Vex Team A

1.0.2

Generated by Doxygen 1.8.13

Contents

Chapter 1

InTheZoneA

Team A code for In The Zone

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 cord Struct Reference

A struct that contains cartesian coordinates.

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

Data Fields

- float x
- float y

4.1.1 Detailed Description

A struct that contains cartesian coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line 31 of file vmath.h.

4.1.2 Field Documentation

4.1.2.1 x

```
float cord::x
```

the x coordinate

Definition at line 33 of file vmath.h.

Referenced by get_joystick_cord().

4.1.2.2 y

```
float cord::y
```

the y coordinate

Definition at line 35 of file vmath.h.

Referenced by get_joystick_cord().

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

- include/vmath.h

4.2 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

4.2.1 Detailed Description

represents the state of the lcd buttons

Author

Chris Jerrett

Date

9/9/2017

Definition at line 48 of file lcd.h.

4.2.2 Field Documentation

4.2.2.1 left

`button_state lcd_buttons::left`

Definition at line 49 of file lcd.h.

Referenced by lcd_get_pressed_buttons().

4.2.2.2 middle

`button_state lcd_buttons::middle`

Definition at line 50 of file lcd.h.

Referenced by lcd_get_pressed_buttons().

4.2.2.3 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

4.3 menu_t Struct Reference

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

```
#include <menu.h>
```

Data Fields

- int current

  contains the current index of menu.
- unsigned int length

  contains the length of options char∗∗.
- int max

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

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

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

  contains the minimum float value of menu. Defaults to minimum int value
- char ∗∗ options

  contains the array of string options.
- char prompt [16]

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

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

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

  contains the type of menu.

4.3.1 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 64 of file menu.h.

4.3.2 Field Documentation

4.3.2.1 current

```
int menu_t::current
```

contains the current index of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line 138 of file menu.h.

Referenced by calculate_current_display(), and display_menu().

4.3.2.2 length

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

contains the length of options char∗∗.

Author

Chris Jerrett

Date

9/8/17

Definition at line 84 of file menu.h.

Referenced by calculate_current_display(), and init_menu_var().

4.3.2.3 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 100 of file menu.h.

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

4.3.2.4 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 124 of file menu.h.

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

4.3.2.5 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 92 of file menu.h.

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

4.3.2.6 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 116 of file menu.h.

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

4.3.2.7 options

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

contains the array of string options.

Author

Chris Jerrett

Date

9/8/17

Definition at line 77 of file menu.h.

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

4.3.2.8 prompt

```
char menu_t::prompt[16]
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 145 of file menu.h.

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

4.3.2.9 step

```
int menu_t::step
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 108 of file menu.h.

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

4.3.2.10 step_f

```
float menu_t::step_f
```

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 132 of file menu.h.

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

4.3.2.11 type

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

contains the type of menu.

Author

Chris Jerrett

Date

9/8/17

Definition at line 70 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

4.4 polar_cord Struct Reference

A struct that contains polar coordinates.

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

Data Fields

- float angle
- float magnitue

4.4.1 Detailed Description

A struct that contains polar coordinates.

Date

9/9/2017

Author

Chris Jerrett

Definition at line 19 of file vmath.h.

4.4.2 Field Documentation

4.4.2.1 angle

```
float polar_cord::angle
```

the angle of the vector

Definition at line 21 of file vmath.h.

Referenced by cartesian_to_polar().

4.4.2.2 magnitue

```
float polar_cord::magnitue
```

the magnitude of the vector

Definition at line 23 of file vmath.h.

Referenced by cartesian_to_polar().

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

- include/vmath.h

Chapter 5

File Documentation

5.1 include/battery.h File Reference

5.2 include/controller.h File Reference

controller definitions, macros

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

Macros

- #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_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)

5.2.1 Detailed Description

controller definitions, macros

Author

Chris Jerrett

Date

9/9/2017

5.2.2 Macro Definition Documentation

5.2.2.1 LEFT_JOY_X

```
#define LEFT_JOY_X 4
```

the left x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 47 of file controller.h.

Referenced by get_joystick_cord().

5.2.2.2 LEFT_JOY_Y

```
#define LEFT_JOY_Y 3
```

the left y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 54 of file controller.h.

Referenced by get_joystick_cord().

5.2.2.3 MASTER

```
#define MASTER 1
```

the master controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 19 of file controller.h.

Referenced by update_drive_motors().

5.2.2.4 PARTNER

```
#define PARTNER 2
```

the slave/partner controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 26 of file controller.h.

5.2.2.5 RIGHT_JOY_X

```
#define RIGHT_JOY_X 1
```

the right x joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 33 of file controller.h.

Referenced by get_joystick_cord().

5.2.2.6 RIGHT_JOY_Y

```
#define RIGHT_JOY_Y 2
```

the right y joystick on controller

Date

9/1/2017

Author

Chris Jerrett

Definition at line 40 of file controller.h.

Referenced by get_joystick_cord().

5.2.3 Enumeration Type Documentation

5.2.3.1 joystick

```
enum joystick
```

Represents a joystick on the controller.

Date

9/10/2017

Author

Chris Jerrett

Enumerator

RIGHT_JOY	The right joystick
LEFT_JOY	The left joystick

Definition at line 61 of file controller.h.

```
61              {
63   RIGHT_JOY,
65   LEFT_JOY,
66 };
```

5.2.4 Function Documentation

5.2.4.1 get_joystick_cord()

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

Definition at line 3 of file controller.c.

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

Referenced by update_drive_motors().

```
                                                                 {
   int x;
   int y;
   if(side == RIGHT_JOY) {
     y = joystickGetAnalog(controller, RIGHT_JOY_X);
     x = joystickGetAnalog(controller, RIGHT_JOY_Y);
   } else {
      y = joystickGetAnalog(controller, LEFT_JOY_X);
      x = joystickGetAnalog(controller, LEFT_JOY_Y);
    }
    struct cord c;
    c.x = x;
    c.y = y;
    return c;
  }
```

5.3 include/drive.h File Reference

Drive base definitions and enumerations.

