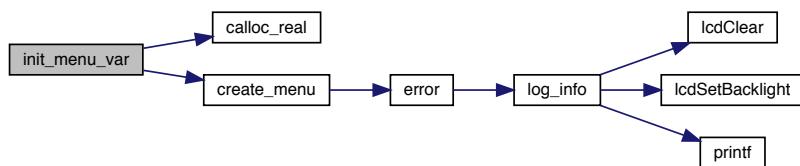
Definition at line 26 of file menu.c.

References `calloc_real()`, `create_menu()`, `menu_t::length`, and `menu_t::options`.

```

26
27   menu_t* menu = create\_menu(type, prompt);
28   va_list values;
29   char **options_array = (char**) calloc\_real(sizeof\(char\*\), nums);
30   va_start(values, options);
31   for(unsigned int i = 0; i < nums; i++){
32     options_array[i] = va_arg(values, char*);
33   }
34   va_end(values);
35   menu->options = options_array;
36   menu->length = nums;
37   return menu;
38 }
```

Here is the call graph for this function:

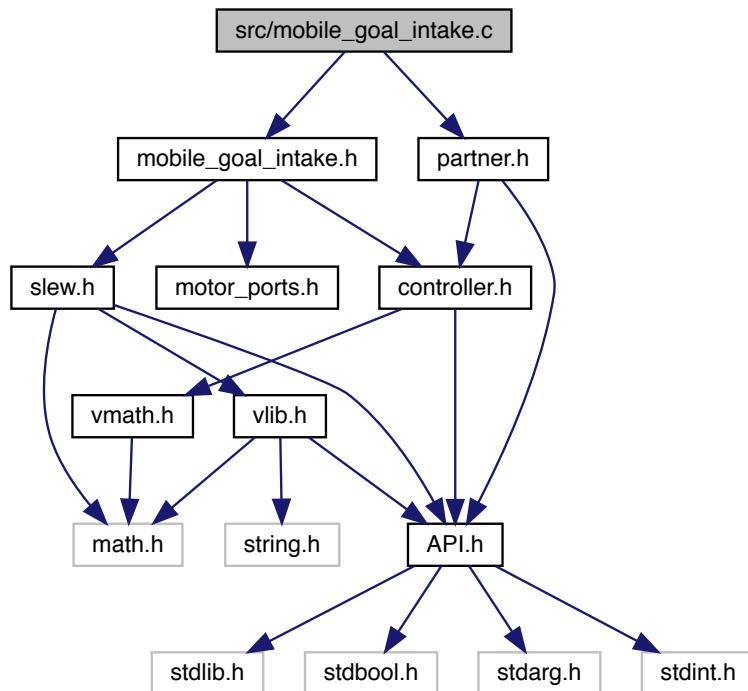


7.38 src/mobile_goal_intake.c File Reference

```
#include "mobile_goal_intake.h"
```

```
#include "partner.h"
```

Include dependency graph for mobile_goal_intake.c:



Functions

- static void [lower_intake \(\)](#)
- static void [raise_intake \(\)](#)
- static void [set_intake_motor \(int n\)](#)
- void [updateIntake \(\)](#)

7.38.1 Function Documentation

7.38.1.1 [lower_intake\(\)](#)

```
static void lower_intake ( ) [static]
```

Definition at line 8 of file `mobile_goal_intake.c`.

References `set_intake_motor()`.

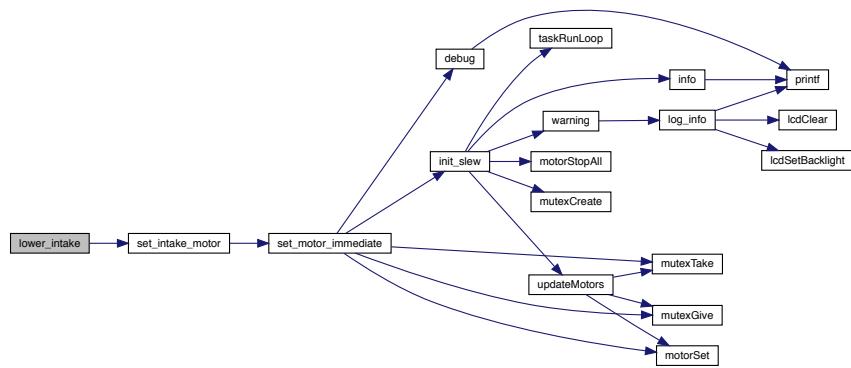
Referenced by `updateIntake()`.

```

8
9     set_intake_motor(100);
10 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.38.1.2 raise_intake()

```
static void raise_intake( ) [static]
```

Definition at line 12 of file mobile_goal_intake.c.

References set_intake_motor().

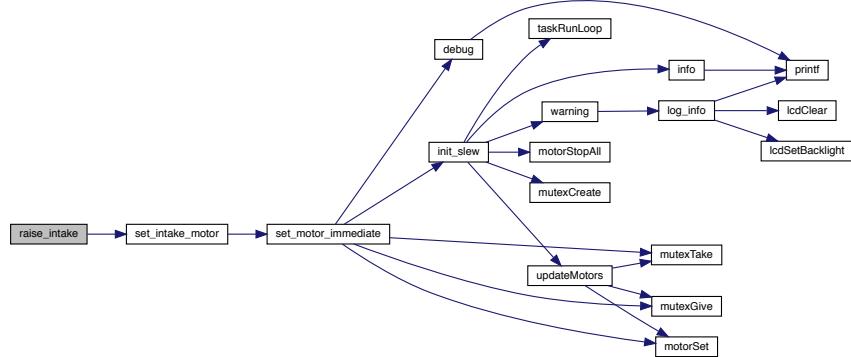
Referenced by updateIntake().

