

Team 750C Nothing But Net Code Reference
Competition_2016-01-16_Ranney

Generated by Doxygen 1.8.11

Contents

1	Class Index	1
1.1	Class List	1
2	File Index	2
2.1	File List	2
3	Class Documentation	3
3.1	joyState Struct Reference	3
3.1.1	Detailed Description	3
3.1.2	Member Data Documentation	3
3.2	MotorGroup Struct Reference	4
3.2.1	Detailed Description	4
3.2.2	Member Data Documentation	4
4	File Documentation	5
4.1	include/autonrecorder.h File Reference	5
4.1.1	Detailed Description	6
4.1.2	Macro Definition Documentation	6
4.1.3	Typedef Documentation	7
4.1.4	Function Documentation	7
4.1.5	Variable Documentation	8
4.2	autonrecorder.h	9
4.3	include/bitwise.h File Reference	9
4.3.1	Detailed Description	9
4.3.2	Macro Definition Documentation	10
4.4	bitwise.h	11
4.5	include/constants.h File Reference	11
4.5.1	Detailed Description	11

4.5.2	Macro Definition Documentation	12
4.6	constants.h	13
4.7	include/friendly.h File Reference	13
4.7.1	Detailed Description	14
4.7.2	Macro Definition Documentation	14
4.8	friendly.h	14
4.9	include/lcddiag.h File Reference	15
4.9.1	Detailed Description	16
4.9.2	Macro Definition Documentation	16
4.9.3	Typedef Documentation	17
4.9.4	Function Documentation	17
4.9.5	Variable Documentation	19
4.10	lcddiag.h	20
4.11	include/lcdmsg.h File Reference	21
4.11.1	Detailed Description	21
4.11.2	Macro Definition Documentation	21
4.11.3	Function Documentation	22
4.11.4	Variable Documentation	22
4.12	lcdmsg.h	23
4.13	include/macros.h File Reference	23
4.13.1	Detailed Description	23
4.13.2	Macro Definition Documentation	23
4.14	macros.h	27
4.15	include/main.h File Reference	28
4.15.1	Detailed Description	28
4.15.2	Function Documentation	29
4.16	main.h	31
4.17	include/motors.h File Reference	31

4.17.1 Detailed Description	32
4.17.2 Macro Definition Documentation	32
4.17.3 Function Documentation	33
4.18 motors.h	35
4.19 include/opcontrol.h File Reference	36
4.19.1 Detailed Description	36
4.19.2 Function Documentation	37
4.19.3 Variable Documentation	37
4.20 opcontrol.h	38
4.21 include/sensors.h File Reference	38
4.21.1 Detailed Description	39
4.21.2 Macro Definition Documentation	39
4.21.3 Function Documentation	41
4.21.4 Variable Documentation	41
4.22 sensors.h	42
4.23 src/auto.c File Reference	42
4.23.1 Detailed Description	43
4.23.2 Function Documentation	43
4.24 auto.c	44
4.25 src/autonrecorder.c File Reference	44
4.25.1 Detailed Description	44
4.25.2 Function Documentation	45
4.25.3 Variable Documentation	45
4.26 autonrecorder.c	46
4.27 src/init.c File Reference	50
4.27.1 Detailed Description	50
4.27.2 Function Documentation	51
4.28 init.c	52

4.29	src/lcddiag.c File Reference	52
4.29.1	Detailed Description	53
4.29.2	Function Documentation	54
4.29.3	Variable Documentation	60
4.30	lcddiag.c	62
4.31	src/lcdmsg.c File Reference	75
4.31.1	Detailed Description	76
4.31.2	Function Documentation	76
4.31.3	Variable Documentation	76
4.32	lcdmsg.c	77
4.33	src/motors.c File Reference	77
4.33.1	Detailed Description	78
4.33.2	Function Documentation	78
4.34	motors.c	78
4.35	src/opcontrol.c File Reference	79
4.35.1	Detailed Description	79
4.35.2	Function Documentation	80
4.35.3	Variable Documentation	80
4.36	opcontrol.c	81
4.37	src/sensors.c File Reference	83
4.37.1	Detailed Description	83
4.37.2	Variable Documentation	83
4.38	sensors.c	84

Index**85****1 Class Index****1.1 Class List**

Here are the classes, structs, unions and interfaces with brief descriptions:

joyState	3
MotorGroup	4

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

include/autonrecorder.h		
Header file for autonomous recorder functions and definitions		5
include/bitwise.h		
Header file for bitwise functions and operations		9
include/constants.h		
Header file for mathematical constants		11
include/friendly.h		
Header file for human-readable definitions		13
include/lcddiag.h		
Header file for LCD diagnostic menu functions and definitions		15
include/lcdmsg.h		
Header file for LCD message code		21
include/macros.h		
Header file for macro definitions		23
include/main.h		
Header file for global functions		28
include/motors.h		
Header file for important motor functions and definitions		31
include/opcontrol.h		
Header file for operator control definitions and prototypes		36
include/sensors.h		
File for important sensor declarations and functions		38
src/auto.c		
File for autonomous code		42
src/autonrecorder.c		
File for autonomous recorder code		44
src/init.c		
File for initialization code		50
src/lcddiag.c		
File for LCD diagnostic menu code		52

src/ lcdmsg.c	
File for LCD message code	75
src/ motors.c	
File for important motor functions	77
src/ opcontrol.c	
File for operator control code	79
src/ sensors.c	
File for important sensor declarations and functions	83

3 Class Documentation

3.1 joyState Struct Reference

```
#include <autonrecorder.h>
```

Public Attributes

- signed char [spd](#)
- signed char [turn](#)
- signed char [sht](#)
- signed char [intk](#)
- signed char [trans](#)
- signed char [dep](#)

3.1.1 Detailed Description

Structure containing the joystick state variables at a point in time.

Definition at line 65 of file [autonrecorder.h](#).

3.1.2 Member Data Documentation

3.1.2.1 signed char joyState::dep

Definition at line 66 of file [autonrecorder.h](#).

3.1.2.2 signed char joyState::intk

Definition at line 66 of file [autonrecorder.h](#).

3.1.2.3 signed char joyState::sht

Definition at line 66 of file [autonrecorder.h](#).

3.1.2.4 signed char joyState::spd

Definition at line 66 of file [autonrecorder.h](#).

3.1.2.5 signed char joyState::trans

Definition at line 66 of file [autonrecorder.h](#).

3.1.2.6 signed char joyState::turn

Definition at line 66 of file [autonrecorder.h](#).

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

- include/[autonrecorder.h](#)

3.2 MotorGroup Struct Reference

```
#include <lcddiag.h>
```

Public Attributes

- bool [motor](#) [11]
- char [name](#) [LCD_MESSAGE_MAX_LENGTH+1]

3.2.1 Detailed Description

Object representing motor groups. Has flags for each motor that belongs to the group, as well as a 16-character name.

Definition at line 142 of file [lcddiag.h](#).

3.2.2 Member Data Documentation

3.2.2.1 bool MotorGroup::motor[11]

Definition at line 143 of file [lcddiag.h](#).

3.2.2.2 char MotorGroup::name[LCD_MESSAGE_MAX_LENGTH+1]

Definition at line 144 of file [lccdiag.h](#).

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

- [include/lccdiag.h](#)

4 File Documentation

4.1 [include/autonrecorder.h](#) File Reference

Header file for autonomous recorder functions and definitions.

Classes

- struct [joyState](#)

Macros

- `#define` [AUTON_TIME](#) 15
- `#define` [PROGSKILL_TIME](#) 60
- `#define` [JOY_POLL_FREQ](#) 50
- `#define` [MAX_AUTON_SLOTS](#) 10
- `#define` [AUTON_FILENAME_MAX_LENGTH](#) 8
- `#define` [AUTON_POT](#) 1
- `#define` [AUTON_BUTTON](#) 2
- `#define` [AUTON_POT_LOW](#) 0
- `#define` [AUTON_POT_HIGH](#) 4095

Typedefs

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

Functions

- void [initAutonRecorder](#) ()
- void [recordAuton](#) ()
- void [saveAuton](#) ()
- void [loadAuton](#) ()
- void [playbackAuton](#) ()

Variables

- [joyState](#) [states](#) [[AUTON_TIME](#) *[JOY_POLL_FREQ](#)]
- [int](#) [autonLoaded](#)
- [int](#) [progSkills](#)

4.1.1 Detailed Description

Header file for autonomous recorder functions and definitions.

This file contains definitions and function declarations for the autonomous recorder. These definitions provide fundamental constants and classes that the autonomous recorder uses. Additionally, this file defines the autonomous selection potentiometer and button. It also provides access to the autonomous recorder functions from other files. This allows for the recorder to be accessed during operator control.

Definition in file [autonrecorder.h](#).

4.1.2 Macro Definition Documentation

4.1.2.1 `#define AUTON_BUTTON 2`

Button for confirming selection of an autonomous routine.

Definition at line [50](#) of file [autonrecorder.h](#).

4.1.2.2 `#define AUTON_FILENAME_MAX_LENGTH 8`

Maximum file name length of autonomous routine files.

Definition at line [40](#) of file [autonrecorder.h](#).

4.1.2.3 `#define AUTON_POT 1`

Potentiometer for selecting which autonomous routine to load.

Definition at line [45](#) of file [autonrecorder.h](#).

4.1.2.4 `#define AUTON_POT_HIGH 4095`

Upper limit of the autonomous routine selector potentiometer.

Definition at line [60](#) of file [autonrecorder.h](#).

4.1.2.5 `#define AUTON_POT_LOW 0`

Lower limit of the autonomous routine selector potentiometer.

Definition at line [55](#) of file [autonrecorder.h](#).

4.1.2.6 `#define AUTON_TIME 15`

Number of seconds the autonomous period lasts.

Definition at line 18 of file [autonrecorder.h](#).

4.1.2.7 `#define JOY_POLL_FREQ 50`

Frequency to poll the joystick for recording. The joystick values will be recorded this many times per second. The joystick updates every 20 milliseconds (50 times per second).

Definition at line 30 of file [autonrecorder.h](#).

4.1.2.8 `#define MAX_AUTON_SLOTS 10`

Maximum number of autonomous routines to be stored.

Definition at line 35 of file [autonrecorder.h](#).

4.1.2.9 `#define PROGSKILL_TIME 60`

Number of seconds the programming skills challenge lasts.

Definition at line 23 of file [autonrecorder.h](#).

4.1.3 Typedef Documentation

4.1.3.1 `typedef struct joyState joyState`

Structure containing the joystick state variables at a point in time.

4.1.4 Function Documentation

4.1.4.1 `void initAutonRecorder ()`

Initializes autonomous recorder by setting joystick states array to zero.

Initializes autonomous recorder by setting states array to zero.

Definition at line 72 of file [autonrecorder.c](#).

4.1.4.2 `void loadAuton ()`

Loads autonomous file contents into states array for playback.

Loads autonomous file contents into states array.

Definition at line 216 of file [autonrecorder.c](#).

4.1.4.3 void playbackAuton ()

Replays autonomous based on loaded values in states array.

Definition at line 309 of file [autonrecorder.c](#).

4.1.4.4 void recordAuton ()

Records driver joystick values into states array for saving.

Records driver joystick values into states array.

Definition at line 88 of file [autonrecorder.c](#).

4.1.4.5 void saveAuton ()

Saves contents of the states array to a file in flash memory for later playback.