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

Macros

- #define DEADSPOT 30

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 update_drive_motors ()

  Updates the drive motors during teleop.

5.3.1 Detailed Description

Drive base definitions and enumerations.

Author

Christian Desimone

Date

9/9/2017

5.3.2 Macro Definition Documentation

5.3.2.1 DEADSPOT

```
#define DEADSPOT 30
```

Definition at line 13 of file drive.h.

Referenced by deadspot().

5.3.3 Typedef Documentation

5.3.3.1 side_t

```
typedef enum side side_t
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

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

5.3.4 Enumeration Type Documentation

5.3.4.1 side

```
enum side
```

enumeration indication side of the robot.

Author

Christian Desimone

Date

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

Enumerator

LEFT	
BOTH	
RIGHT	

Definition at line 21 of file drive.h.

```
21                  {
22   LEFT,
23   BOTH,
24   RIGHT
25 } side_t;
```

5.3.5 Function Documentation

5.3.5.1 set_side_speed()

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

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

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

Definition at line 7 of file drive.c.

References BOTH, LEFT, MOTOR_BACK_LEFT, MOTOR_BACK_RIGHT, MOTOR_FRONT_RIGHT, MOTOR_↩
MIDDLE_RIGHT, and RIGHT.

```
                                            {
   if(side == RIGHT || side == BOTH){
     motorSet(MOTOR_BACK_RIGHT , speed);
      motorSet(MOTOR_FRONT_RIGHT, speed);
      motorSet(MOTOR_MIDDLE_RIGHT, speed);
    }
    if(side == LEFT || side == BOTH){
      motorSet(MOTOR_BACK_LEFT, speed);
      motorSet(MOTOR_BACK_LEFT, speed);
      motorSet(MOTOR_BACK_LEFT, speed);
    }
 }
```

5.3.5.2 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

Definition at line 28 of file drive.c.

References cartesian_cord_to_polar(), get_joystick_cord(), MASTER, and RIGHT_JOY.

```
                        {
    struct polar_cord cord = cartesian_cord_to_polar(
     get_joystick_cord(RIGHT_JOY, MASTER));
 }
```

5.4 include/encoders.h File Reference

wrapper around encoder functions

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

Macros

- #define IME_NUMBER 0

  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.

5.4.1 Detailed Description

wrapper around encoder functions

Author

Chris Jerrett

Date

9/9/2017

5.4.2 Macro Definition Documentation

5.4.2.1 IME_NUMBER

```
#define IME_NUMBER 0
```

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().

5.4.3 Function Documentation

5.4.3.1 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.

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

5.4.3.2 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.

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

5.4.3.3 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.

Referenced by initialize().

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

5.5 include/lcd.h File Reference

LCD wrapper functions and macros.

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

Data Structures

- struct lcd_buttons

  represents the state of the lcd buttons

Macros

- #define BOTTOM_ROW 2

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

  The top row on the lcd screen.

Enumerations

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

  Represents the state of a button.

Functions

- void init_main_lcd (FILE ∗lcd)

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

  Clears the lcd.
- lcd_buttons lcd_get_pressed_buttons ()

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

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

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

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

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

5.5.1 Detailed Description

LCD wrapper functions and macros.

Author

Chris Jerrett

Date

9/9/2017

5.5.2 Macro Definition Documentation

5.5.2.1 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().

5.5.2.2 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().

5.5.3 Enumeration Type Documentation

5.5.3.1 button_state

```
enum button_state
```

Represents the state of a button.

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

Author

Chris Jerrett

Date

9/9/2017

Enumerator

RELEASED	A released button
PRESSED	A pressed button

Definition at line 36 of file lcd.h.

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

5.5.4 Function Documentation

5.5.4.1 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.

Referenced by initialize().

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

5.5.4.2 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(), and lcd_port.

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

5.5.4.3 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, lcd_buttons::left, lcd_buttons::middle, PRESSED, RELEASED, and lcd_↩ buttons::right.

Referenced by display_menu(), and promt_confirmation().

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

  return btns;
}
```

5.5.4.4 lcd_print()

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

prints a string to a line on the lcd

Parameters

line	the line to print on
str	string to print

Author

Chris Jerrett

Date

9/9/2017

Definition at line 44 of file lcd.c.

References lcd_assert(), and lcd_port.

Referenced by display_menu(), and promt_confirmation().

```
                                                      {
  lcd_assert();
  lcdSetText(lcd_port, line, str);
}
```

5.5.4.5 lcd_printf()

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

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

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

Definition at line 49 of file lcd.c.

References lcd_assert(), and lcd_port.

```
                                                        {
  lcd_assert();
  lcdPrint(lcd_port, line, format_str);
}
```

5.5.4.6 lcd_set_backlight()

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

sets the backlight of the lcd

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

Definition at line 54 of file lcd.c.

References lcd_assert(), and lcd_port.

```
                                {
  lcd_assert();
  lcdSetBacklight(lcd_port, state);
}
```

5.5.4.7 promt_confirmation()

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

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

Parameters

confirm_text	the text for the user to confirm.

Author

Chris Jerrett

Date

9/9/2017

Definition at line 59 of file lcd.c.

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

Referenced by initialize().

```
                                                   {
  lcd_assert();
  lcd_print(1, confirm_text);
  while(lcd_get_pressed_buttons().middle != PRESSED){
    delay(200);
  }
}
```

5.6 include/log.h File Reference

Contains logging functions.