```

12
13     set_intake_motor(-100);
14 }

```

Here is the call graph for this function:



Here is the caller graph for this function:



7.38.1.3 set_intake_motor()

```
static void set_intake_motor (
    int n )  [static]
```

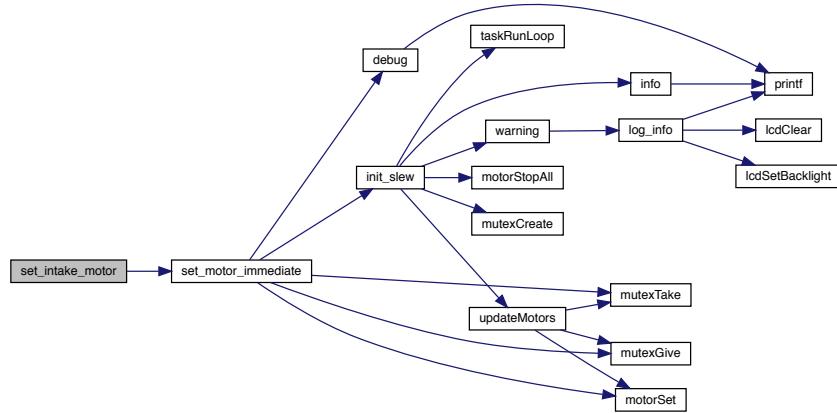
Definition at line 4 of file mobile_goal_intake.c.

References INTAKE_MOTOR, and set_motor_immediate().

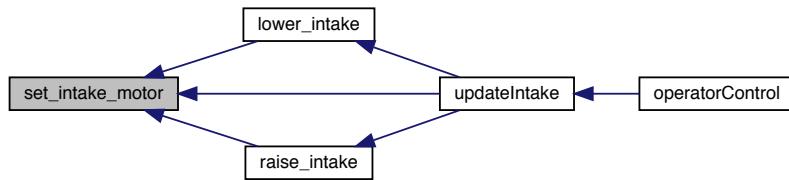
Referenced by lower_intake(), raise_intake(), and updateIntake().

```
4
5     set_motor_immediate(INTAKE_MOTOR, n);
6 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.38.1.4 updateIntake()

```
void updateIntake ( )
```

Definition at line 16 of file mobile_goal_intake.c.

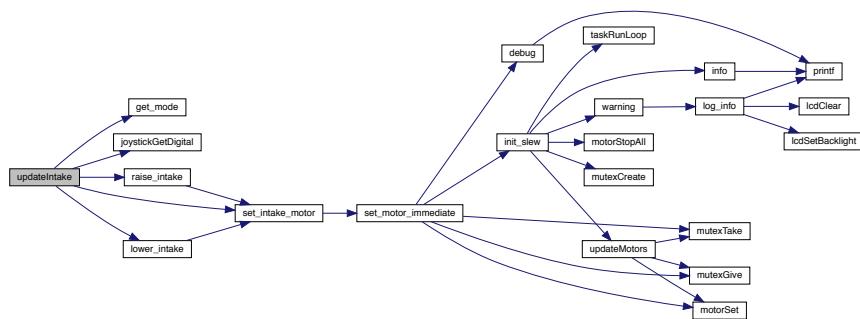
References get_mode(), JOY_DOWN, JOY_UP, joystickGetDigital(), lower_intake(), MAIN_CONTROLLER_MODE, MASTER, PARTNER, PARTNER_CONTROLLER_MODE, raise_intake(), and set_intake_motor().

Referenced by operatorControl().

```

16
17     if(joystickGetDigital(MASTER, 7, JOY_UP) &&
18         get_mode() == MAIN_CONTROLLER_MODE)
19     || joystickGetDigital(PARTNER, 6, JOY_UP) &&
20         get_mode() == PARTNER_CONTROLLER_MODE) {
21         raise_intake();
22     }
23     else if(joystickGetDigital(MASTER, 7, JOY_DOWN) &&
24         get_mode() == MAIN_CONTROLLER_MODE)
25     || joystickGetDigital(PARTNER, 6, JOY_DOWN) &&
26         get_mode() == PARTNER_CONTROLLER_MODE) {
27         lower_intake();
28     }
29     else set_intake_motor(0);
30 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



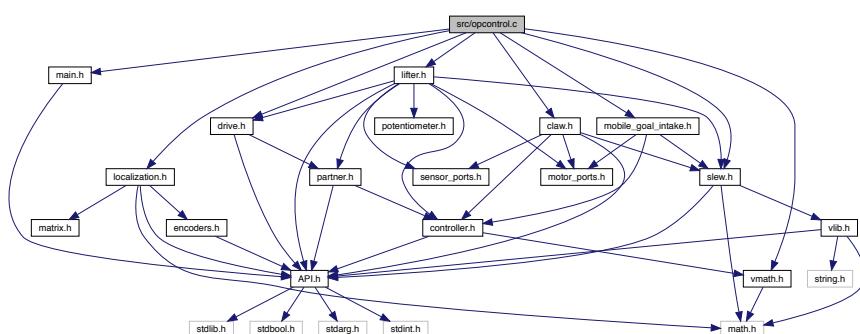
7.39 src/opcontrol.c File Reference

File for operator control code.