Saves contents of the states array to a file in flash memory.

Definition at line 133 of file [autonrecorder.c](#).

4.1.5 Variable Documentation

4.1.5.1 int autonLoaded

Slot number of currently loaded autonomous routine.

Definition at line 24 of file [autonrecorder.c](#).

4.1.5.2 int progSkills

Section number (0-3) of currently loaded programming skills routine. Since programming skills lasts for 60 seconds, it can be represented by 4 standard autonomous recordings.

Section number (0-3) of currently loaded programming skills routine.

Definition at line 29 of file [autonrecorder.c](#).

4.1.5.3 joyState states[AUTON_TIME*JOY_POLL_FREQ]

Stores the joystick state variables for moving the robot. Used for recording and playing back autonomous routines.

Definition at line 19 of file [autonrecorder.c](#).

4.2 autonrecorder.h

```

00001
00012 #ifndef AUTONRECORDER_H
00013 #define AUTONRECORDER_H
00014
00018 #define AUTON_TIME 15
00019
00023 #define PROGSKILL_TIME 60
00024
00030 #define JOY_POLL_FREQ 50
00031
00035 #define MAX_AUTON_SLOTS 10
00036
00040 #define AUTON_FILENAME_MAX_LENGTH 8
00041
00045 #define AUTON_POT 1
00046
00050 #define AUTON_BUTTON 2
00051
00055 #define AUTON_POT_LOW 0
00056
00060 #define AUTON_POT_HIGH 4095
00061
00065 typedef struct joyState {
00066     signed char spd, turn, sht, intk, trans, dep;
00067 } joyState;
00068
00073 extern joyState states[AUTON_TIME*JOY_POLL_FREQ];
00074
00078 extern int autonLoaded;
00079
00084 extern int progSkills;
00085
00089 void initAutonRecorder();
00090
00094 void recordAuton();
00095
00099 void saveAuton();
00100
00104 void loadAuton();
00105
00109 void playbackAuton();
00110
00111 #endif

```

4.3 include/bitwise.h File Reference

Header file for bitwise functions and operations.

Macros

- #define [bitRead](#)(value, bit) (((value) >> (bit)) & 0x01)
- #define [bitSet](#)(value, bit) ((value) |= (1UL << (bit)))
- #define [bitClear](#)(value, bit) ((value) &= ~(1UL << (bit)))
- #define [bitWrite](#)(value, bit, bitvalue) (bitvalue ? [bitSet](#)(value, bit) : [bitClear](#)(value, bit))

4.3.1 Detailed Description

Header file for bitwise functions and operations.

This file provides macro definitions for bitwise manipulation of variables. These definitions provide a more human-readable way to manipulate individual bits.

Definition in file [bitwise.h](#).

4.3.2 Macro Definition Documentation

4.3.2.1 `#define bitClear(value, bit) ((value) &= ~(1UL << (bit)))`

Clears the value of a bit in a variable to 0.

Parameters

<i>value</i>	the variable to clear in
<i>bit</i>	the bit number to clear (0 being the rightmost)

Definition at line 35 of file [bitwise.h](#).

4.3.2.2 `#define bitRead(value, bit) (((value) >> (bit)) & 0x01)`

Reads the value of a bit (1 or 0) from a variable.

Parameters

<i>value</i>	the variable to read from
<i>bit</i>	the bit number to read (0 being the rightmost)

Returns

the value of the bit (1 or 0)

Definition at line 19 of file [bitwise.h](#).

4.3.2.3 `#define bitSet(value, bit) ((value) |= (1UL << (bit)))`

Sets the value of a bit in a variable to 1.

Parameters

<i>value</i>	the variable to set in
<i>bit</i>	the bit number to set (0 being the rightmost)

Definition at line 27 of file [bitwise.h](#).

4.3.2.4 `#define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))`

Writes a value (1 or 0) to a bit in a variable.

Parameters

<i>value</i>	the variable to write in
--------------	--------------------------

Parameters

<i>bit</i>	the bit number to set (0 being the rightmost)
<i>bitvalue</i>	the value to write (1 or 0)

Definition at line 44 of file [bitwise.h](#).

4.4 bitwise.h

```

00001
00008 #ifndef BITWISE_H_
00009 #define BITWISE_H_
00010
00019 #define bitRead(value, bit) (((value) >> (bit)) & 0x01)
00020
00027 #define bitSet(value, bit) ((value) |= (1UL << (bit)))
00028
00035 #define bitClear(value, bit) ((value) &= ~(1UL << (bit)))
00036
00044 #define bitWrite(value, bit, bitvalue) (bitvalue ? bitSet(value, bit) : bitClear(value, bit))
00045
00046 #endif

```

4.5 include/constants.h File Reference

Header file for mathematical constants.

Macros

- `#define MATH_PI 3.141592653589793238462643383279`
- `#define MATH_E 2.718281828459045`
- `#define PI 3.1415926535897932384626433832795`
- `#define HALF_PI 1.5707963267948966192313216916398`
- `#define TWO_PI 6.283185307179586476925286766559`
- `#define TAU 6.283185307179586476925286766559`
- `#define DEG_TO_RAD 0.017453292519943295769236907684886`
- `#define RAD_TO_DEG 57.295779513082320876798154814105`
- `#define ROTATION_DEG 360`
- `#define ROTATION_RAD TWO_PI`

4.5.1 Detailed Description

Header file for mathematical constants.

This file provides constant definitions for various mathematical values that appear frequently. These definitions provide a more human-readable way to do math operations on variables.

Definition in file [constants.h](#).

4.5.2 Macro Definition Documentation

4.5.2.1 `#define DEG_TO_RAD 0.017453292519943295769236907684886`

Conversion factor from degrees to radians.

Definition at line 45 of file [constants.h](#).

4.5.2.2 `#define HALF_PI 1.5707963267948966192313216916398`

Half the value of pi.

Definition at line 30 of file [constants.h](#).

4.5.2.3 `#define MATH_E 2.718281828459045`

The mathematical constant e (Euler's Number).

Definition at line 20 of file [constants.h](#).

4.5.2.4 `#define MATH_PI 3.141592653589793238462643383279`

The mathematical constant pi.

Definition at line 15 of file [constants.h](#).

4.5.2.5 `#define PI 3.1415926535897932384626433832795`

The mathematical constant pi.

Definition at line 25 of file [constants.h](#).

4.5.2.6 `#define RAD_TO_DEG 57.295779513082320876798154814105`

Conversion factor from radians to degrees.

Definition at line 50 of file [constants.h](#).

4.5.2.7 `#define ROTATION_DEG 360`

Amount of degrees in a circle.

Definition at line 55 of file [constants.h](#).

4.5.2.8 `#define ROTATION_RAD TWO_PI`

Amount of radians in a circle.

Definition at line 60 of file [constants.h](#).

4.5.2.9 #define TAU 6.283185307179586476925286766559

The mathematical constant tau (two times the value of pi).

Definition at line 40 of file [constants.h](#).

4.5.2.10 #define TWO_PI 6.283185307179586476925286766559

Two times the value of pi (the mathematical constant tau).

Definition at line 35 of file [constants.h](#).

4.6 constants.h

```
00001
00009 #ifndef CONSTANTS_H_
00010 #define CONSTANTS_H_
00011
00015 #define MATH_PI 3.141592653589793238462643383279
00016
00020 #define MATH_E 2.718281828459045
00021
00025 #define PI 3.1415926535897932384626433832795
00026
00030 #define HALF_PI 1.5707963267948966192313216916398
00031
00035 #define TWO_PI 6.283185307179586476925286766559
00036
00040 #define TAU 6.283185307179586476925286766559
00041
00045 #define DEG_TO_RAD 0.017453292519943295769236907684886
00046
00050 #define RAD_TO_DEG 57.295779513082320876798154814105
00051
00055 #define ROTATION_DEG 360
00056
00060 #define ROTATION_RAD TWO_PI
00061
00062 #endif
```

4.7 include/friendly.h File Reference

Header file for human-readable definitions.

Macros

- #define [PRESSED](#) LOW
- #define [UNPRESSED](#) HIGH
- #define [UNRELEASED](#) LOW
- #define [RELEASED](#) HIGH

4.7.1 Detailed Description

Header file for human-readable definitions.

This file provides constant definitions for various sensory states that appear frequently. Since the button pressed state is represented by an unintuitive LOW value, these constants create definitions for the pressed and unpressed states of a button.

Definition in file [friendly.h](#).

4.7.2 Macro Definition Documentation

4.7.2.1 #define PRESSED LOW

More readable definition for button pressed/unreleased state.

Definition at line 16 of file [friendly.h](#).

4.7.2.2 #define RELEASED HIGH

More readable definition for button released/unpressed state.

Definition at line 31 of file [friendly.h](#).

4.7.2.3 #define UNPRESSED HIGH

More readable definition for button unpressed/released state.

Definition at line 21 of file [friendly.h](#).

4.7.2.4 #define UNRELEASED LOW

More readable definition for button unreleased/pressed state.

Definition at line 26 of file [friendly.h](#).

4.8 friendly.h

```
00001
00010 #ifndef FRIENDLY_H_
00011 #define FRIENDLY_H_
00012
00016 #define PRESSED LOW
00017
00021 #define UNPRESSED HIGH
00022
00026 #define UNRELEASED LOW
00027
00031 #define RELEASED HIGH
00032
00033 #endif
00034
```

4.9 include/lcddiag.h File Reference

Header file for LCD diagnostic menu functions and definitions.

Classes

- struct [MotorGroup](#)

Macros

- #define [LCD_MENU_COUNT](#) 9
- #define [MENU_MOTOR](#) 0
- #define [MENU_MOTOR_MGMT](#) 1
- #define [MENU_BATTERY](#) 2
- #define [MENU_CONNECTION](#) 3
- #define [MENU_ROBOT](#) 4
- #define [MENU_AUTON](#) 5
- #define [MENU_BACKLIGHT](#) 6
- #define [MENU_SCREENSAVER](#) 7
- #define [MENU_CREDITS](#) 8

Typedefs

- typedef struct [MotorGroup](#) [MotorGroup](#)

Functions

- bool [lcdButtonPressed](#) (int btn)
- bool [lcdAnyButtonPressed](#) ()
- void [initGroups](#) ()
- char * [typeString](#) (char *dest)
- void [formatLCDDisplay](#) (void *ignore)

Variables

- char [menuChoices](#) [[LCD_MENU_COUNT](#)][[LCD_MESSAGE_MAX_LENGTH](#)+1]
- TaskHandle [lcdDiagTask](#)
- [MotorGroup](#) * [groups](#)
- int [numgroups](#)
- bool [disableOpControl](#)

4.9.1 Detailed Description

Header file for LCD diagnostic menu functions and definitions.

This file contains function prototypes and definitions for the LCD diagnostic menu. The menu provides live debugging and testing functionality. It provides the following functions:

- Motor testing functionality (individual and group)
- Motor group management
- Battery voltage information
- Joystick connection status
- Robot sensory data
- Autonomous recorder status
- LCD backlight toggle
- Screensaver that displays during operator control
- Credits menu

This file maintains constants and internal states regarding the menu's functionality.

The idea behind this was inspired by Team 750W and Akram Sandhu. Without them, this project would not be possible.

Note: the implementation of this feature is completely different between the two teams. No code was reused from their implementation of the LCD diagnostic menu.