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

Macros

- #define DEBUG 4

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

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

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

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

    logs errors and warnings. Also displays error to lcd

Functions

- void debug (const char ∗debug_message)

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

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

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

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

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

5.6.1 Detailed Description

Contains logging functions.

Author

Chris Jerrett

Date

9/16/2017

5.6.2 Macro Definition Documentation

5.6.2.1 DEBUG

```
#define DEBUG 4
```

logging only info debug. most verbose level

Author

Chris Jerrett

Date

9/10/17

Definition at line 51 of file log.h.

5.6.2.2 ERROR

```
#define ERROR 1
```

logging only errors. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line 28 of file log.h.

Referenced by debug(), and info().

5.6.2.3 INFO

```
#define INFO 3
```

logging only info messages and higher.

Author

Chris Jerrett

Date

9/10/17

Definition at line 43 of file log.h.

5.6.2.4 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 20 of file log.h.

Referenced by error().

5.6.2.5 WARNING

```
#define WARNING 2
```

logs errors and warnings. Also displays error to lcd

Author

Chris Jerrett

Date

9/10/17

Definition at line 36 of file log.h.

Referenced by warning().

5.6.3 Function Documentation

5.6.3.1 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

debug_message	the message

Definition at line 37 of file log.c.

References ERROR, and log_level.

Referenced by updateMotors().

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

5.6.3.2 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

error_message	the message

Definition at line 21 of file log.c.

References log_info(), log_level, and NONE.

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

5.6.3.3 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

info_message	the message

Definition at line 31 of file log.c.

References ERROR, and log_level.

Referenced by init_slew().

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

5.6.3.4 init_error()

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

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

use_lcd	whether to use the lcd
lcd	the lcd

Definition at line 6 of file log.c.

References log_lcd.

Referenced by initialize().

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

5.6.3.5 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

warning_message	the message

Definition at line 26 of file log.c.

References log_info(), log_level, and WARNING.

Referenced by initialize().

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

5.7 include/main.h File Reference

Header file for global functions.

`#include <API.h>`

Functions

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

5.7.1 Detailed Description

Header file for global functions.

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

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

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

5.7.2 Function Documentation

5.7.2.1 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 29 of file auto.c.

```
                  {
}
```

5.7.2.2 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 45 of file init.c.

References init_encoders(), init_error(), init_main_lcd(), init_slew(), promt_confirmation(), and warning().

```
                  {
  setTeamName("9228A");
  init_slew();
  init_main_lcd(uart1);
  init_error(true, uart2);
  if(!init_encoders()) {
    promt_confirmation("Check IME");
    warning("CHECK IME");
  }

  if(powerLevelBackup()/1000 == 0) {
    promt_confirmation("Check Backup");
    warning("Checkbackup bat");
  }
}
```

5.7.2.3 initializeIO()

```
void initializeIO ( )
```

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

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

Definition at line 28 of file init.c.

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

5.7.2.4 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 33 of file opcontrol.c.

References set_motor_slew().

```
                     {
    while (1) {
        set_motor_slew(2, 100);
        delay(20);
    }
}
```

5.8 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>
```

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, 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, 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, char ∗prompt, char ∗options,...)

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

5.8.1 Detailed Description

Contains menu functionality and abstraction.

Author

Chris Jerrett

Date

9/9/2017

5.8.2 Typedef Documentation

5.8.2.1 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

5.8.3 Enumeration Type Documentation

5.8.3.1 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 28 of file menu.h.

```
                {
    INT_TYPE,
    FLOAT_TYPE,
    STRING_TYPE
 };
```

5.8.4 Function Documentation

5.8.4.1 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

5.8.4.2 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

5.8.4.3 denint_menu()

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

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

Parameters

menu	the menu to free

See also

menu

Author

Chris Jerrett

Date

9/8/17

Definition at line 92 of file menu.c.

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

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

5.8.4.4 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

menu	the menu to display

See also

menu_type

Author

Chris Jerrett

Date

9/8/17

Definition at line 74 of file menu.c.

References calculate_current_display(), menu_t::current, lcd_get_pressed_buttons(), lcd_print(), PRESSED, menu_t::prompt, RELEASED, and TOP_ROW.

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

    if(lcd_get_pressed_buttons().right == PRESSED) {
      menu->current += 1;
    }
    if(lcd_get_pressed_buttons().left == PRESSED) {
      menu->current -= 1;
    }
    delay(500);
  }
  return menu->current;
}
```

5.8.4.5 init_menu_float()

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

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

Parameters

type	the type of menu

See also

menu_type

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 39 of file menu.c.

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

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

5.8.4.6 init_menu_int()

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

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

Parameters

type	the type of menu

See also

menu_type

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 31 of file menu.c.

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

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

5.8.4.7 init_menu_var()

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

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

Parameters

type	the type of menu

See also

menu_type

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 17 of file menu.c.

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

```
                                                                                                {
  menu_t* menu = create_menu(type, prompt);
  va_list values;
  char **options_array = (char**)malloc(sizeof(char*) * nums);
  va_start(values, options);
  for(unsigned int i = 0; i < nums; i++){
    options_array[i] = va_arg(values, char*);
  }
  va_end(values);
  menu->options = options_array;
  menu->length = nums;
  return menu;
}
```

5.9 include/ports.h File Reference

contains port macros for sensors

Macros

- #define IME_FRONT_RIGHT 0

  Number of integrated motor encoders Used when checking to see if all imes are plugged in.
- #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 1

  Front left drive motor of robot base.
- #define MOTOR_FRONT_RIGHT 0

  Front right drive motor of robot base.
- #define MOTOR_MIDDLE_LEFT 3

  Middle left drive motor of robot base.
- #define MOTOR_MIDDLE_RIGHT 2

  Middle right drive motor of robot base.