```

#include "main.h"
#include "slew.h"
#include "drive.h"
#include "lifter.h"
#include "localization.h"
#include "claw.h"
#include "mobile_goal_intake.h"
#include "vmath.h"

Include dependency graph for opcontrol.c:
  
```



Functions

- void [operatorControl \(\)](#)

7.39.1 Detailed Description

File for operator control code.

This file should contain the user [operatorControl\(\)](#) 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.

7.39.2 Function Documentation

7.39.2.1 [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.

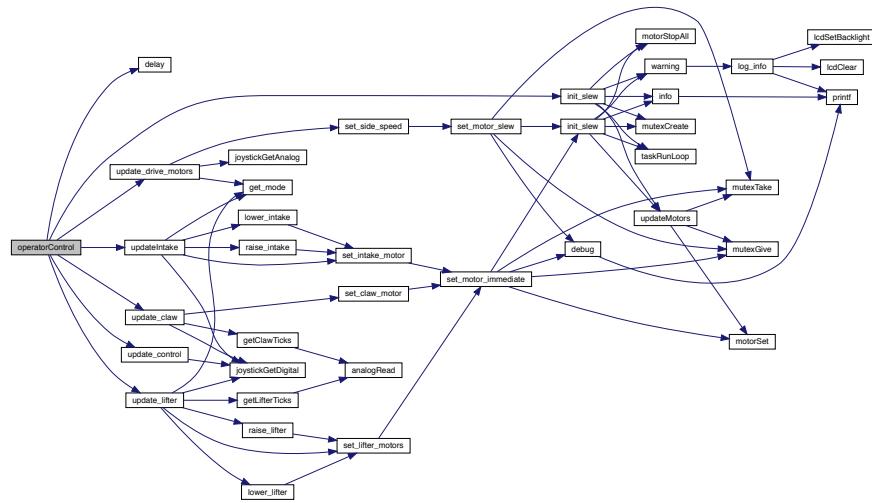
This task should never exit; it should end with some kind of infinite loop, even if empty.

Definition at line 40 of file opcontrol.c.

References [delay\(\)](#), [init_slew\(\)](#), [update_claw\(\)](#), [update_control\(\)](#), [update_drive_motors\(\)](#), [update_lifter\(\)](#), and [updateIntake\(\)](#).

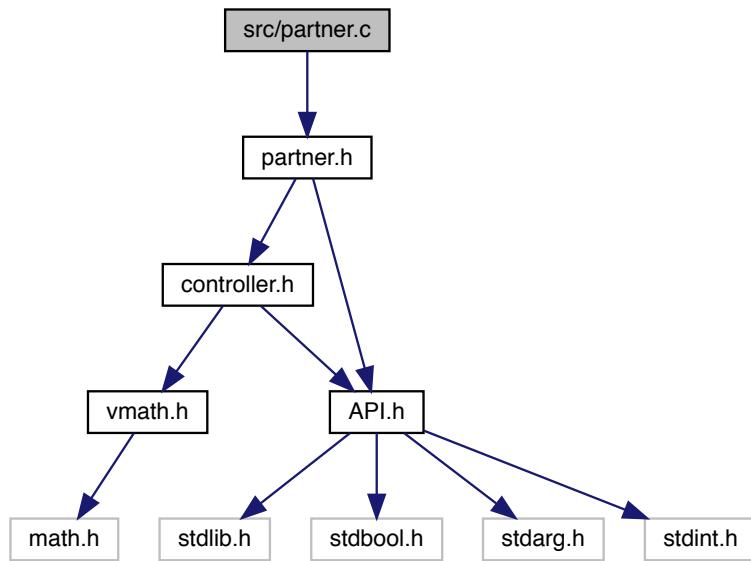
```
40      {
41      init_slew();
42      delay(10);
43      while (1) {
44          update_drive_motors();
45          update_lifter();
46          update_claw();
47          updateIntake();
48          update_control();
49          delay(25);
50      }
51 }
```

Here is the call graph for this function:



7.40 src/partner.c File Reference

```
#include "partner.h"
Include dependency graph for partner.c:
```



Functions

- enum `CONTROL_MODE` `get_mode ()`
- void `update_control ()`

Variables

- static enum `CONTROL_MODE` mode = `MAIN_CONTROLLER_MODE`

7.40.1 Function Documentation

7.40.1.1 `get_mode()`

```
enum CONTROL_MODE get_mode ( )
```

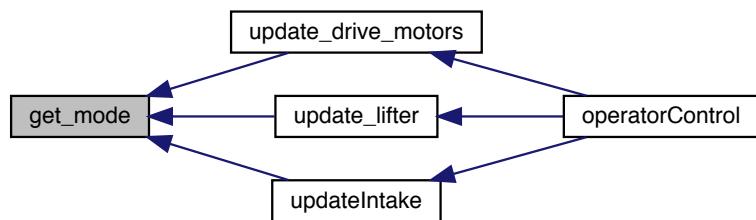
Definition at line 5 of file partner.c.

References mode.

Referenced by `update_drive_motors()`, `update_lifter()`, and `updateIntake()`.

```
5           {  
6     return mode;  
7 }
```

Here is the caller graph for this function:



7.40.1.2 update_control()

```
void update_control ( )
```

Definition at line 9 of file partner.c.

References JOY_LEFT, JOY_RIGHT, joystickGetDigital(), MAIN_CONTROLLER_MODE, mode, PARTNER, and PARTNER_CONTROLLER_MODE.

Referenced by operatorControl().