Definition in file [lcddiag.h](#).

4.9.2 Macro Definition Documentation

4.9.2.1 `#define LCD_MENU_COUNT 9`

The number of top-level menus available in the LCD diagnostic menu system.

Definition at line [78](#) of file [lcddiag.h](#).

4.9.2.2 `#define MENU_AUTON 5`

Menu ID number of the autonomous recorder status indicator.

Definition at line [108](#) of file [lcddiag.h](#).

4.9.2.3 `#define MENU_BACKLIGHT 6`

Menu ID number of the backlight toggle.

Definition at line [113](#) of file [lcddiag.h](#).

4.9.2.4 #define MENU_BATTERY 2

Menu ID number of the battery voltage readout.

Definition at line 93 of file [lcddiag.h](#).

4.9.2.5 #define MENU_CONNECTION 3

Menu ID number of the joystick connection state indicator.

Definition at line 98 of file [lcddiag.h](#).

4.9.2.6 #define MENU_CREDITS 8

Menu ID number of the credits menu. Thanks again to Team 750W and Akram Sandhu.

Definition at line 124 of file [lcddiag.h](#).

4.9.2.7 #define MENU_MOTOR 0

Menu ID number of the motor testing menu.

Definition at line 83 of file [lcddiag.h](#).

4.9.2.8 #define MENU_MOTOR_MGMT 1

Menu ID number of the motor group manager.

Definition at line 88 of file [lcddiag.h](#).

4.9.2.9 #define MENU_ROBOT 4

Menu ID number of the robot sensory state indicator.

Definition at line 103 of file [lcddiag.h](#).

4.9.2.10 #define MENU_SCREENSAVER 7

Menu ID number of the LCD message screensaver.

Definition at line 118 of file [lcddiag.h](#).

4.9.3 Typedef Documentation

4.9.3.1 typedef struct MotorGroup MotorGroup

Object representing motor groups. Has flags for each motor that belongs to the group, as well as a 16-character name.

4.9.4 Function Documentation

4.9.4.1 void formatLCDDisplay (void * *ignore*)

Runs the LCD diagnostic menu task. The task starts in screensaver mode. Pressing any button cancels screensaver mode and enters the selection menu.

Parameters

<i>ignore</i>	does nothing - required by task definition
---------------	--

Runs the LCD diagnostic menu task. This thread executes concurrently with the operator control task. The LCD diagnostic menu starts in screensaver mode. Pressing any button cancels screensaver mode and enters the selection menu.

Parameters

<i>ignore</i>	does nothing - required by task definition
---------------	--

Definition at line 1300 of file [lcddiag.c](#).

4.9.4.2 void initGroups ()

Initializes the motor groups array to contain the standard set of groups. This includes: Left Drive, Right Drive, Full Drive, Nautilus Shooter, Intake, and Transmission.

Definition at line 230 of file [lcddiag.c](#).

4.9.4.3 bool lcdAnyButtonPressed () [inline]

Checks if the any LCD button is pressed. This function waits for the button to be released before terminating. Due to this, it can only be called once per loop iteration. Its value should be stored in a boolean variable.

Returns

true if pressed, false if not

Definition at line 64 of file [lcddiag.h](#).

4.9.4.4 bool lcdButtonPressed (int btn) [inline]

Checks if the specified LCD button is pressed. This function's valid parameters are:

- LCD_BTN_LEFT
- LCD_BTN_RIGHT
- LCD_BTN_CENTER.

This function waits for the specified button to be released before terminating. Due to this, it can only be called once per loop iteration. Its value should be stored in a boolean variable.

Parameters

<i>btn</i>	the button to check
------------	---------------------

Returns

true if pressed, false if not

Definition at line 44 of file [lcddiag.h](#).

4.9.4.5 char* typeString (char * dest)

Uses the LCD and the autonomous potentiometer to type a string. This is used to name motor groups and autonomous recordings. The maximum length of string this function can type is 16 characters.

Parameters

<i>dest</i>	a buffer to store the typed string (must be at least 17 characters to hold null terminator)
-------------	---

Returns

a pointer to the buffer

Definition at line 81 of file [lcddiag.c](#).

4.9.5 Variable Documentation**4.9.5.1 bool disableOpControl**

Disables operator control loop during motor testing. Since running motors is not thread safe, it is necessary to stop operator control of the motors during testing.

Definition at line 42 of file [lcddiag.c](#).

4.9.5.2 MotorGroup* groups

Array that stores the motor groups. As this is a dynamic array, creating and editing new motor groups is possible. These motor groups are added to the array via the Motor Group Management menu.

Definition at line 49 of file [lcddiag.c](#).

4.9.5.3 TaskHandle lcdDiagTask

Object representing the LCD diagnostic menu task. The LCD diagnostic menu runs in a separate thread from the operator control code. The TaskHandle allows for pausing and resuming of the LCD diagnostic menu during autonomous recording.

Definition at line 31 of file [lcddiag.c](#).

4.9.5.4 char menuChoices[LCD_MENU_COUNT][LCD_MESSAGE_MAX_LENGTH+1]

Stores the top-level menu names.

Definition at line 60 of file [lcddiag.c](#).

4.9.5.5 int numgroups

Stores the number of motor groups. This is functionally identical to the size of the motor group array.

Definition at line 55 of file [lcddiag.c](#).

4.10 lcddiag.h

```

00001
00026 #ifndef LCDDIAG_H
00027 #define LCDDIAG_H
00028
00044 inline bool lcdButtonPressed(int btn){
00045     if(lcdReadButtons(LCD_PORT) & btn){
00046         do{
00047             delay(20);
00048         } while (lcdReadButtons(LCD_PORT) & btn);
00049         return true;
00050     } else {
00051         return false;
00052     }
00053     return true;
00054 }
00055
00064 inline bool lcdAnyButtonPressed(){
00065     if(lcdReadButtons(LCD_PORT)){
00066         do{
00067             delay(20);
00068         } while (lcdReadButtons(LCD_PORT));
00069     } else {
00070         return false;
00071     }
00072     return true;
00073 }
00074
00078 #define LCD_MENU_COUNT 9
00079
00083 #define MENU_MOTOR 0
00084
00088 #define MENU_MOTOR_MGMT 1
00089
00093 #define MENU_BATTERY 2
00094
00098 #define MENU_CONNECTION 3
00099
00103 #define MENU_ROBOT 4
00104
00108 #define MENU_AUTON 5
00109
00113 #define MENU_BACKLIGHT 6
00114
00118 #define MENU_SCREENSAVER 7
00119
00124 #define MENU_CREDITS 8
00125
00129 extern char menuChoices[LCD_MENU_COUNT][
    LCD_MESSAGE_MAX_LENGTH+1];
00130
00136 extern TaskHandle lcdDiagTask;
00137
00142 typedef struct MotorGroup {
00143     bool motor[11];
00144     char name[LCD_MESSAGE_MAX_LENGTH+1];
00145 } MotorGroup;

```



```
00146
00152 extern MotorGroup *groups;
00153
00158 extern int numgroups;
00159
00164 extern bool disableOpControl;
00165
00170 void initGroups();
00171
00181 char* typeString(char* dest);
00182
00190 void formatLCDDisplay(void* ignore);
00191
00192 #endif
```

4.11 include/lcdmsg.h File Reference

Header file for LCD message code.

Macros

- `#define LCD_PORT` `uart1`
- `#define LCD_750C_TITLE` `"$$$ 750C $$$ "`
- `#define LCD_MESSAGE_COUNT` `44`
- `#define LCD_MESSAGE_MAX_LENGTH` `16`

Functions

- void `randlcdmsg` (FILE *lcdport, int line)
- void `screensaver` (FILE *lcdport)

Variables

- char * `lcdmsg` []

4.11.1 Detailed Description

Header file for LCD message code.

This file contains definitions for the LCD screensaver messages. This file also defines the LCD port. These messages display randomly while the LCD diagnostic menu is set to the screensaver mode. These messages are mainly inside jokes among the team.

Definition in file [lcdmsg.h](#).

4.11.2 Macro Definition Documentation

4.11.2.1 `#define LCD_750C_TITLE " $$$ 750C $$$ "`

Defines title string for LCD to display.

Definition at line [21](#) of file [lcdmsg.h](#).

4.11.2.2 `#define LCD_MESSAGE_COUNT 44`

Defines the amount of LCD messages in the master list.

Definition at line 26 of file [lcdmsg.h](#).

4.11.2.3 `#define LCD_MESSAGE_MAX_LENGTH 16`

Defines the max length for LCD messages.

Definition at line 31 of file [lcdmsg.h](#).

4.11.2.4 `#define LCD_PORT uart1`

Defines the port the LCD is plugged into.

Definition at line 16 of file [lcdmsg.h](#).

4.11.3 Function Documentation

4.11.3.1 `void randlcdmsg (FILE * lcdport, int line)`

Displays a random LCD message from the master list.

Parameters

<i>lcdport</i>	the port the LCD is connected to
<i>line</i>	the line to display the message on

Definition at line 67 of file [lcdmsg.c](#).

4.11.3.2 `void screensaver (FILE * lcdport)`

Formats the LCD by displaying 750C title and message.

Parameters

<i>lcdport</i>	the port the LCD is connected to
----------------	----------------------------------

Definition at line 86 of file [lcdmsg.c](#).

4.11.4 Variable Documentation

4.11.4.1 char* lcdmsg[]

Master list of all LCD messages.

Definition at line 14 of file [lcdmsg.c](#).

4.12 lcdmsg.h

```

00001
00010 #ifndef LCDMSG_H_
00011 #define LCDMSG_H_
00012
00016 #define LCD_PORT uart1
00017
00021 #define LCD_750C_TITLE " $$$ 750C $$$ "
00022
00026 #define LCD_MESSAGE_COUNT 44
00027
00031 #define LCD_MESSAGE_MAX_LENGTH 16
00032
00036 extern char *lcdmsg[];
00037
00044 void randlcdmsg(FILE *lcdport, int line);
00045
00051 void screensaver(FILE *lcdport);
00052
00053 #endif

```

4.13 include/macros.h File Reference

Header file for macro definitions.

Macros

- `#define min(a, b) ((a)<(b)?(a):(b))`
- `#define MIN(a, b) ((a)<(b)?(a):(b))`
- `#define max(a, b) ((a)>(b)?(a):(b))`
- `#define MAX(a, b) ((a)>(b)?(a):(b))`
- `#define abs(x) ((x)>0?(x):- (x))`
- `#define constrain(amt, low, high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))`
- `#define round(x) (((x) >=0) ?(long)((x)+0.5):(long)((x)-0.5))`
- `#define sign(x) ((x)>0?1:((x)<0?-1:0))`
- `#define radians(deg) ((deg)*DEG_TO_RAD)`
- `#define degrees(rad) ((rad)*RAD_TO_DEG)`
- `#define sq(x) ((x)*(x))`
- `#define SQ(x) ((x)*(x))`

4.13.1 Detailed Description

Header file for macro definitions.

This file provides macro definitions for various basic functions that come about frequently.

Definition in file [macros.h](#).

4.13.2 Macro Definition Documentation

4.13.2.1 `#define abs(x) ((x)>0?(x):- (x))`

Returns the absolute value of the input value.

Parameters

<i>x</i>	the input value
----------	-----------------

Returns

the absolute value of *x*

Definition at line 57 of file [macros.h](#).