5.9.1 Detailed Description

contains port macros for sensors

Author

Chris Jerrett

Date

9/9/2017

5.9.2 Macro Definition Documentation

5.9.2.1 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 ports.h.

5.9.2.2 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 59 of file ports.h.

Referenced by set_side_speed().

5.9.2.3 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 53 of file ports.h.

Referenced by set_side_speed().

5.9.2.4 MOTOR_FRONT_LEFT

```
#define MOTOR_FRONT_LEFT 1
```

Front left drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 32 of file ports.h.

5.9.2.5 MOTOR_FRONT_RIGHT

```
#define MOTOR_FRONT_RIGHT 0
```

Front right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 25 of file ports.h.

Referenced by set_side_speed().

5.9.2.6 MOTOR_MIDDLE_LEFT

```
#define MOTOR_MIDDLE_LEFT 3
```

Middle left drive motor of robot base.

Date

9/7/2017

Author

Christian Desimone

Definition at line 46 of file ports.h.

5.9.2.7 MOTOR_MIDDLE_RIGHT

```
#define MOTOR_MIDDLE_RIGHT 2
```

Middle right drive motor of robot base.

Author

Christian Desimone

Date

9/7/2017

Definition at line 39 of file ports.h.

Referenced by set_side_speed().

5.10 include/slew.h File Reference

Contains the slew rate controller wrapper for the motors.

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

Macros

- #define MOTOR_PORTS 12

  The number of motor ports on the robot.
- #define RAMP_PROPORTION 2

  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_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.

5.10.1 Detailed Description

Contains the slew rate controller wrapper for the motors.

Author

Chris Jerrett

Date

9/14/17

5.10.2 Macro Definition Documentation

5.10.2.1 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.

Referenced by init_slew(), and updateMotors().

5.10.2.2 RAMP_PROPORTION

```
#define RAMP_PROPORTION 2
```

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.

Referenced by updateMotors().

5.10.2.3 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.

Referenced by init_slew().

5.10.3 Function Documentation

5.10.3.1 deinitslew()

```
void deinitslew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line 62 of file slew.c.

References motors_set_speeds, and slew.

```
                  {
  free(motors_set_speeds);
  taskDelete(slew);
}
```

5.10.3.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

Definition at line 49 of file slew.c.

References calloc_real(), info(), initialized, MOTOR_PORTS, mutex, slew, UPDATE_PERIOD_MS, and update↩
Motors().

Referenced by initialize().

```
                 {
  info("Init Slew");
  calloc_real(MOTOR_PORTS, sizeof(char));
  mutex = mutexCreate();
  slew = taskRunLoop(updateMotors, UPDATE_PERIOD_MS);
  initialized = true;
}
```

5.10.3.3 set_motor_slew()

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

Sets motor speed wrapped inside the slew rate controller.

Parameters

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

Author

Chris Jerrett

Date

9/14/17

Definition at line 67 of file slew.c.

References motors_set_speeds, and mutex.

Referenced by operatorControl().

```
                                          {
  if(mutexTake(mutex, 100)) {
      motors_set_speeds[motor] = speed;
      mutexGive(mutex);
  }
}
```

5.10.3.4 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 31 of file slew.c.

References debug(), MOTOR_PORTS, motors_set_speeds, mutex, and RAMP_PROPORTION.

Referenced by init_slew().

```
                    {
  //Take back half approach
  //Not linear but equal to setSpeed(1-(1/2)^x)
  if(mutexTake(mutex, 10)) {
    for(int i = 0; i < MOTOR_PORTS; i++) {
      char set_speed = motors_set_speeds[i];
      char curr_speed = motorGet(i);
      char diff = set_speed - curr_speed;
      int n = (int) curr_speed + ceil(diff/(float)RAMP_PROPORTION);
      char c[16];
      sprintf(c, "Set Motor %d: %d", i, n);
      debug(c);
      motorSet(i, n);
    }
    mutexGive(mutex);
  }
}
```

5.11 include/vlib.h File Reference

Contains misc helpful functions.

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

Functions

- void * calloc_real (size_t elements, size_t size)
- void ftoa (float a, char *buffer, int precision)

  converts a float to string.
- int itoa (int a, char *buffer, int digits)

  converts a int to string.
- void reverse (char *str, int len)

  reverses a string 'str' of length 'len'

5.11.1 Detailed Description

Contains misc helpful functions.

Author

Chris Jerrett

Date

9/9/2017

5.11.2 Function Documentation

5.11.2.1 calloc_real()

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

Definition at line 53 of file vlib.c.

Referenced by init_slew().

```
                                                 {
  void *mem = malloc(elements * size);
  //This is not a error. Bad ATOM!
  memset(mem, 0, elements * size);
  return mem;
}
```

5.11.2.2 ftoa()

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

converts a float to string.

Parameters

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

Definition at line 30 of file vlib.c.

References itoa().

Referenced by calculate_current_display().

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

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

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

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

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

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

5.11.2.3 itoa()

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

converts a int to string.

Parameters

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

Returns

the digits

Author

Chris Jerrett

Date

9/9/2017

Definition at line 13 of file vlib.c.

References reverse().

Referenced by calculate_current_display(), and ftoa().

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

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

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

5.11.2.4 reverse()

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

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

str	the string to reverse
len	the length

Definition at line 3 of file vlib.c.

Referenced by itoa().

```
                                  {
      int i=0, j=len-1, temp;
      while (i<j) {
          temp = str[i];
          str[i] = str[j];
          str[j] = temp;
          i++; j--;
      }
  }
```

5.12 include/vmath.h File Reference

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

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

Data Structures

- struct cord
  A struct that contains cartesian coordinates.
- struct polar_cord
  A struct that contains polar coordinates.

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.

5.12.1 Detailed Description

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

Author

Christian Desimone
Chris Jerrett

Date

9/9/2017

5.12.2 Function Documentation

5.12.2.1 cartesian_cord_to_polar()

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

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

Author

Christian Desimone

Date

9/8/2017

Parameters

cords	the cartesian cords

Returns

a struct containing the angle and magnitude.

See also

polar_cord
cord

Definition at line 33 of file vmath.c.

References cartesian_to_polar().

Referenced by update_drive_motors().