```
9      {
10     if(joystickGetDigital(PARTNER, 7, JOY_LEFT)) {
11       mode = MAIN_CONTROLLER_MODE;
12     } else if(joystickGetDigital(PARTNER, 7, JOY_RIGHT)) {
13       mode = PARTNER_CONTROLLER_MODE;
14     }
15 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.40.2 Variable Documentation

7.40.2.1 mode

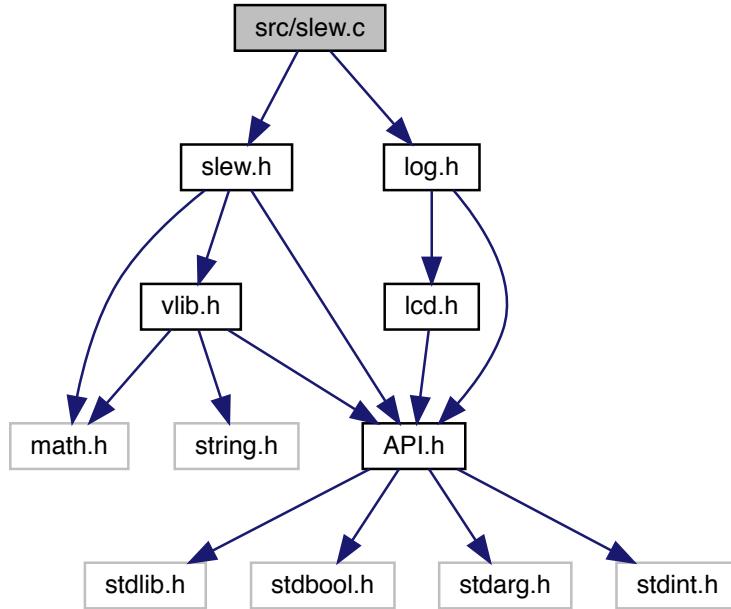
```
enum CONTROL_MODE mode = MAIN_CONTROLLER_MODE [static]
```

Definition at line 3 of file partner.c.

Referenced by get_mode(), and update_control().

7.41 src/slew.c File Reference

```
#include "slew.h"
#include "log.h"
Include dependency graph for slew.c:
```



Functions

- void [deinitSlew \(\)](#)
Deinitializes the slew rate controller and frees memory.
- void [init_slew \(\)](#)
Initializes the slew rate controller.
- void [set_motor_immediate](#) (int motor, int speed)
- 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](#)

7.41.1 Function Documentation

7.41.1.1 deinitsllew()

```
void deinitsllew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line 43 of file slew.c.

References initialized, motors_curr_speeds, motors_set_speeds, slew, and taskDelete().

Referenced by autonomous().

```
43     {
44     taskDelete(slew);
45     memset(motors_set_speeds, 0, sizeof(int) * 10);
46     memset(motors_curr_speeds, 0, sizeof(int) * 10);
47     initialized = false;
48 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.41.1.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

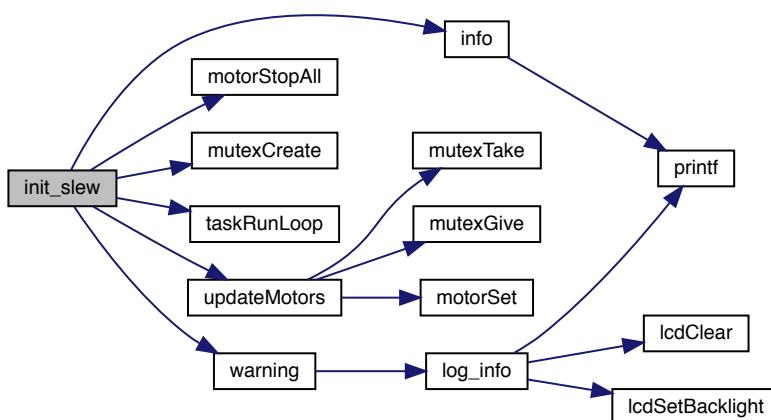
Definition at line 30 of file slew.c.

References info(), initialized, motors_curr_speeds, motors_set_speeds, motorStopAll(), mutexCreate(), slew, speeds_mutex, taskRunLoop(), updateMotors(), and warning().