4.13.2.2 `#define constrain(amt, low, high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))`

Constrains a value to a set of boundaries.

Parameters

<i>amt</i>	the value to constrain
<i>low</i>	the lower bound
<i>high</i>	the higher bound

Returns

high, amt, or low if amt is higher, in between, or lower than the range specified, respectively

Definition at line 68 of file [macros.h](#).

4.13.2.3 `#define degrees(rad) ((rad)*RAD_TO_DEG)`

Converts an angle measure in degrees to radians.

Parameters

<i>rad</i>	the angle measure in radians
------------	------------------------------

Returns

the angle measure in degrees

Definition at line 104 of file [macros.h](#).

4.13.2.4 `#define max(a, b) ((a)>(b)?(a):(b))`

Returns the maximum of the two values.

Parameters

<i>a</i>	the first input value
<i>b</i>	the second input value

Returns

the greater of the two values

Definition at line 38 of file [macros.h](#).

4.13.2.5 `#define MAX(a, b) ((a)>(b)?(a):(b))`

Returns the maximum of the two values.

Parameters

<i>a</i>	the first input value
<i>b</i>	the second input value

Returns

the greater of the two values

Definition at line 48 of file [macros.h](#).

4.13.2.6 `#define min(a, b) ((a)<(b)?(a):(b))`

Returns the minimum of the two values.

Parameters

<i>a</i>	the first input value
<i>b</i>	the second input value

Returns

the lesser of the two values

Definition at line 18 of file [macros.h](#).

4.13.2.7 `#define MIN(a, b) ((a)<(b)?(a):(b))`

Returns the minimum of the two values.

Parameters

<i>a</i>	the first input value
<i>b</i>	the second input value

Returns

the lesser of the two values

Definition at line 28 of file [macros.h](#).

4.13.2.8 #define radians(*deg*) ((deg)*DEG_TO_RAD)

Converts an angle measure in degrees to radians.

Parameters

<i>deg</i>	the angle measure in degrees
------------	------------------------------

Returns

the angle measure in radians

Definition at line 95 of file [macros.h](#).

4.13.2.9 #define round(*x*) (((x) >= 0) ? (long)((x)+0.5) : (long)((x)-0.5))

Rounds a value to the nearest integer.

Parameters

<i>x</i>	the value to round
----------	--------------------

Returns

the rounded value

Definition at line 77 of file [macros.h](#).

4.13.2.10 #define sign(*x*) ((x)>0?1:((x)<0?-1:0))

Returns the sign of the input value.

Parameters

x	the value to determine the sign of
---	------------------------------------

Returns

-1, 0, or 1 if the value is negative, zero, or positive, respectively

Definition at line 86 of file [macros.h](#).

4.13.2.11 `#define sq(x) ((x)*(x))`

Squares the input value.

Parameters

x	the value to square
---	---------------------

Returns

the square of the input value

Definition at line 113 of file [macros.h](#).

4.13.2.12 `#define SQ(x) ((x)*(x))`

Squares the input value.

Parameters

x	the value to square
---	---------------------

Returns

the square of the input value

Definition at line 122 of file [macros.h](#).

4.14 macros.h

```
00001
00007 #ifndef MACROS_H_
00008 #define MACROS_H_
00009
00018 #define min(a,b) ((a)<(b)?(a):(b))
00019
```

```

00028 #define MIN(a,b) ((a)<(b)?(a):(b))
00029
00038 #define max(a,b) ((a)>(b)?(a):(b))
00039
00048 #define MAX(a,b) ((a)>(b)?(a):(b))
00049
00057 #define abs(x) ((x)>0?(x):- (x))
00058
00068 #define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))
00069
00077 #define round(x) (((x) >=0) ? (long) ((x)+0.5) : (long) ((x)-0.5))
00078
00086 #define sign(x) ((x)>0?1:((x)<0?-1:0))
00087
00095 #define radians(deg) ((deg)*DEG_TO_RAD)
00096
00104 #define degrees(rad) ((rad)*RAD_TO_DEG)
00105
00113 #define sq(x) ((x)*(x))
00114
00122 #define SQ(x) ((x)*(x))
00123
00124 #endif

```

4.15 include/main.h File Reference

Header file for global functions.

```

#include <stdint.h>
#include <string.h>
#include <API.h>
#include <constants.h>
#include <friendly.h>
#include <macros.h>
#include <bitwise.h>
#include <sensors.h>
#include <motors.h>
#include <lcdmsg.h>
#include <lcddiag.h>
#include <autonrecorder.h>
#include <opcontrol.h>

```

Functions

- void [autonomous](#) ()
- void [initializeIO](#) ()
- void [initialize](#) ()
- void [operatorControl](#) ()

4.15.1 Detailed Description

Header file for global functions.

Any experienced C or C++ programmer knows the importance of header files. For those who do not, a header file allows multiple files to reference functions in other files without necessarily having to see the code (and therefore

causing a multiple definition). To make a function in "opcontrol.c", "auto.c", "main.c", or any other C file visible to the core implementation files, prototype it here.

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

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

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

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

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

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

Definition in file [main.h](#).

4.15.2 Function Documentation

4.15.2.1 void autonomous ()

C standard integer type header C standard string function header PROS main API header Mathematical constant definitions Useful redefinitions to make code more readable Simple macros for performing common operations Macros for performing bitwise operations Sensor definitions and function declarations Motor definitions and function declarations LCD definitions and function declarations LCD diagnostics menu definitions and function declarations Autonomous recorder definitions and function declarations Operator control definitions and function declarations 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 51 of file [auto.c](#).

4.15.2.2 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 61 of file [init.c](#).

4.15.2.3 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 45 of file [init.c](#).

4.15.2.4 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 153 of file [opcontrol.c](#).

4.16 main.h

```

00001
00041 #ifndef MAIN_H_
00042 #define MAIN_H_
00043
00047 #include <stdint.h>
00048
00052 #include <string.h>
00053
00057 #include <API.h>
00058
00062 #include <constants.h>
00063
00067 #include <friendly.h>
00068
00072 #include <macros.h>
00073
00077 #include <bitwise.h>
00078
00082 #include <sensors.h>
00083
00087 #include <motors.h>
00088
00092 #include <lcdmsg.h>
00093
00097 #include <lcddiag.h>
00098
00102 #include <autonrecorder.h>
00103
00107 #include <opcontrol.h>
00108
00109 // Allow usage of this file in C++ programs
00110 #ifdef __cplusplus
00111 extern "C" {
00112 #endif
00113
00114 // A function prototype looks exactly like its declaration, but with a semicolon instead of
00115 // actual code. If a function does not match a prototype, compile errors will occur.
00116
00117 // Prototypes for initialization, operator control and autonomous
00118
00133 void autonomous();
00142 void initializeIO();
00156 void initialize();
00174 void operatorControl();
00175
00176 // End C++ export structure
00177 #ifdef __cplusplus
00178 }
00179 #endif
00180
00181 #endif

```

4.17 include/motors.h File Reference

Header file for important motor functions and definitions.

Macros

- #define MOTOR_MAX 127
- #define MOTOR_MIN -127
- #define TRANSMISSION_MOTOR 1
- #define LEFT_MOTOR_TOP 2
- #define LEFT_MOTOR_BOT 6
- #define RIGHT_MOTOR_TOP 4
- #define RIGHT_MOTOR_BOT 5

- `#define SHOOTER_HAS_THREE_MOTORS`
- `#define NAUTILUS_SHOOTER_MOTOR_LEFT 3`
- `#define NAUTILUS_SHOOTER_MOTOR_RIGHT 7`
- `#define NAUTILUS_SHOOTER_MOTOR_CENTER 10`
- `#define INTAKE_ROLLER_MOTOR 8`
- `#define LIFT_DEPLOY 9`

Functions

- void `transmission` (int `spd`)
- void `transmissionSetPos` (void *pos)
- void `changeGear` (int gear)
- void `move` (int `spd`, int `turn`)
- void `shoot` (int `spd`)
- void `intake` (int `spd`)
- void `deploy` (int `spd`)

4.17.1 Detailed Description

Header file for important motor functions and definitions.

This file contains the code for functions and definitions regarding motor status. Mainly, this file defines the motor ports for each mechanism. It also defines certain motor-related constants. Lastly, basic movement functions are defined as inline in this file.

Some functions are too complex to be defined as inline functions in the `motors.h` file. See the `motors.c` file for these more complicated movement functions.

See also

[motors.c](#)

Definition in file [motors.h](#).

4.17.2 Macro Definition Documentation

4.17.2.1 `#define INTAKE_ROLLER_MOTOR 8`

Defines motor ports for the intake mechanism.

Definition at line 111 of file [motors.h](#).

4.17.2.2 `#define LEFT_MOTOR_BOT 6`

Definition at line 63 of file [motors.h](#).

4.17.2.3 #define LEFT_MOTOR_TOP 2

Defines motor ports for the left side of the drivetrain.

Definition at line 62 of file [motors.h](#).

4.17.2.4 #define LIFT_DEPLOY 9

Definition at line 122 of file [motors.h](#).

4.17.2.5 #define MOTOR_MAX 127

Defines maximum motor speed value.

Definition at line 21 of file [motors.h](#).

4.17.2.6 #define MOTOR_MIN -127

Defines motor minimum speed value.

Definition at line 26 of file [motors.h](#).

4.17.2.7 #define NAUTILUS_SHOOTER_MOTOR_CENTER 10

Definition at line 92 of file [motors.h](#).

4.17.2.8 #define NAUTILUS_SHOOTER_MOTOR_LEFT 3

Definition at line 89 of file [motors.h](#).

4.17.2.9 #define NAUTILUS_SHOOTER_MOTOR_RIGHT 7

Definition at line 90 of file [motors.h](#).

4.17.2.10 #define RIGHT_MOTOR_BOT 5

Definition at line 69 of file [motors.h](#).

4.17.2.11 #define RIGHT_MOTOR_TOP 4

Defines motor ports for the right side of the drivetrain.

Definition at line 68 of file [motors.h](#).

4.17.2.12 #define SHOOTER_HAS_THREE_MOTORS

Defines motor ports for the nautilus gear shooting mechanism.

Definition at line 87 of file [motors.h](#).

4.17.2.13 #define TRANSMISSION_MOTOR 1

Defines motor port for the transmission to change between driving and lifting

Definition at line 31 of file [motors.h](#).

4.17.3 Function Documentation

4.17.3.1 void changeGear (int gear) [inline]

Changes the gear of the transmission to either driving or lifting. Runs a task in a separate thread to change the gear.

Parameters

<i>gear</i>	the gear to change to
-------------	-----------------------

Definition at line 55 of file [motors.h](#).

4.17.3.2 void deploy (int *spd*) [inline]

Definition at line 124 of file [motors.h](#).

4.17.3.3 void intake (int *spd*) [inline]

Intakes balls using the intake mechanism by setting the intake motor values.

Parameters

<i>spd</i>	the speed to set the intake motors
------------	------------------------------------

Definition at line 118 of file [motors.h](#).

4.17.3.4 void move (int *spd*, int *turn*) [inline]

Moves the robot by setting the drive motor values.

Parameters

<i>spd</i>	the forward/backward speed value
<i>turn</i>	the turning speed value

Definition at line 77 of file [motors.h](#).

4.17.3.5 void shoot (int *spd*) [inline]

Shoots balls from the shooter mechanism by setting the shooter motor values.