```
                                                    {
    return cartesian_to_polar(cords.x, cords.y);
}
```

5.12.2.2 cartesian_to_polar()

```
struct polar_cord cartesian_to_polar (
             float x,
             float y )
```

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

Author

Christian Desimone

Date

9/8/2017

Parameters

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

Returns

a struct containing the angle and magnitude.

See also

polar_cord

Definition at line 3 of file vmath.c.

References polar_cord::angle, and polar_cord::magnitue.

Referenced by cartesian_cord_to_polar().

```
                                                        {
  float degree = 0;
  double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));

  if(x < 0){
    degree += 180.0;
  }
   else if(x > 0 && y < 0){
     degree += 360.0;
   }

   if(x != 0 && y != 0){
     degree += atan((float)y / (float)x);
   }
   else if(x == 0 && y > 0){
     degree = 90.0;
   }
   else if(y == 0 && x < 0){
     degree = 180.0;
   }
   else if(x == 0 && y < 0){
     degree = 270.0;
   }

   struct polar_cord p;
   p.angle = degree;
   p.magnitue = magnitude;
   return p;
 }
```

5.13 README.md File Reference

5.14 src/auto.c File Reference

File for autonomous code.

`#include "main.h"`

Functions

- void autonomous ()

5.14.1 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.

5.14.2 Function Documentation

5.14.2.1 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 29 of file auto.c.

```
29                  {
30 }
```

5.15 src/battery.c File Reference

```
#include "battery.h"
```

5.16 src/controller.c File Reference

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

Functions

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

5.16.1 Function Documentation

5.16.1.1 get_joystick_cord()

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

Definition at line 3 of file controller.c.

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

Referenced by update_drive_motors().

```
                                                                  {
  int x;
  int y;
  if(side == RIGHT_JOY) {
    y = joystickGetAnalog(controller, RIGHT_JOY_X);
    x = joystickGetAnalog(controller, RIGHT_JOY_Y);
  } else {
     y = joystickGetAnalog(controller, LEFT_JOY_X);
     x = joystickGetAnalog(controller, LEFT_JOY_Y);
   }
   struct cord c;
   c.x = x;
   c.y = y;
   return c;
 }
```

5.17 src/drive.c File Reference

```
#include "drive.h"
#include "ports.h"
#include "vmath.h"
#include "controller.h"
#include <API.h>
```

Functions

- static int deadspot (int val)
- static int joystick_interpolate (int val)
- void set_side_speed (side_t side, int speed)

    sets the speed of one side of the robot.
- void update_drive_motors ()

    Updates the drive motors during teleop.

5.17.1 Function Documentation

5.17.1.1 deadspot()

```
static int deadspot (
            int val ) [static]
```

Definition at line 24 of file drive.c.

References DEADSPOT.

```
{
  return abs(val) > DEADSPOT ? val : 0;
}
```

5.17.1.2 joystick_interpolate()

```
static int joystick_interpolate (
            int val ) [static]
```

Definition at line 20 of file drive.c.

```
{

}
```

5.17.1.3 set_side_speed()

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

sets the speed of one side of the robot.

Author

Christian Desimone

Parameters

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

Definition at line 7 of file drive.c.

References BOTH, LEFT, MOTOR_BACK_LEFT, MOTOR_BACK_RIGHT, MOTOR_FRONT_RIGHT, MOTOR_↩ MIDDLE_RIGHT, and RIGHT.

```
                                            {
   if(side == RIGHT || side == BOTH){
     motorSet(MOTOR_BACK_RIGHT , speed);
      motorSet(MOTOR_FRONT_RIGHT, speed);
      motorSet(MOTOR_MIDDLE_RIGHT, speed);
    }
    if(side == LEFT || side == BOTH){
      motorSet(MOTOR_BACK_LEFT, speed);
      motorSet(MOTOR_BACK_LEFT, speed);
      motorSet(MOTOR_BACK_LEFT, speed);
    }
 }
```

5.17.1.4 update_drive_motors()

```
void update_drive_motors ( )
```

Updates the drive motors during teleop.

Author

Christian Desimone

Date

9/5/17

Definition at line 28 of file drive.c.

References cartesian_cord_to_polar(), get_joystick_cord(), MASTER, and RIGHT_JOY.

```
                             {
    struct polar_cord cord = cartesian_cord_to_polar(
     get_joystick_cord(RIGHT_JOY, MASTER));
 }
```

5.18 src/encoders.c File Reference

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

Functions

- int get_encoder_ticks (unsigned char address)

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

  Gets the encoder reads.
- bool init_encoders ()

  Initializes all motor encoders.

5.18.1 Function Documentation

5.18.1.1 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.

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

5.18.1.2 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.

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

5.18.1.3 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.

Referenced by initialize().

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

5.19 src/init.c File Reference

File for initialization code.

```
#include "main.h"
#include "slew.h"
#include "lcd.h"
#include "log.h"
#include "encoders.h"
```

Functions

- void initialize ()
- void initializeIO ()

5.19.1 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.

5.19.2 Function Documentation

5.19.2.1 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 45 of file init.c.

References init_encoders(), init_error(), init_main_lcd(), init_slew(), promt_confirmation(), and warning().