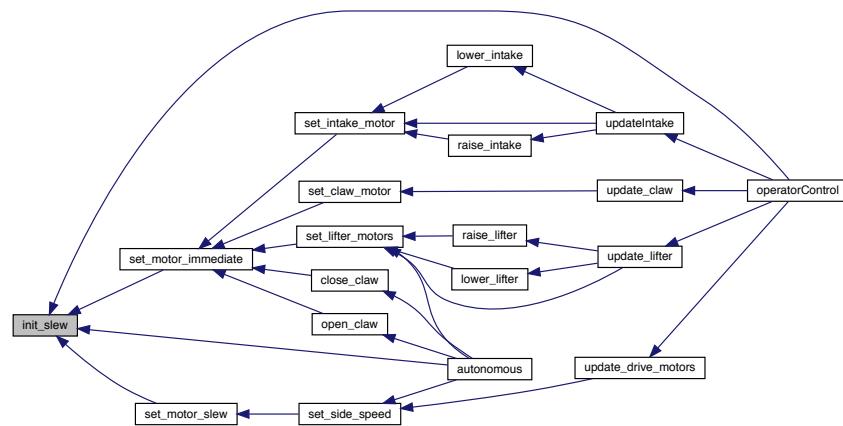
Referenced by autonomous(), operatorControl(), set_motor_immediate(), and set_motor_slew().

```
30      {
31      if(initialized) {
32          warning("Trying to init already init slew");
33      }
34      memset(motors_set_speeds, 0, sizeof(int) * 10);
35      memset(motors_curr_speeds, 0, sizeof(int) * 10);
36      motorStopAll();
37      info("Did Init Slew");
38      speeds_mutex = mutexCreate();
39      slew = taskRunLoop(updateMotors, 100);
40      initialized = true;
41 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.41.1.3 `set_motor_immediate()`

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

Definition at line 60 of file `slew.c`.

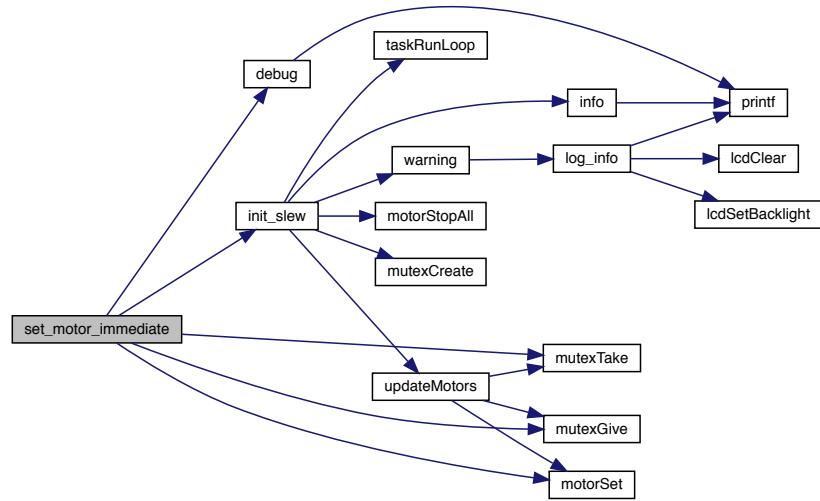
References `debug()`, `init_slew()`, `initialized`, `motors_curr_speeds`, `motors_set_speeds`, `motorSet()`, `mutexGive()`, `mutexTake()`, and `speeds_mutex`.

Referenced by `close_claw()`, `open_claw()`, `set_claw_motor()`, `set_intake_motor()`, and `set_lifter_motors()`.

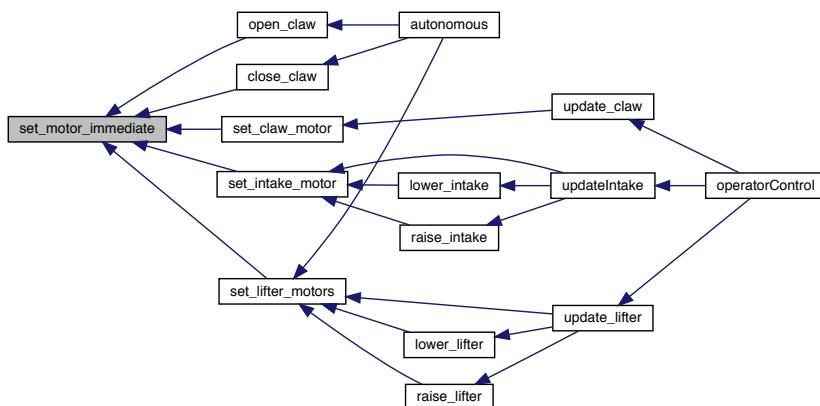
```

60
61     if(!initialized) {
62         debug("Slew Not Initialized! Initializing");
63         init_slew();
64     }
65     motorSet(motor, speed);
66     mutexTake(speeds_mutex, 10);
67     motors_curr_speeds[motor-1] = speed;
68     motors_set_speeds[motor-1] = speed;
69     mutexGive(speeds_mutex);
70 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.41.1.4 `set_motor_slew()`

```

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

Sets motor speed wrapped inside the slew rate controller.

Parameters

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

Author

Chris Jerrett

Date

9/14/17

Definition at line 50 of file slew.c.

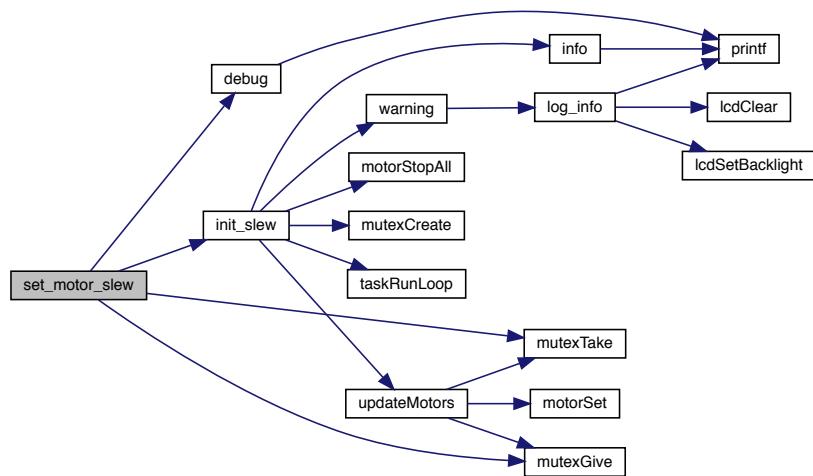
References debug(), init_slew(), initialized, motors_set_speeds, mutexGive(), mutexTake(), and speeds_mutex.

Referenced by set_side_speed().

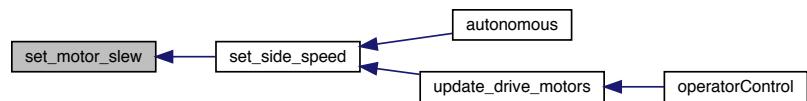
```

50
51 if(!initialized) {
52     debug("Slew Not Initialized! Initializing");
53     init_slew();
54 }
55 mutexTake(speeds_mutex, 10);
56 motors_set_speeds[motor-1] = speed;
57 mutexGive(speeds_mutex);
58 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.41.1.5 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

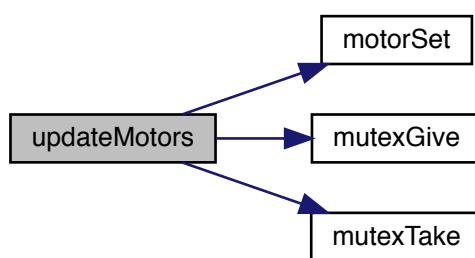
Definition at line 13 of file slew.c.

References motors_curr_speeds, motors_set_speeds, motorSet(), mutexGive(), mutexTake(), and speeds_mutex.