Parameters

<i>spd</i>	the speed to set the shooter motors
------------	-------------------------------------

Definition at line 100 of file [motors.h](#).

4.17.3.6 void transmission (int *spd*) [inline]

Runs the transmission motor by setting the motor value.

Parameters

<i>spd</i>	the speed to run the transmission motor
------------	---

Definition at line 38 of file [motors.h](#).

4.17.3.7 void transmissionSetPos (void * pos)

Sets the position of the transmission.

Parameters

<i>pos</i>	the position to set the transmission to.
------------	--

Definition at line 19 of file [motors.c](#).

4.18 motors.h

```

00001
00015 #ifndef MOTORS_H_
00016 #define MOTORS_H_
00017
00021 #define MOTOR_MAX 127
00022
00026 #define MOTOR_MIN -127
00027
00031 #define TRANSMISSION_MOTOR 1
00032
00038 inline void transmission(int spd){
00039     motorSet(TRANSMISSION_MOTOR, spd);
00040 }
00041
00047 void transmissionSetPos(void *pos);
00048
00055 inline void changeGear(int gear){
00056     taskCreate(transmissionSetPos, TASK_DEFAULT_STACK_SIZE, (void *) (intptr_t) gear,
00057         TASK_PRIORITY_DEFAULT);
00058 }
00058
00062 #define LEFT_MOTOR_TOP 2
00063 #define LEFT_MOTOR_BOT 6
00064
00068 #define RIGHT_MOTOR_TOP 4
00069 #define RIGHT_MOTOR_BOT 5
00070
00077 inline void move(int spd, int turn){
00078     motorSet(LEFT_MOTOR_TOP, spd + turn);
00079     motorSet(LEFT_MOTOR_BOT, spd + turn);
00080     motorSet(RIGHT_MOTOR_TOP, -spd + turn);
00081     motorSet(RIGHT_MOTOR_BOT, -spd + turn);
00082 }
00083
00087 #define SHOOTER_HAS_THREE_MOTORS
00088
00089 #define NAUTILUS_SHOOTER_MOTOR_LEFT 3
00090 #define NAUTILUS_SHOOTER_MOTOR_RIGHT 7
00091 #ifdef SHOOTER_HAS_THREE_MOTORS
00092 #define NAUTILUS_SHOOTER_MOTOR_CENTER 10
00093 #endif
00094
00100 inline void shoot(int spd){
00101     motorSet(NAUTILUS_SHOOTER_MOTOR_LEFT, spd);
00102     motorSet(NAUTILUS_SHOOTER_MOTOR_RIGHT, -spd);
00103 #ifdef SHOOTER_HAS_THREE_MOTORS

```

```

00104     motorSet (NAUTILUS_SHOOTER_MOTOR_CENTER, -spd);
00105 #endif /* SHOOTER_HAS_THREE_MOTORS */
00106 }
00107
00111 #define INTAKE_ROLLER_MOTOR 8
00112
00118 inline void intake(int spd){
00119     motorSet (INTAKE_ROLLER_MOTOR, spd);
00120 }
00121
00122 #define LIFT_DEPLOY 9
00123
00124 inline void deploy(int spd){
00125     motorSet (LIFT_DEPLOY, spd);
00126 }
00127
00128 #ifndef SHOOTER_HAS_THREE_MOTORS
00129
00132 #define SHOOTER_ANGLE_MOTOR 10
00133
00139 inline void adjust(int spd){
00140     motorSet (SHOOTER_ANGLE_MOTOR, spd);
00141 }
00142 #endif /* SHOOTER_HAS_THREE_MOTORS */
00143
00144 #endif

```

4.19 include/opcontrol.h File Reference

Header file for operator control definitions and prototypes.

Functions

- void [recordJoyInfo](#) ()
- void [moveRobot](#) ()

Variables

- int [spd](#)
- int [turn](#)
- int [sht](#)
- int [intk](#)
- int [trans](#)
- int [dep](#)

4.19.1 Detailed Description

Header file for operator control definitions and prototypes.

This file contains definitions of the internal motor state variables and prototypes for functions that record these variables and move the robot based on their value.

Definition in file [opcontrol.h](#).

4.19.2 Function Documentation

4.19.2.1 void moveRobot ()

Moves the robot based on the motor state variables.

Definition at line 128 of file [opcontrol.c](#).

4.19.2.2 void recordJoyInfo ()

Populates the motor state variables based on the joystick's current values.

Definition at line 81 of file [opcontrol.c](#).

4.19.3 Variable Documentation

4.19.3.1 int dep

Speed of the lift deployment motor.

Definition at line 65 of file [opcontrol.c](#).

4.19.3.2 int intk

Speed of the intake motors.

Definition at line 55 of file [opcontrol.c](#).

4.19.3.3 int sht

Speed of the shooter motors.

Definition at line 50 of file [opcontrol.c](#).

4.19.3.4 int spd

Forward/backward speed of the drive motors.

Definition at line 40 of file [opcontrol.c](#).

4.19.3.5 int trans

Speed of the transmission motors.

Definition at line 60 of file [opcontrol.c](#).

4.19.3.6 int turn

Turning speed of the drive motors.

Definition at line 45 of file [opcontrol.c](#).

4.20 opcontrol.h

```

00001
00008 #ifndef OPCONTROL_H
00009 #define OPCONTROL_H
00010
00014 extern int spd;
00015
00019 extern int turn;
00020
00024 extern int sht;
00025
00029 extern int intk;
00030
00034 extern int trans;
00035
00039 extern int dep;
00040
00044 void recordJoyInfo();
00045
00049 void moveRobot();
00050
00051 #endif
00052

```

4.21 include/sensors.h File Reference

File for important sensor declarations and functions.

Macros

- `#define LEFT_ENC_TOP 1`
- `#define LEFT_ENC_BOT 2`
- `#define RIGHT_ENC_TOP 3`
- `#define RIGHT_ENC_BOT 4`
- `#define TRANSMISSION_POT 2`
- `#define GEAR_DRIVE 1860`
- `#define GEAR_LIFT 4095`
- `#define POWER_EXPANDER_STATUS 3`
- `#define POWER_EXPANDER_VOLTAGE_DIVISOR 280`
- `#define NUM_BATTS 3`
- `#define BATT_MAIN 0`
- `#define BATT_BKUP 1`
- `#define BATT_PEXP 2`
- `#define GYRO_PORT 4`
- `#define GYRO_SENSITIVITY 0`
- `#define GYRO_NET_TARGET 0`

Functions

- unsigned int [powerLevelExpander](#) ()

Variables

- Encoder [leftenc](#)
- Encoder [rightenc](#)
- Gyro [gyro](#)

4.21.1 Detailed Description

File for important sensor declarations and functions.

This file contains the code for declarations and functions regarding sensors. The definitions contained herein define sensor ports. The functions contained herein process certain sensor values for later use.

Some functions defined herein are too complex to be defined as inline functions in the [sensors.h](#) file. Additionally, some sensors must be instantiated as object types. See the [sensors.c](#) file for these more object instantiations and function definitions

See also

[sensors.c](#)

Definition in file [sensors.h](#).

4.21.2 Macro Definition Documentation

4.21.2.1 `#define BATT_BKUP 1`

Battery ID number of the robot's backup battery.

Definition at line [89](#) of file [sensors.h](#).

4.21.2.2 `#define BATT_MAIN 0`

Battery ID number of the robot's main battery.

Definition at line [84](#) of file [sensors.h](#).

4.21.2.3 `#define BATT_PEXP 2`

Battery ID number of the power expander's battery.

Definition at line [94](#) of file [sensors.h](#).

4.21.2.4 `#define GEAR_DRIVE 1860`

Defines potentiometer values for drive gearing.

Definition at line 48 of file [sensors.h](#).

4.21.2.5 `#define GEAR_LIFT 4095`

Defines potentiometer values for lift gearing.

Definition at line 53 of file [sensors.h](#).

4.21.2.6 `#define GYRO_NET_TARGET 0`

Defines gyroscope target angle for the opposite corner net.

Definition at line 109 of file [sensors.h](#).

4.21.2.7 `#define GYRO_PORT 4`

Defines the port for the gyroscope.

Definition at line 99 of file [sensors.h](#).

4.21.2.8 `#define GYRO_SENSITIVITY 0`

Defines the gyroscope sensitivity. A value of zero represents the default sensitivity.

Definition at line 104 of file [sensors.h](#).

4.21.2.9 `#define LEFT_ENC_BOT 2`

Definition at line 22 of file [sensors.h](#).

4.21.2.10 `#define LEFT_ENC_TOP 1`

Defines the encoder ports on the left side of the drivetrain.

Definition at line 21 of file [sensors.h](#).

4.21.2.11 `#define NUM_BATTS 3`

The number of batteries present on the robot.

Definition at line 79 of file [sensors.h](#).

4.21.2.12 `#define POWER_EXPANDER_STATUS 3`

Defines power expander status port. This is used to get the power expander battery voltage.

Definition at line 59 of file [sensors.h](#).

4.21.2.13 `#define POWER_EXPANDER_VOLTAGE_DIVISOR 280`

Defines power expander divisor. The sensor's value is divided by this to get the battery voltage.

Definition at line 65 of file [sensors.h](#).

4.21.2.14 `#define RIGHT_ENC_BOT 4`

Definition at line 33 of file [sensors.h](#).

4.21.2.15 `#define RIGHT_ENC_TOP 3`

Defines the encoder ports on the right side of the drivetrain.

Definition at line 32 of file [sensors.h](#).

4.21.2.16 `#define TRANSMISSION_POT 2`

Defines the transmission potentiometer for position determination.

Definition at line 43 of file [sensors.h](#).

4.21.3 Function Documentation

4.21.3.1 `unsigned int powerLevelExpander () [inline]`

Returns the electric potential of the power expander battery in millivolts.

Returns

the power expander battery voltage, in millivolts

Definition at line 72 of file [sensors.h](#).

4.21.4 Variable Documentation

4.21.4.1 Gyro gyro

Object representing the gyroscope.

Definition at line 28 of file [sensors.c](#).

4.21.4.2 Encoder leftenc

Object representing the encoder on the left side of the drivetrain.

Definition at line 18 of file [sensors.c](#).

4.21.4.3 Encoder rightenc

Object representing the encoder on the right side of the drivetrain.

Definition at line 23 of file [sensors.c](#).

4.22 sensors.h

```

00001
00015 #ifndef SENSORS_H_
00016 #define SENSORS_H_
00017
00021 #define LEFT_ENC_TOP 1
00022 #define LEFT_ENC_BOT 2
00023
00027 extern Encoder leftenc;
00028
00032 #define RIGHT_ENC_TOP 3
00033 #define RIGHT_ENC_BOT 4
00034
00038 extern Encoder rightenc;
00039
00043 #define TRANSMISSION_POT 2
00044
00048 #define GEAR_DRIVE 1860
00049
00053 #define GEAR_LIFT 4095
00054
00059 #define POWER_EXPANDER_STATUS 3
00060
00065 #define POWER_EXPANDER_VOLTAGE_DIVISOR 280 //Hardware revision A2
00066
00072 inline unsigned int powerLevelExpander(){
00073     return analogRead(POWER_EXPANDER_STATUS)*1000/
00074         POWER_EXPANDER_VOLTAGE_DIVISOR;
00075 }
00079 #define NUM_BATTS 3
00080
00084 #define BATT_MAIN 0
00085
00089 #define BATT_BKUP 1
00090
00094 #define BATT_PEXP 2
00095
00099 #define GYRO_PORT 4
00100
00104 #define GYRO_SENSITIVITY 0
00105
00109 #define GYRO_NET_TARGET 0
00110
00114 extern Gyro gyro;
00115
00116 #endif

```

4.23 src/auto.c File Reference

File for autonomous code.