```
                  {
  setTeamName("9228A");
  init_slew();
  init_main_lcd(uart1);
  init_error(true, uart2);
  if(!init_encoders()) {
    promt_confirmation("Check IME");
    warning("CHECK IME");
  }

  if(powerLevelBackup()/1000 == 0) {
    promt_confirmation("Check Backup");
    warning("Checkbackup bat");
  }
}
```

5.19.2.2 initializeIO()

```
void initializeIO ( )
```

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

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

Definition at line 28 of file init.c.

```
28                    {
29     watchdogInit();
30 }
```

5.20 src/lcd.c File Reference

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

Functions

- void init_main_lcd (FILE ∗lcd)

  Initializes the lcd screen. Also will initialize the lcd_port var. Must be called before any lcd function can be called.
- static 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 promt_confirmation (const char ∗confirm_text)

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

Variables

- static FILE ∗ lcd_port = NULL

5.20.1 Function Documentation

5.20.1.1 init_main_lcd()

```
void init_main_lcd (
            FILE ∗ lcd )
```

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

Parameters

lcd	the urart port of the lcd screen

See also

uart1
uart2

Author

Chris Jerrett

Date

9/9/2017

Definition at line 39 of file lcd.c.

References lcd_port.

Referenced by initialize().

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

5.20.1.2 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.

Referenced by lcd_clear(), lcd_get_pressed_buttons(), lcd_print(), lcd_printf(), lcd_set_backlight(), and promt_↩
confirmation().

```
                            {
  if(lcd_port != NULL) {
    printf("LCD NULL!");
    exit(1);
  }
}
```

5.20.1.3 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(), and lcd_port.

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

5.20.1.4 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, lcd_buttons::left, lcd_buttons::middle, PRESSED, RELEASED, and lcd_↩ buttons::right.

Referenced by display_menu(), and promt_confirmation().

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

  return btns;
}
```

5.20.1.5 lcd_print()

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

prints a string to a line on the lcd

Parameters

line	the line to print on
str	string to print

Author

Chris Jerrett

Date

9/9/2017

Definition at line 44 of file lcd.c.

References lcd_assert(), and lcd_port.

Referenced by display_menu(), and promt_confirmation().

```
                                               {
  lcd_assert();
  lcdSetText(lcd_port, line, str);
}
```

5.20.1.6 lcd_printf()

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

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

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

Definition at line 49 of file lcd.c.

References lcd_assert(), and lcd_port.

```
                                                                  {
  lcd_assert();
  lcdPrint(lcd_port, line, format_str);
}
```

5.20.1.7 lcd_set_backlight()

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

sets the backlight of the lcd

Parameters

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

Author

Chris Jerrett

Date

9/9/2017

Definition at line 54 of file lcd.c.

References lcd_assert(), and lcd_port.

```
                                  {
  lcd_assert();
  lcdSetBacklight(lcd_port, state);
}
```

5.20.1.8 promt_confirmation()

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

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

Parameters

confirm_text	the text for the user to confirm.

Author

Chris Jerrett

Date

9/9/2017

Definition at line 59 of file lcd.c.

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

Referenced by initialize().

```
                                                    {
  lcd_assert();
  lcd_print(1, confirm_text);
  while(lcd_get_pressed_buttons().middle != PRESSED){
    delay(200);
  }
}
```

5.20.2 Variable Documentation

5.20.2.1 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().

5.21 src/log.c File Reference

`#include "log.h"`

Functions

- void debug (const char ∗debug_message)

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

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

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

  Initializes the error lcd system Only required if using lcd.
- static void log_info (const char ∗s, const char ∗mess)
- void warning (const char ∗warning_message)

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

Variables

- static FILE ∗ log_lcd = NULL
- unsigned int log_level = DEBUG

5.21.1 Function Documentation

5.21.1.1 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

debug_message	the message

Definition at line 37 of file log.c.

References ERROR, and log_level.

Referenced by updateMotors().

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

5.21.1.2 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

error_message	the message

Definition at line 21 of file log.c.

References log_info(), log_level, and NONE.

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

5.21.1.3 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

info_message	the message

Definition at line 31 of file log.c.

References ERROR, and log_level.

Referenced by init_slew().

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

5.21.1.4 init_error()

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

Initializes the error lcd system Only required if using lcd.

Author

Chris Jerrett

Date

9/10/17

Parameters

use_lcd	whether to use the lcd
lcd	the lcd

Definition at line 6 of file log.c.

References log_lcd.

Referenced by initialize().

```
                                          {
  if(use_lcd) {
    lcdInit(lcd);
    log_lcd = lcd;
  }
}
```

5.21.1.5 log_info()

```
static void log_info (
            const char * s,
            const char * mess )  [static]
```

Definition at line 13 of file log.c.

References BOTTOM_ROW, log_lcd, and TOP_ROW.

Referenced by error(), and warning().

```
                                                   {
  printf("[%s]: %s\n", s, mess);
  lcdSetBacklight(log_lcd, true);
  lcdClear(log_lcd);
  lcdPrint(log_lcd, TOP_ROW, s);
  lcdPrint(log_lcd, BOTTOM_ROW, mess);
}
```

5.21.1.6 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

warning_message	the message

Definition at line 26 of file log.c.

References log_info(), log_level, and WARNING.

Referenced by initialize().

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

5.21.2 Variable Documentation

5.21.2.1 log_lcd

```
FILE* log_lcd = NULL [static]
```

Definition at line 4 of file log.c.

Referenced by init_error(), and log_info().

5.21.2.2 log_level

```
unsigned int log_level = DEBUG
```

Definition at line 3 of file log.c.

Referenced by debug(), error(), info(), and warning().

5.22 src/menu.c File Reference

```
#include "menu.h"
```

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, 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, 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, char ∗prompt, char ∗options,...)

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

5.22.1 Function Documentation

5.22.1.1 calculate_current_display()

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

Definition at line 47 of file menu.c.