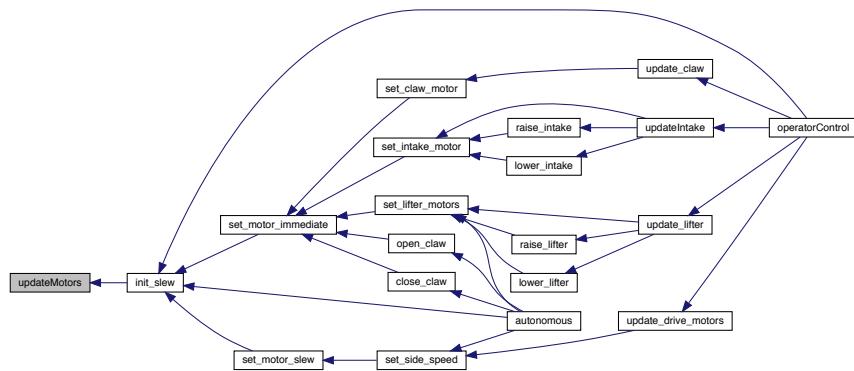
Referenced by init_slew().

```
13
14     {
15     //Take back half approach
16     //Not linear but equal to setSpeed(1-(1/2)^x)
17     for(unsigned int i = 0; i < 9; i++) {
18         if(motors_set_speeds[i] == motors_curr_speeds[i]) continue;
19         mutexTake(speeds_mutex, 10);
20         int set_speed = (motors_set_speeds[i]);
21         int curr_speed = motors_curr_speeds[i];
22         mutexGive(speeds_mutex);
23         int diff = set_speed - curr_speed;
24         int offset = diff;
25         int n = curr_speed + offset;
26         motors_curr_speeds[i] = n;
27         motorSet(i+1, n);
28     }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.41.2 Variable Documentation

7.41.2.1 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().

7.41.2.2 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().

7.41.2.3 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().

7.41.2.4 slew

```
TaskHandle slew = NULL [static]
```

Definition at line 9 of file slew.c.

Referenced by deinitSlew(), and init_slew().

7.41.2.5 speeds_mutex

```
Mutex speeds_mutex [static]
```

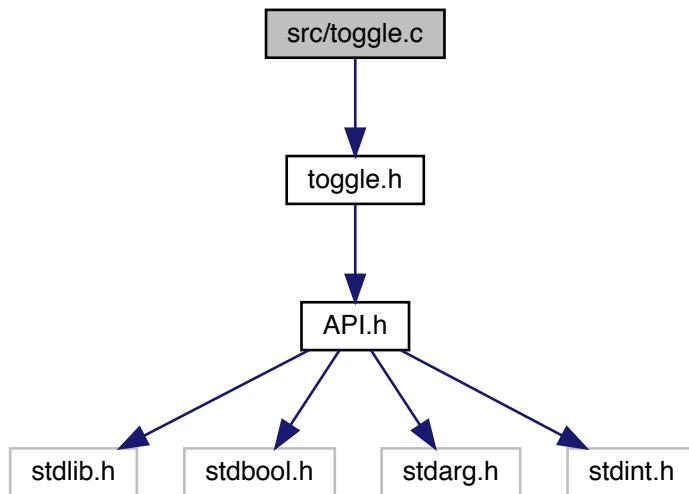
Definition at line 4 of file slew.c.

Referenced by init_slew(), set_motor_immediate(), set_motor_slew(), and updateMotors().

7.42 src/toggle.c File Reference

```
#include "toggle.h"
```

Include dependency graph for toggle.c:



Functions

- bool `buttonGetState (button_t button)`

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

- void `buttonInit ()`

Initializes the buttons array.

- bool `buttonIsNewPress (button_t button)`

Detects if button is a new press from most recent check by comparing previous value to current value.

Variables

- bool `buttonPressed` [27]

7.42.1 Function Documentation

7.42.1.1 `buttonGetState()`

```
bool buttonGetState (
    button_t    )
```

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

Parameters

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

Returns

true (pressed) or false (not pressed)

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

References `JOY_DOWN`, `JOY_LEFT`, `JOY_RIGHT`, `JOY_UP`, `joystickGetDigital()`, `LCD_BTN_CENTER`, `LCD_BTN_LEFT`, `LCD_BTN_RIGHT`, `LCD_CENT`, `LCD_LEFT`, `LCD_RIGHT`, `lcdReadButtons()`, and `uart1`.

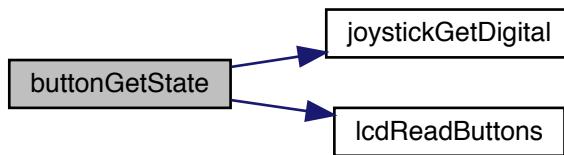
Referenced by `buttonIsNewPress()`.

```
25
26     bool currentButton = false;
27
28     // Determine how to get the current button value (from what function) and where it
29     // is, then get it.
30     if (button < LCD_LEFT) {
31         // button is a joystick button
32         unsigned char joystick;
33         unsigned char buttonGroup;
34         unsigned char buttonLocation;
35
36         button_t newButton;
37         if (button <= 11) {
38             // button is on joystick 1
39             joystick = 1;
40             newButton = button;
41         }
42         else {
43             // button is on joystick 2
44             joystick = 2;
45             // shift button down to joystick 1 buttons in order to
46             // detect which button on joystick is queried
47             newButton = (button_t)(button - 12);
48         }
49
50         switch (newButton) {
51             case 0:
52                 buttonGroup = 5;
53                 buttonLocation = JOY_DOWN;
54                 break;
55             case 1:
56                 buttonGroup = 5;
```