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

Functions

- void [autonomous](#) ()

4.23.1 Detailed Description

File for autonomous code.

This file should contain the user [autonomous\(\)](#) function and any functions related to it.

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

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

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

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

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

Definition in file [auto.c](#).

4.23.2 Function Documentation

4.23.2.1 void autonomous ()

C standard integer type header C standard string function header PROS main API header Mathematical constant definitions Useful redefinitions to make code more readable Simple macros for performing common operations Macros for performing bitwise operations Sensor definitions and function declarations Motor definitions and function declarations LCD definitions and function declarations LCD diagnostics menu definitions and function declarations Autonomous recorder definitions and function declarations Operator control definitions and function declarations 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 51 of file [auto.c](#).

4.24 auto.c

```

00001
00035 #include "main.h"
00036
00037 /*
00038  * Runs the user autonomous code. This function will be started in its own task with the default
00039  * priority and stack size whenever the robot is enabled via the Field Management System or the
00040  * VEX Competition Switch in the autonomous mode. If the robot is disabled or communications is
00041  * lost, the autonomous task will be stopped by the kernel. Re-enabling the robot will restart
00042  * the task, not re-start it from where it left off.
00043  *
00044  * Code running in the autonomous task cannot access information from the VEX Joystick. However,
00045  * the autonomous function can be invoked from another task if a VEX Competition Switch is not
00046  * available, and it can access joystick information if called in this way.
00047  *
00048  * The autonomous task may exit, unlike operatorControl() which should never exit. If it does
00049  * so, the robot will await a switch to another mode or disable/enable cycle.
00050  */
00051 void autonomous() {
00052     playbackAuton();
00053 }
```

4.25 src/autonrecorder.c File Reference

File for autonomous recorder code.

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

Functions

- int [selectAuton](#) ()
- void [initAutonRecorder](#) ()
- void [recordAuton](#) ()
- void [saveAuton](#) ()
- void [loadAuton](#) ()
- void [playbackAuton](#) ()

Variables

- [joyState](#) states [AUTON_TIME *JOY_POLL_FREQ]
- int [autonLoaded](#)
- int [progSkills](#)

4.25.1 Detailed Description

File for autonomous recorder code.

This file contains the code for the saving, loading, and playback of autonomous files. When an autonomous routine is recorded, it is saved to a file to flash memory. This file is loaded and executed during the autonomous period of the game. It works by saving the motor values at a point in time. At the corresponding point in time, the values are played back.

This file also handles the recording of programming skills by stitching 4 autonomous routines together.

Definition in file [autonrecorder.c](#).

4.25.2 Function Documentation

4.25.2.1 `void initAutonRecorder ()`

Initializes autonomous recorder by setting states array to zero.

Definition at line 72 of file [autonrecorder.c](#).

4.25.2.2 `void loadAuton ()`

Loads autonomous file contents into states array.

Definition at line 216 of file [autonrecorder.c](#).

4.25.2.3 `void playbackAuton ()`

Replays autonomous based on loaded values in states array.

Definition at line 309 of file [autonrecorder.c](#).

4.25.2.4 `void recordAuton ()`

Records driver joystick values into states array.

Definition at line 88 of file [autonrecorder.c](#).

4.25.2.5 `void saveAuton ()`

Saves contents of the states array to a file in flash memory.

Definition at line 133 of file [autonrecorder.c](#).

4.25.2.6 `int selectAuton ()`

Selects which autonomous file to use based on the potentiometer reading.

Returns

the autonomous selected (slot number)

Definition at line 36 of file [autonrecorder.c](#).

4.25.3 Variable Documentation

4.25.3.1 `int autonLoaded`

Slot number of currently loaded autonomous routine.

Definition at line 24 of file [autonrecorder.c](#).

4.25.3.2 int progSkills

Section number (0-3) of currently loaded programming skills routine.

Definition at line 29 of file [autonrecorder.c](#).

4.25.3.3 joyState states[AUTON_TIME*JOY_POLL_FREQ]

Stores the joystick state variables for moving the robot. Used for recording and playing back autonomous routines.

Definition at line 19 of file [autonrecorder.c](#).

4.26 autonrecorder.c

```

00001
00013 #include "main.h"
00014
00019 joyState states[AUTON_TIME*JOY_POLL_FREQ];
00020
00024 int autonLoaded;
00025
00029 int progSkills;
00030
00036 int selectAuton() {
00037     bool done = false;
00038     int val;
00039     do {
00040         val = (float) ((float) analogRead(AUTON_POT) / (float)
AUTON_POT_HIGH) * (MAX_AUTON_SLOTS+2);
00041         if(val > MAX_AUTON_SLOTS+1){
00042             val = MAX_AUTON_SLOTS+1;
00043         }
00044         if(val == 0) {
00045             lcdSetText(LCD_PORT, 2, "NONE");
00046         } else if(val == MAX_AUTON_SLOTS+1) {
00047             lcdSetText(LCD_PORT, 2, "Prog. Skills");
00048         } else {
00049             char filename[AUTON_FILENAME_MAX_LENGTH];
00050             snprintf(filename, sizeof(filename)/sizeof(char), "a%d", val);
00051             FILE* autonFile = fopen(filename, "r");
00052             if(autonFile == NULL){
00053                 lcdPrint(LCD_PORT, 2, "Slot: %d (EMPTY)", val);
00054             } else {
00055                 char name[LCD_MESSAGE_MAX_LENGTH+1];
00056                 memset(name, 0, sizeof(name));
00057                 fread(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
00058                 lcdSetText(LCD_PORT, 2, name);
00059                 fclose(autonFile);
00060             }
00061         }
00062         done = (digitalRead(AUTON_BUTTON) == PRESSED);
00063         delay(20);
00064     } while(!done);
00065     printf("Selected autonomous: %d\n", val);
00066     return val;
00067 }
00068
00072 void initAutonRecorder() {
00073     printf("Beginning initialization of autonomous recorder...\n");
00074     lcdClear(LCD_PORT);
00075     lcdSetText(LCD_PORT, 1, "Init recorder...");
00076     lcdSetText(LCD_PORT, 2, "");
00077     memset(states, 0, sizeof(*states));
00078     printf("Completed initialization of autonomous recorder.\n");
00079     lcdSetText(LCD_PORT, 1, "Init-ed recorder!");
00080     lcdSetText(LCD_PORT, 2, "");
00081     autonLoaded = -1;
00082     progSkills = 0;
00083 }
00084

```

```

00088 void recordAuton() {
00089     lcdClear(LCD_PORT);
00090     for(int i = 3; i > 0; i--){
00091         lcdSetBacklight(LCD_PORT, true);
00092         printf("Beginning autonomous recording in %d...\n", i);
00093         lcdSetText(LCD_PORT, 1, "Recording auton");
00094         lcdPrint(LCD_PORT, 2, "in %d...", i);
00095         delay(1000);
00096     }
00097     printf("Ready to begin autonomous recording.\n");
00098     lcdSetText(LCD_PORT, 1, "Recording auton...");
00099     lcdSetText(LCD_PORT, 2, "");
00100     bool lightState = false;
00101     for (int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00102         printf("Recording state %d...\n", i);
00103         lcdSetBacklight(LCD_PORT, lightState);
00104         lightState = !lightState;
00105         recordJoyInfo();
00106         states[i].spd = spd;
00107         states[i].turn = turn;
00108         states[i].sht = sht;
00109         states[i].intk = intk;
00110         states[i].trans = trans;
00111         states[i].dep = dep;
00112         if (joystickGetDigital(1, 7, JOY_UP)) {
00113             printf("Autonomous recording manually cancelled.\n");
00114             lcdSetText(LCD_PORT, 1, "Cancelled record.");
00115             lcdSetText(LCD_PORT, 2, "");
00116             memset(states + i + 1, 0, sizeof(joyState) * (AUTON_TIME * JOY_POLL_FREQ - i
- 1));
00117             i = AUTON_TIME * JOY_POLL_FREQ;
00118         }
00119         moveRobot();
00120         delay(1000 / JOY_POLL_FREQ);
00121     }
00122     printf("Completed autonomous recording.\n");
00123     lcdSetText(LCD_PORT, 1, "Recorded auton!");
00124     lcdSetText(LCD_PORT, 2, "");
00125     motorStopAll();
00126     delay(1000);
00127     autonLoaded = 0;
00128 }
00129
00133 void saveAuton() {
00134     printf("Waiting for file selection...\n");
00135     lcdClear(LCD_PORT);
00136     lcdSetText(LCD_PORT, 1, "Save to?");
00137     lcdSetText(LCD_PORT, 2, "");
00138     int autonSlot;
00139     if(progSkills == 0) {
00140         autonSlot = selectAuton();
00141     } else {
00142         printf("Currently in the middle of a programming skills run.\n");
00143         autonSlot = MAX_AUTON_SLOTS + 1;
00144     }
00145     char name[LCD_MESSAGE_MAX_LENGTH+1];
00146     memset(name, 0, sizeof(name));
00147     if(autonSlot == 0) {
00148         printf("Not saving this autonomous!\n");
00149         return;
00150     } else if(autonSlot != MAX_AUTON_SLOTS+1) {
00151         typeString(name);
00152     }
00153     lcdSetText(LCD_PORT, 1, "Saving auton...");
00154     char filename[AUTON_FILENAME_MAX_LENGTH];
00155     if(autonSlot != MAX_AUTON_SLOTS + 1) {
00156         printf("Not doing programming skills, recording to slot %d.\n",autonSlot);
00157         snprintf(filename, sizeof(filename)/sizeof(char), "a%d", autonSlot);
00158         //lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00159         lcdPrint(LCD_PORT, 2, "%s", name);
00160     } else {
00161         printf("Doing programming skills, recording to section %d.\n",
progSkills);
00162         snprintf(filename, sizeof(filename)/sizeof(char), "p%d", progSkills);
00163         lcdPrint(LCD_PORT, 2, "Skills Part: %d", progSkills+1);
00164     }
00165     printf("Saving to file %s...\n",filename);
00166     FILE *autonFile = fopen(filename, "w");
00167     if (autonFile == NULL) {
00168         printf("Error saving autonomous in file %s!\n", filename);
00169         lcdSetText(LCD_PORT, 1, "Error saving!");