References menu_t::current, FLOAT_TYPE, ftoa(), INT_TYPE, itoa(), menu_t::length, menu_t::max, menu_t::max_f, menu_t::min, menu_t::min_f, menu_t::options, menu_t::step, menu_t::step_f, STRING_TYPE, and menu_t::type.

Referenced by display_menu().

```
                                                                 {
  if(menu->type == STRING_TYPE){
    //Ignore warning
    rtn = (menu->options[menu->current % (menu->length)]);
  }
  if(menu->type == INT_TYPE) {
    int step = (menu->step);
    int min = (menu->min);
    int max = (menu->max);
    int value = menu->current * step;
    value = value < min ? min : value;
    value = value > max ? max : value;
    itoa(value, rtn, 4);
  }
  if(menu->type == FLOAT_TYPE) {
    float step = (menu->step_f);
    float min = (menu->min_f);
    float max = (menu->max_f);
    float value = menu->current * step;
    value = value < min ? min : value;
    value = value > max ? max : value;

    ftoa(value, rtn, 5);
  }
}
```

5.22.1.2 create_menu()

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

Definition at line 3 of file menu.c.

References menu_t::max, menu_t::max_f, menu_t::min, menu_t::min_f, menu_t::prompt, menu_t::step, menu_t::step_f, and menu_t::type.

Referenced by init_menu_float(), init_menu_int(), and init_menu_var().

```
                                                                     {
  menu_t* menu = (menu_t*) malloc(sizeof(menu_t));
  menu->type = type;
  strcpy(menu->prompt, prompt);
  menu->max = INT_MAX;
  menu->min = INT_MIN;
  menu->step = 1;
  menu->min_f = FLT_MIN;
  menu->max_f = FLT_MAX;
  menu->step_f = 1;

  return menu;
}
```

5.22.1.3 denint_menu()

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

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

Parameters

menu	the menu to free

See also

menu

Author

Chris Jerrett

Date

9/8/17

Definition at line 92 of file menu.c.

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

```
92                              {
93   free(menu->prompt);
94   if(menu->options != NULL) free(menu->options);
95   free(menu);
96 }
```

5.22.1.4 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

menu	the menu to display

See also

menu_type

Author

Chris Jerrett

Date

9/8/17

Definition at line 74 of file menu.c.

References calculate_current_display(), menu_t::current, lcd_get_pressed_buttons(), lcd_print(), PRESSED, menu_t::prompt, RELEASED, and TOP_ROW.

```
74                             {
75   lcd_print(TOP_ROW, menu->prompt);
76   //Will exit if teleop or autonomous begin. This is extremely important if robot disconnects or resets.
77   while(lcd_get_pressed_buttons().middle == RELEASED && !isEnabled()) {
78     char val[16];
79     calculate_current_display(val, menu);
80
81     if(lcd_get_pressed_buttons().right == PRESSED) {
82       menu->current += 1;
83     }
84     if(lcd_get_pressed_buttons().left == PRESSED) {
85       menu->current -= 1;
86     }
87     delay(500);
88   }
89   return menu->current;
90 }
```

5.22.1.5 init_menu_float()

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

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

Parameters

type	the type of menu

See also

menu_type

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 39 of file menu.c.

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

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

5.22.1.6 init_menu_int()

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

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

Parameters

type	the type of menu

See also

menu_type

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 31 of file menu.c.

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

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

5.22.1.7 init_menu_var()

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

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

Parameters

type	the type of menu

See also

menu_type

Parameters

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

Author

Chris Jerrett

Date

9/8/17

Definition at line 17 of file menu.c.

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

```
                                                                                                  {
  menu_t* menu = create_menu(type, prompt);
  va_list values;
  char **options_array = (char**)malloc(sizeof(char*) * nums);
  va_start(values, options);
  for(unsigned int i = 0; i < nums; i++){
    options_array[i] = va_arg(values, char*);
  }
  va_end(values);
  menu->options = options_array;
  menu->length = nums;
  return menu;
}
```

5.23 src/opcontrol.c File Reference

File for operator control code.

```
#include "main.h"
#include "slew.h"
```

Functions

- void operatorControl ()

5.23.1 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.

5.23.2 Function Documentation

5.23.2.1 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 33 of file opcontrol.c.

References set_motor_slew().

```
33                      {
34     while (1) {
35         set_motor_slew(2, 100);
36         delay(20);
37     }
38 }
```

5.24 src/slew.c File Reference

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

Functions

- void deinitslew ()

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

  Initializes the slew rate controller.
- void set_motor_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

  Boolean indicating whether or not the slew rate controller has been initialized.
- static signed char ∗ motors_set_speeds = NULL

  Array of motor speed values to set the motors to.
- static Mutex mutex

  mutex to protect the data in the array of speeds from being read or written to simultaneously.
- static TaskHandle slew = NULL

  Task that will handle updating the motors on a routine period.

5.24.1 Function Documentation

5.24.1.1 deinitslew()

```
void deinitslew ( )
```

Deinitializes the slew rate controller and frees memory.

Author

Chris Jerrett

Date

9/14/17

Definition at line 62 of file slew.c.

References motors_set_speeds, and slew.

```
                 {
  free(motors_set_speeds);
  taskDelete(slew);
}
```

5.24.1.2 init_slew()

```
void init_slew ( )
```

Initializes the slew rate controller.

Author

Chris Jerrett, Christian DeSimone

Date

9/14/17

Definition at line 49 of file slew.c.

References calloc_real(), info(), initialized, MOTOR_PORTS, mutex, slew, UPDATE_PERIOD_MS, and update↩ Motors().

Referenced by initialize().

```
                {
  info("Init Slew");
  calloc_real(MOTOR_PORTS, sizeof(char));
  mutex = mutexCreate();
  slew = taskRunLoop(updateMotors, UPDATE_PERIOD_MS);
  initialized = true;
}
```

5.24.1.3 set_motor_slew()

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

Sets motor speed wrapped inside the slew rate controller.