```

57         buttonLocation = JOY_UP;
58     break;
59     case 2:
60         buttonGroup = 6;
61         buttonLocation = JOY_DOWN;
62     break;
63     case 3:
64         buttonGroup = 6;
65         buttonLocation = JOY_UP;
66     break;
67     case 4:
68         buttonGroup = 7;
69         buttonLocation = JOY_UP;
70     break;
71     case 5:
72         buttonGroup = 7;
73         buttonLocation = JOY_LEFT;
74     break;
75     case 6:
76         buttonGroup = 7;
77         buttonLocation = JOY_RIGHT;
78     break;
79     case 7:
80         buttonGroup = 7;
81         buttonLocation = JOY_DOWN;
82     break;
83     case 8:
84         buttonGroup = 8;
85         buttonLocation = JOY_UP;
86     break;
87     case 9:
88         buttonGroup = 8;
89         buttonLocation = JOY_LEFT;
90     break;
91     case 10:
92         buttonGroup = 8;
93         buttonLocation = JOY_RIGHT;
94     break;
95     case 11:
96         buttonGroup = 8;
97         buttonLocation = JOY_DOWN;
98     break;
99     default:
100         break;
101     }
102     currentButton = joystickGetDigital(joystick, buttonGroup, buttonLocation);
103 }
104 else {
105     // button is on LCD
106     if (button == LCD_LEFT)
107         currentButton = (lcdReadButtons(uart1) ==
LCD_BTN_LEFT);
108
109     if (button == LCD_CENT)
110         currentButton = (lcdReadButtons(uart1) ==
LCD_BTN_CENTER);
111
112     if (button == LCD_RIGHT)
113         currentButton = (lcdReadButtons(uart1) ==
LCD_BTN_RIGHT);
114 }
115 return currentButton;
116 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.42.1.2 buttonInit()

```
void buttonInit ( )
```

Initializes the buttons array.

Initializes the buttons.

Definition at line 20 of file toggle.c.

References buttonPressed.

```
20      {
21      for (int i = 0; i < 27; i++)
22          buttonPressed[i] = false;
23 }
```

7.42.1.3 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

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

Returns

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

Example code:

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

Definition at line 135 of file toggle.c.

References buttonGetState(), and buttonPressed.

```
135     bool currentButton = buttonGetState(button);
136
137     if (!currentButton) // buttons is not currently pressed
138         buttonPressed[button] = false;
139
140     if (currentButton && !buttonPressed[button]) {
141         // button is currently pressed and was not detected as being pressed during last check
142         buttonPressed[button] = true;
143         return true;
144     }
145
146     else return false; // button is not pressed or was already detected
147 }
```

Here is the call graph for this function:



7.42.2 Variable Documentation

7.42.2.1 buttonPressed

```
bool buttonPressed[27];
```

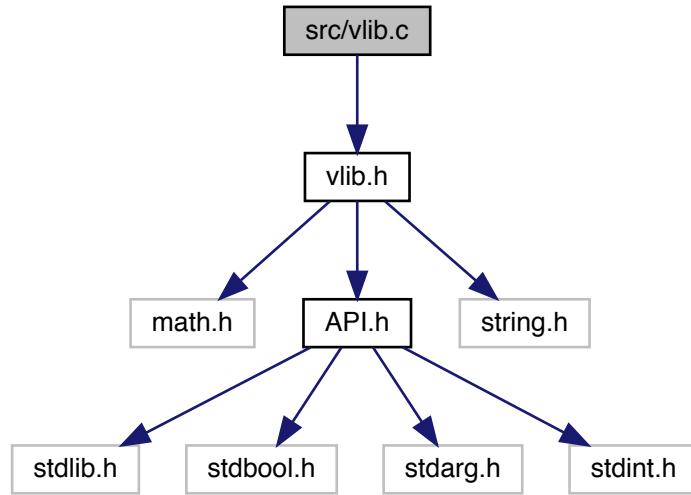
Represents the array of "wasPressed" for all 27 available buttons.

Definition at line 15 of file toggle.c.

Referenced by buttonInit(), and buttonIsNewPress().

7.43 src/vlib.c File Reference

```
#include "vlib.h"
Include dependency graph for vlib.c:
```



Functions

- void [ftoa_bad](#) (float a, char *buffer, int precision)
converts a float to string.
- int [itoa_bad](#) (int a, char *buffer, int digits)
converts a int to string.
- void [reverse](#) (char *str, int len)
reverses a string 'str' of length 'len'

7.43.1 Function Documentation

7.43.1.1 [ftoa_bad\(\)](#)

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

converts a float to string.

Parameters

<i>a</i>	the float
<i>buffer</i>	the string the float will be written to.
<i>precision</i>	digits after the decimal to write

Author

Christian DeSimone

Date

9/26/2017

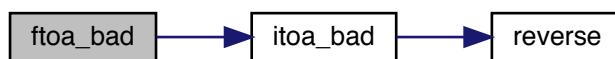
Definition at line 30 of file vlib.c.

References `itoa_bad()`.Referenced by `calculate_current_display()`.

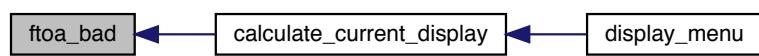
```

30
31 // Extract integer part
32 int ipart = (int)a;
33
34 // Extract floating part
35 float fpart = a - (float)ipart;
36
37 // convert integer part to string
38 int i = itoa_bad(ipart, buffer, 0);
39
40 // check for display option after point
41 if(precision != 0) {
42     buffer[i] = '.';
43     // add dot
44
45     // Get the value of fraction part up to given num.
46     // of points after dot. The third parameter is needed
47     // to handle cases like 233.007
48     fpart = fpart * pow(10, precision);
49     itoa_bad((int)fpart, buffer + i + 1, precision);
50 }
51 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.43.1.2 itoa_bad()

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

converts a int to string.

Parameters

<i>a</i>	the integer
<i>buffer</i>	the string the int will be written to.
<i>digits</i>	the number of digits to be written

Returns

the digits

Author

Chris Jerrett, Christian DeSimone

Date

9/9/2017

Definition at line 13 of file vlib.c.

References reverse().

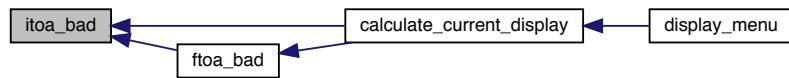
Referenced by calculate_current_display(), and ftoa_bad().