```

```

00170         if(autonSlot != MAX_AUTON_SLOTS + 1){
00171             printf("Not doing programming skills, error saving auton in slot %d!\n", autonSlot);
00172             lcdSetText(LCD_PORT, 1, "Error saving!");
00173             lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00174         } else {
00175             printf("Doing programming skills, error saving auton in section 0!\n");
00176             lcdSetText(LCD_PORT, 1, "Error saving!");
00177             lcdSetText(LCD_PORT, 2, "Prog. Skills");
00178         }
00179         delay(1000);
00180         return;
00181     }
00182     if(autonSlot != MAX_AUTON_SLOTS+1){
00183         fwrite(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
00184     }
00185     for (int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00186         printf("Recording state %d to file %s...\n", i, filename);
00187         signed char write[6] = {states[i].spd, states[i].turn, states[i].
sht, states[i].intk, states[i].trans,
00188                                 states[i].dep};
00189         fwrite(write, sizeof(char), sizeof(write) / sizeof(char), autonFile);
00190         delay(10);
00191     }
00192     fclose(autonFile);
00193     printf("Completed saving autonomous to file %s.\n", filename);
00194     lcdSetText(LCD_PORT, 1, "Saved auton!");
00195     if(autonSlot != MAX_AUTON_SLOTS + 1) {
00196         printf("Not doing programming skills, recorded to slot %d.\n", autonSlot);
00197         lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00198     } else {
00199         printf("Doing programming skills, recorded to section %d.\n", progSkills);
00200         lcdPrint(LCD_PORT, 2, "Skills Part: %d", progSkills+1);
00201     }
00202     delay(1000);
00203     if(autonSlot == MAX_AUTON_SLOTS + 1) {
00204         printf("Proceeding to next programming skills section (%d).\n", ++
progSkills);
00205     }
00206     if(progSkills == PROGSKILL_TIME/AUTON_TIME) {
00207         printf("Finished recording programming skills (all parts).\n");
00208         progSkills = 0;
00209     }
00210     autonLoaded = autonSlot;
00211 }
00212
00216 void loadAuton() {
00217     lcdClear(LCD_PORT);
00218     bool done = false;
00219     int autonSlot;
00220     FILE* autonFile;
00221     char filename[AUTON_FILENAME_MAX_LENGTH];
00222     do {
00223         printf("Waiting for file selection...\n");
00224         lcdSetText(LCD_PORT, 1, "Load from?");
00225         lcdSetText(LCD_PORT, 2, "");
00226         autonSlot = selectAuton();
00227         if(autonSlot == 0) {
00228             printf("Not loading an autonomous!\n");
00229             lcdSetText(LCD_PORT, 1, "Not loading!");
00230             lcdSetText(LCD_PORT, 2, "");
00231             autonLoaded = 0;
00232             return;
00233         } else if(autonSlot == MAX_AUTON_SLOTS + 1){
00234             printf("Performing programming skills.\n");
00235             lcdSetText(LCD_PORT, 1, "Loading skills...");
00236             lcdPrint(LCD_PORT, 2, "Skills Part: 1");
00237             autonLoaded = MAX_AUTON_SLOTS + 1;
00238         } else if(autonSlot == autonLoaded) {
00239             printf("Autonomous %d is already loaded.\n", autonSlot);
00240             lcdSetText(LCD_PORT, 1, "Loaded auton!");
00241             lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00242             return;
00243         }
00244         printf("Loading autonomous from slot %d...\n", autonSlot);
00245         lcdSetText(LCD_PORT, 1, "Loading auton...");
00246         if(autonSlot != MAX_AUTON_SLOTS + 1){
00247             lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00248         }
00249         if(autonSlot != MAX_AUTON_SLOTS + 1){
00250             printf("Not doing programming skills, loading slot %d\n", autonSlot);
00251             snprintf(filename, sizeof(filename)/sizeof(char), "a%d", autonSlot);

```

```

00252     } else {
00253         printf("Doing programming skills, loading section 0.\n");
00254         snprintf(filename, sizeof(filename)/sizeof(char), "p0");
00255     }
00256     printf("Loading from file %s...\n", filename);
00257     autonFile = fopen(filename, "r");
00258     if (autonFile == NULL) {
00259         printf("No autonomous was saved in file %s!\n", filename);
00260         lcdSetText(LCD_PORT, 1, "No auton saved!");
00261         if(autonSlot != MAX_AUTON_SLOTS + 1){
00262             printf("Not doing programming skills, no auton in slot %d!\n", autonSlot);
00263             lcdSetText(LCD_PORT, 1, "No auton saved!");
00264             lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00265         } else {
00266             printf("Doing programming skills, no auton in section 0!\n");
00267             lcdSetText(LCD_PORT, 1, "No skills saved!");
00268         }
00269         delay(1000);
00270     } else {
00271         done = true;
00272     }
00273 } while(!done);
00274 fseek(autonFile, 0, SEEK_SET);
00275 char name[LCD_MESSAGE_MAX_LENGTH+1];
00276 memset(name, 0, sizeof(name));
00277 if(autonSlot != MAX_AUTON_SLOTS + 1){
00278     fread(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
00279 }
00280 for (int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00281     printf("Loading state %d from file %s...\n", i, filename);
00282     char read[6] = {0, 0, 0, 0, 0, 0};
00283     fread(read, sizeof(char), sizeof(read) / sizeof(char), autonFile);
00284     states[i].spd = (signed char) read[0];
00285     states[i].turn = (signed char) read[1];
00286     states[i].sht = (signed char) read[2];
00287     states[i].intk = (signed char) read[3];
00288     states[i].trans = (signed char) read[4];
00289     states[i].dep = (signed char) read[5];
00290     delay(10);
00291 }
00292 fclose(autonFile);
00293 printf("Completed loading autonomous from file %s.\n", filename);
00294 lcdSetText(LCD_PORT, 1, "Loaded auton!");
00295 if(autonSlot != MAX_AUTON_SLOTS + 1){
00296     printf("Not doing programming skills, loaded from slot %d.\n", autonSlot);
00297     //lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00298     lcdPrint(LCD_PORT, 2, "%s", name);
00299 } else {
00300     printf("Doing programming skills, loaded from section %d.\n", progSkills);
00301     lcdSetText(LCD_PORT, 2, "Skills Section: 1");
00302 }
00303 autonLoaded = autonSlot;
00304 }
00305
00309 void playbackAuton() { //must load autonomous first!
00310     if (autonLoaded == -1 /* nothing in memory */) {
00311         printf("No autonomous loaded, entering loadAuton()\n");
00312         loadAuton();
00313     }
00314     if(autonLoaded == 0) {
00315         printf("autonLoaded = 0, doing nothing.\n");
00316         return;
00317     }
00318     printf("Beginning playback...\n");
00319     lcdSetText(LCD_PORT, 1, "Playing back...");
00320     lcdSetText(LCD_PORT, 2, "");
00321     lcdSetBacklight(LCD_PORT, true);
00322     int file=0;
00323     do{
00324         FILE* nextFile = NULL;
00325         lcdPrint(LCD_PORT, 2, "File: %d", file+1);
00326         char filename[AUTON_FILENAME_MAX_LENGTH];
00327         if(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
PROGSKILL_TIME/AUTON_TIME - 1){
00328             printf("Next section: %d\n", file+1);
00329             snprintf(filename, sizeof(filename)/sizeof(char), "p%d", file+1);
00330             nextFile = fopen(filename, "r");
00331         }
00332         for(int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00333             printf("Playing back state %d...\n", i);
00334             spd = states[i].spd;

```

```

00335         turn = states[i].turn;
00336         sht = states[i].sht;
00337         intk = states[i].intk;
00338         trans = states[i].trans;
00339         dep = states[i].dep;
00340         if (joystickGetDigital(1, 7, JOY_UP) && !isOnline()) {
00341             printf("Playback manually cancelled.\n");
00342             lcdSetText(LCD_PORT, 1, "Cancelled playback.");
00343             lcdSetText(LCD_PORT, 2, "");
00344             i = AUTON_TIME * JOY_POLL_FREQ;
00345             file = PROGSKILL_TIME/AUTON_TIME;
00346         }
00347         moveRobot();
00348         if(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
PROGSKILL_TIME/AUTON_TIME - 1){
00349             printf("Loading state %d from file %s...\n", i, filename);
00350             char read[6] = {0, 0, 0, 0, 0, 0};
00351             fread(read, sizeof(char), sizeof(read) / sizeof(char), nextFile);
00352             states[i].spd = (signed char) read[0];
00353             states[i].turn = (signed char) read[1];
00354             states[i].sht = (signed char) read[2];
00355             states[i].intk = (signed char) read[3];
00356             states[i].trans = (signed char) read[4];
00357             states[i].dep = (signed char) read[5];
00358         }
00359         delay(1000 / JOY_POLL_FREQ);
00360     }
00361     if(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
PROGSKILL_TIME/AUTON_TIME - 1){
00362         printf("Finished with section %d, closing file.\n", file+1);
00363         fclose(nextFile);
00364     }
00365     file++;
00366 } while(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
PROGSKILL_TIME/AUTON_TIME);
00367 motorStopAll();
00368 printf("Completed playback.\n");
00369 lcdSetText(LCD_PORT, 1, "Played back!");
00370 lcdSetText(LCD_PORT, 2, "");
00371 delay(1000);
00372 }

```

4.27 src/init.c File Reference

File for initialization code.

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

Functions

- void [initializeIO](#) ()
- void [initialize](#) ()

4.27.1 Detailed Description

File for initialization code.

This file should contain the user [initialize\(\)](#) function and any functions related to it.

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

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

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

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

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

Definition in file [init.c](#).

4.27.2 Function Documentation

4.27.2.1 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 61 of file [init.c](#).

4.27.2.2 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 45 of file [init.c](#).

4.28 init.c

```

00001
00035 #include "main.h"
00036
00037 /*
00038  * Runs pre-initialization code. This function will be started in kernel mode one time while the
00039  * VEX Cortex is starting up. As the scheduler is still paused, most API functions will fail.
00040  *
00041  * The purpose of this function is solely to set the default pin modes (pinMode()) and port
00042  * states (digitalWrite()) of limit switches, push buttons, and solenoids. It can also safely
00043  * configure a UART port (usartOpen()) but cannot set up an LCD (lcdInit()).
00044  */
00045 void initializeIO() {
00046 }
00047
00048 /*
00049  * Runs user initialization code. This function will be started in its own task with the default
00050  * priority and stack size once when the robot is starting up. It is possible that the VEXnet
00051  * communication link may not be fully established at this time, so reading from the VEX
00052  * Joystick may fail.
00053  *
00054  * This function should initialize most sensors (gyro, encoders, ultrasonics), LCDs, global
00055  * variables, and IMEs.
00056  *
00057  * This function must exit relatively promptly, or the operatorControl() and autonomous() tasks
00058  * will not start. An autonomous mode selection menu like the pre_auton() in other environments
00059  * can be implemented in this task if desired.
00060  */
00061 void initialize() {
00062     int seed = powerLevelMain() + powerLevelBackup();
00063     for(int i = 0; i < BOARD_NR_ADC_PINS; i++) {
00064         seed += analogRead(i);
00065     }
00066     srand(seed);
00067     leftenc = encoderInit(LEFT_ENC_TOP, LEFT_ENC_BOT, false);
00068     rightenc = encoderInit(RIGHT_ENC_TOP, RIGHT_ENC_BOT, false);
00069     lcdInit(LCD_PORT);
00070     lcdClear(LCD_PORT);
00071     lcdSetBacklight(LCD_PORT, true);
00072     lcdSetText(LCD_PORT, 1, "Init-ing gyro...");
00073     gyro = gyroInit(GYRO_PORT, GYRO_SENSITIVITY);
00074     delay(1100);
00075     gyroReset(gyro);
00076     lcdSetText(LCD_PORT, 1, "Init-ed gyro!");
00077     initAutonRecorder();
00078     initGroups();
00079     if(isOnline()){
00080         loadAuton();
00081     }
00082 }

```

4.29 src/lcddiag.c File Reference

File for LCD diagnostic menu code.