Parameters

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

Author

Chris Jerrett

Date

9/14/17

Definition at line 67 of file slew.c.

References motors_set_speeds, and mutex.

Referenced by operatorControl().

```
                                        {
  if(mutexTake(mutex, 100)) {
      motors_set_speeds[motor] = speed;
      mutexGive(mutex);
  }
}
```

5.24.1.4 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 31 of file slew.c.

References debug(), MOTOR_PORTS, motors_set_speeds, mutex, and RAMP_PROPORTION.

Referenced by init_slew().

```
                    {
  //Take back half approach
  //Not linear but equal to setSpeed(1-(1/2)^x)
  if(mutexTake(mutex, 10)) {
    for(int i = 0; i < MOTOR_PORTS; i++) {
      char set_speed = motors_set_speeds[i];
      char curr_speed = motorGet(i);
      char diff = set_speed - curr_speed;
      int n = (int) curr_speed + ceil(diff/(float)RAMP_PROPORTION);
      char c[16];
      sprintf(c, "Set Motor %d: %d", i, n);
      debug(c);
      motorSet(i, n);
    }
    mutexGive(mutex);
  }
}
```

5.24.2 Variable Documentation

5.24.2.1 initialized

```
bool initialized = false  [static]
```

Boolean indicating whether or not the slew rate controller has been initialized.

Author

Chris Jerrett

Date

9/14/17

Definition at line 29 of file slew.c.

Referenced by init_slew().

5.24.2.2 motors_set_speeds

```
signed char* motors_set_speeds = NULL  [static]
```

Array of motor speed values to set the motors to.

Author

Chris Jerrett

Date

9/14/17

Definition at line 15 of file slew.c.

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

5.24.2.3 mutex

```
Mutex mutex  [static]
```

mutex to protect the data in the array of speeds from being read or written to simultaneously.

Author

Chris Jerrett

Date

9/14/17

Definition at line 8 of file slew.c.

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

5.24.2.4 slew

```
TaskHandle slew = NULL  [static]
```

Task that will handle updating the motors on a routine period.

Author

Chris Jerrett

Date

9/14/17

Definition at line 22 of file slew.c.

Referenced by deinitslew(), and init_slew().

5.25 src/vlib.c File Reference

```
#include "vlib.h"
```

Functions

- void ∗ calloc_real (size_t elements, size_t size)
- void ftoa (float a, char ∗buffer, int precision)

  converts a float to string.
- int itoa (int a, char ∗buffer, int digits)

  converts a int to string.
- void reverse (char ∗str, int len)

  reverses a string 'str' of length 'len'

5.25.1 Function Documentation

5.25.1.1 calloc_real()

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

Definition at line 53 of file vlib.c.

Referenced by init_slew().

```
                                                 {
  void *mem = malloc(elements * size);
  //This is not a error. Bad ATOM!
  memset(mem, 0, elements * size);
  return mem;
}
```

5.25.1.2 ftoa()

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

converts a float to string.

Parameters

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

Definition at line 30 of file vlib.c.

References itoa().

Referenced by calculate_current_display().

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

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

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

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

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

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

5.25.1.3 itoa()

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

converts a int to string.

Parameters

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

Returns

the digits

Author

Chris Jerrett

Date

9/9/2017

Definition at line 13 of file vlib.c.

References reverse().

Referenced by calculate_current_display(), and ftoa().

```
13                                          {
14   int i = 0;
15    while (a) {
16        buffer[i++] = (a%10) + '0';
17        a = a/10;
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 }
```

5.25.1.4 reverse()

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

reverses a string 'str' of length 'len'

Author

Chris Jerrett

Date

9/9/2017

Parameters

str	the string to reverse
len	the length

Definition at line 3 of file vlib.c.

Referenced by itoa().

```
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 }
```

5.26 src/vmath.c File Reference

```
#include "vmath.h"
```

Functions

- struct polar_cord cartesian_cord_to_polar (struct cord cords)

  Function to convert x and y 2 dimensional cartesian cordinated to polar coordinates.
- struct polar_cord cartesian_to_polar (float x, float y)

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

5.26.1 Function Documentation

5.26.1.1 cartesian_cord_to_polar()

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

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

Author

Christian Desimone

Date

9/8/2017

Parameters

cords	the cartesian cords

Returns

a struct containing the angle and magnitude.

See also

polar_cord
cord

Definition at line 33 of file vmath.c.

References cartesian_to_polar().

Referenced by update_drive_motors().

```
                                                                    {
    return cartesian_to_polar(cords.x, cords.y);
}
```

5.26.1.2 cartesian_to_polar()

```
struct polar_cord cartesian_to_polar (
             float x,
             float y )
```

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

Author

Christian Desimone

Date

9/8/2017

Parameters

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

Returns

a struct containing the angle and magnitude.

See also

polar_cord

Definition at line 3 of file vmath.c.

References polar_cord::angle, and polar_cord::magnitue.

Referenced by cartesian_cord_to_polar().

```
                                                       {
  float degree = 0;
  double magnitude = sqrt((fabs(x) * fabs(x)) + (fabs(y) * fabs(y)));

  if(x < 0){
    degree += 180.0;
  }
   else if(x > 0 && y < 0){
     degree += 360.0;
   }

   if(x != 0 && y != 0){
     degree += atan((float)y / (float)x);
   }
   else if(x == 0 && y > 0){
     degree = 90.0;
   }
   else if(y == 0 && x < 0){
     degree = 180.0;
   }
   else if(x == 0 && y < 0){
     degree = 270.0;
   }

   struct polar_cord p;
   p.angle = degree;
   p.magnitue = magnitude;
   return p;
}
```