```
13     int i = 0;
14     while (a) {
15         buffer[i++] = (a%10) + '0';
16         a = a/10;
17     }
18 }
19
20 // If number of digits required is more, then
21 // add 0s at the beginning
22 while (i < digits)
23     buffer[i++] = '0';
24
25 reverse(buffer, i);
26 buffer[i] = '\0';
27 return i;
28 }
```

Here is the call graph for this function:



Here is the caller graph for this function:



7.43.1.3 reverse()

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

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

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

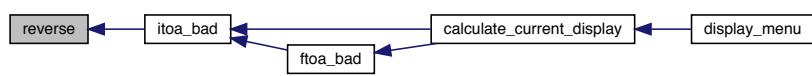
Definition at line 3 of file vlib.c.

Referenced by itoa_bad().

```

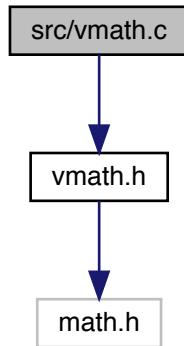
3
4     int i=0, j=len-1, temp;
5     while (i<j) {
6         temp = str[i];
7         str[i] = str[j];
8         str[j] = temp;
9         i++; j--;
10    }
11 }
```

Here is the caller graph for this function:



7.44 src/vmath.c File Reference

```
#include "vmath.h"
Include dependency graph for vmath.c:
```



Functions

- struct [polar_cord cartesian_cord_to_polar](#) (struct [cord](#) cords)
Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.
- struct [polar_cord cartesian_to_polar](#) (float x, float y)
Function to convert x and y 2 dimensional cartesian coordinated to polar coordinates.
- int [max](#) (int a, int b)
- int [min](#) (int a, int b)
- double [sind](#) (double angle)

7.44.1 Function Documentation

7.44.1.1 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

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

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)
[cord](#)

Definition at line 33 of file vmath.c.

References [cartesian_to_polar\(\)](#).

```
33
34     return cartesian_to_polar(cords.x, cords.y);
35 }
```

Here is the call graph for this function:



7.44.1.2 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

<code>x</code>	float value of the x cartesian coordinate.
<code>y</code>	float value of the y cartesian coordinate.

Returns

a struct containing the angle and magnitude.

See also

[polar_cord](#)

Definition at line 3 of file vmath.c.

References `polar_cord::angle`, and `polar_cord::magnitue`.

Referenced by `cartesian_cord_to_polar()`.

```

3
4   float degree = 0;
5   double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));
6
7   if(x < 0){
8       degree += 180.0;
9   }
10  else if(x > 0 && y < 0){
11      degree += 360.0;
12  }
13
14  if(x != 0 && y != 0){
15      degree += atan((float)y / (float)x);
16  }
17  else if(x == 0 && y > 0){
18      degree = 90.0;
19  }
20  else if(y == 0 && x < 0){
21      degree = 180.0;
22  }
23  else if(x == 0 && y < 0){
24      degree = 270.0;
25  }
26
27  struct polar_cord p;
28  p.angle = degree;
29  p.magnitue = magnitude;
30  return p;
31 }
```

Here is the caller graph for this function:



7.44.1.3 max()

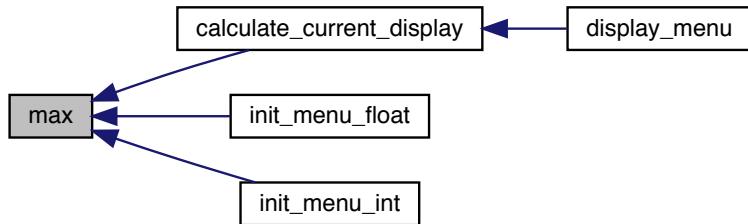
```
int max (
    int a,
    int b )
```

Definition at line 48 of file vmath.c.

Referenced by calculate_current_display(), init_menu_float(), and init_menu_int().

```
48
49     if(a > b)  return a;
50     return b;
51 }
```

Here is the caller graph for this function:



7.44.1.4 min()

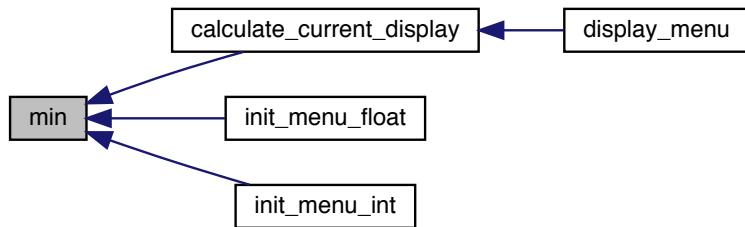
```
int min (
    int a,
    int b )
```

Definition at line 42 of file vmath.c.

Referenced by calculate_current_display(), init_menu_float(), and init_menu_int().

```
42
43     if(a < b)  return a;
44     return b;
45 }
```

Here is the caller graph for this function:



7.44.1.5 `sind()`

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

Definition at line 37 of file vmath.c.

References M_PI.

```
37      {
38      double angleradians = angle * M_PI / 180.0f;
39      return sin(angleradians);
40 }
```

7.45 testMath.py File Reference

Namespaces

- [testMath](#)

Functions

- def [testMath.test](#) (l1, l2)