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

Functions

- char * [typeString](#) (char *dest)
- void [saveGroups](#) ()
- void [loadGroups](#) ()
- void [initGroups](#) ()
- void [formatMenuNameCenter](#) (FILE *lcdport, int line, int index)

- int [selectMenu](#) ()
- void [runScreensaver](#) (FILE *lcdport)
- void [runBattery](#) (FILE *lcdport)
- int [selectMotor](#) (FILE *lcdport)
- int [selectSpd](#) (int mtr)
- void [runIndivMotor](#) (FILE *lcdport)
- int [selectMotorGroup](#) (FILE *lcdport)
- int [selectSpdGroup](#) (int mtr)
- void [runGroupMotor](#) (FILE *lcdport)
- void [runMotor](#) (FILE *lcdport)
- bool * [selectMotorGroupMembers](#) (bool *motor)
- void [addMotorGroup](#) ()
- void [editMotorGroup](#) (int mtr)
- void [delMotorGroup](#) (int mtr)
- void [runMotorGroupMgmt](#) (FILE *lcdport)
- void [runConnection](#) (FILE *lcdport)
- void [runRobot](#) (FILE *lcdport)
- void [runAuton](#) (FILE *lcdport)
- void [runCredits](#) (FILE *lcdport)
- void [doMenuChoice](#) (int choice)
- void [formatLCDDisplay](#) (void *ignore)

Variables

- TaskHandle [lcdDiagTask](#) = NULL
- bool [backlight](#) = true
- bool [disableOpControl](#) = false
- [MotorGroup](#) * [groups](#)
- int [numgroups](#)
- char [menuChoices](#) [[LCD_MENU_COUNT](#)][[LCD_MESSAGE_MAX_LENGTH](#)+1]

4.29.1 Detailed Description

File for LCD diagnostic menu code.

This file contains the code for the LCD diagnostic menu. The menu provides live debugging and testing functionality. It provides the following functions:

- Motor testing functionality (individual and group)
- Motor group management
- Battery voltage information
- Joystick connection status
- Robot sensory data
- Autonomous recorder status
- LCD backlight toggle

- Screensaver that displays during operator control
- Credits menu

The idea behind this was inspired by Team 750W and Akram Sandhu. Without them, this project would not be possible.

Note: the implementation of this feature is completely different between the two teams. No code was reused from their implementation of the LCD diagnostic menu.

Definition in file [lcddiag.c](#).

4.29.2 Function Documentation

4.29.2.1 void addMotorGroup ()

Adds a new motor group to the dynamic array.

Definition at line 901 of file [lcddiag.c](#).

4.29.2.2 void delMotorGroup (int *mtr*)

Deletes a motor group. Prompts the user whether to cancel or to proceed with deletion.

Parameters

<i>mtr</i>	the ID number to delete
------------	-------------------------

Definition at line 972 of file [lcddiag.c](#).

4.29.2.3 void doMenuChoice (int *choice*)

Dispatcher function that executes the selected LCD diagnostic menu function. This function is called with the result of [selectMenu\(\)](#).

See also

[selectMenu\(\)](#)

Parameters

<i>choice</i>	the selected menu to run
---------------	--------------------------

Definition at line 1278 of file [lcddiag.c](#).

4.29.2.4 void editMotorGroup (int *mtr*)

Edits a motor group. Prompts the user to either edit the name or the motors in a motor group.

Parameters

<i>mtr</i>	the ID number of the motor group to edit
------------	--

Definition at line 917 of file [lcddiag.c](#).

4.29.2.5 void formatLCDDisplay (void * *ignore*)

Runs the LCD diagnostic menu task. This thread executes concurrently with the operator control task. The LCD diagnostic menu starts in screensaver mode. Pressing any button cancels screensaver mode and enters the selection menu.

Parameters

<i>ignore</i>	does nothing - required by task definition
---------------	--

Definition at line 1300 of file [lcddiag.c](#).

4.29.2.6 void formatMenuNameCenter (FILE * *lcdport*, int *line*, int *index*)

Formats the LCD diagnostic menu name in the center of the screen.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
<i>line</i>	the line on the LCD to print the name
<i>index</i>	the ID number of the menu

Definition at line 284 of file [lcddiag.c](#).

4.29.2.7 void initGroups ()

Initializes the motor groups array to contain the standard set of groups. This includes: Left Drive, Right Drive, Full Drive, Nautilus Shooter, Intake, and Transmission.

Definition at line 230 of file [lcddiag.c](#).

4.29.2.8 void loadGroups ()

Loads motor groups from a file on the Cortex flash memory. This is used to add custom motor groups for testing purposes.

Definition at line 198 of file [lcddiag.c](#).

4.29.2.9 void runAuton (FILE * *lcdport*)

Runs the autonomous recorder status menu. Displays the autonomous that is currently loaded, and if controller playback is enabled. Controller playback is automatically disabled when plugged into the competition switch.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 1170 of file [lcddiag.c](#).

4.29.2.10 void runBattery (FILE * *lcdport*)

Displays the battery voltages, allowing for switching between batteries.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 356 of file [lcddiag.c](#).

4.29.2.11 void runConnection (FILE * *lcdport*)

Runs the joystick connection debugging menu. Prints whether the main and partner joysticks are connected.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 1070 of file [lcddiag.c](#).

4.29.2.12 void runCredits (FILE * *lcdport*)

Runs the credits menu. The LCD diagnostic menu was inspired by Team 750W and Akram Sandhu. This would not be possible without their generosity and permissiveness to use their idea.

Note: the implementation of this feature is completely different between the two teams. No code was reused from their implementation of the LCD diagnostic menu.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 1235 of file [lcddiag.c](#).

4.29.2.13 void runGroupMotor (FILE * *lcdport*)

Runs the motor group test. Selection of the motor group and speed is handled by other functions.

See also

[selectMotorGroup\(\)](#)
[selectSpdGroup\(\)](#)

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 729 of file [lcddiag.c](#).

4.29.2.14 void runIndivMotor (FILE * *lcdport*)

Runs the individual motor test. Selection of the motor and speed is handled by other functions

See also

[selectMotor\(\)](#)
[selectSpd\(\)](#)

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 557 of file [lcddiag.c](#).

4.29.2.15 void runMotor (FILE * *lcdport*)

Runs the top-level motor testing menu. Prompts the user to select between individual and group motor testing.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 793 of file [lcddiag.c](#).

4.29.2.16 void runMotorGroupMgmt (FILE * *lcdport*)

Runs the top-level motor group management menu. Prompts the user whether to add, edit, or delete motor groups.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 1029 of file [lcddiag.c](#).

4.29.2.17 void runRobot (FILE * *lcdport*)

Runs the robot sensory information menu. Displays information regarding competition switch status and gyroscope angle.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 1119 of file [lcddiag.c](#).

4.29.2.18 void runScreensaver (FILE * *lcdport*)

Runs the screensaver that displays LCD messages.

See also

[lcdmsg.c](#)

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Definition at line 337 of file [lcddiag.c](#).

4.29.2.19 void saveGroups ()

Saves motor groups out to a file. This file can be loaded to add custom motor groups into memory.

Definition at line 173 of file [lcddiag.c](#).

4.29.2.20 int selectMenu ()

Displays the list of menus and waits for the user to select one.

Returns

the ID number of the menu selected

Definition at line 302 of file [lcddiag.c](#).

4.29.2.21 int selectMotor (FILE * *lcdport*)

Prompts the user to select an individual motor to test.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Returns

the motor number selected

Definition at line 424 of file [lcddiag.c](#).

4.29.2.22 int selectMotorGroup (FILE * *lcdport*)

Selects the motor group to test.

Parameters

<i>lcdport</i>	the LCD screen's port (either UART1 or UART2)
----------------	---

Returns

the ID number of the group to test

Definition at line 620 of file [lcddiag.c](#).

4.29.2.23 bool* selectMotorGroupMembers (bool * *motor*)

Selects the motors that constitute the specified motor group.

Parameters

<i>motor</i>	a boolean array that stores the states of each motor in the group
--------------	---

Returns

a pointer to the array passed

Definition at line 831 of file [lcddiag.c](#).

4.29.2.24 int selectSpd (int *mtr*)

Selects the speed of the motor to test.

Parameters

<i>mtr</i>	the motor number that is being tested
------------	---------------------------------------

Returns

the speed selected

Definition at line 492 of file [lcddiag.c](#).

4.29.2.25 int selectSpdGroup (int *mtr*)

Selects the speed of the motor group to be tested.

Parameters

<i>mtr</i>	the ID number of the motor group to be tested
------------	---

Returns

the speed selected

Definition at line 667 of file [lcddiag.c](#).

4.29.2.26 char* typeString (char * *dest*)

Uses the LCD and the autonomous potentiometer to type a string. This is used to name motor groups and autonomous recordings. The maximum length of string this function can type is 16 characters.

Parameters

<i>dest</i>	a buffer to store the typed string (must be at least 17 characters to hold null terminator)
-------------	---

Returns

a pointer to the buffer

Definition at line 81 of file [lcddiag.c](#).

4.29.3 Variable Documentation**4.29.3.1 bool backlight = true**

Boolean representing the LCD screen's backlight state.

Definition at line 36 of file [lcddiag.c](#).

4.29.3.2 bool disableOpControl = false

Disables operator control loop during motor testing. Since running motors is not thread safe, it is necessary to stop operator control of the motors during testing.

Definition at line 42 of file [lcddiag.c](#).

4.29.3.3 MotorGroup* groups

Array that stores the motor groups. As this is a dynamic array, creating and editing new motor groups is possible. These motor groups are added to the array via the Motor Group Management menu.

Definition at line 49 of file [lcddiag.c](#).

4.29.3.4 TaskHandle lcdDiagTask = NULL

Object representing the LCD diagnostic menu task. The LCD diagnostic menu runs in a separate thread from the operator control code. The TaskHandle allows for pausing and resuming of the LCD diagnostic menu during autonomous recording.

Definition at line 31 of file [lcddiag.c](#).

4.29.3.5 char menuChoices[LCD_MENU_COUNT][LCD_MESSAGE_MAX_LENGTH+1]

Initial value:

```
= {  
    "Motor Test",  
    "Motor Group Mgmt",  
    "Battery Info",  
    "Connection Info",  
    "Robot Info",  
    "Autonomous Info",  
    "Toggle Backlight",  
    "Screensaver",  
    "Credits"  
}
```

Stores the top-level menu names.

Definition at line 60 of file [lcddiag.c](#).

4.29.3.6 int numgroups

Stores the number of motor groups. This is functionally identical to the size of the motor group array.

Definition at line 55 of file [lcddiag.c](#).

4.30 lcddiag.c

```

00001
00024 #include "main.h"
00025
00031 TaskHandle lcdDiagTask = NULL;
00032
00036 bool backlight = true;
00037
00042 bool disableOpControl = false;
00043
00049 MotorGroup *groups;
00050
00055 int numgroups;
00056
00060 char menuChoices[LCD_MENU_COUNT][LCD_MESSAGE_MAX_LENGTH+1] =
    {
00061     "Motor Test",
00062     "Motor Group Mgmt",
00063     "Battery Info",
00064     "Connection Info",
00065     "Robot Info",
00066     "Autonomous Info",
00067     "Toggle Backlight",
00068     "Screensaver",
00069     "Credits"
00070 };
00071
00081 char* typeString(char *dest){
00082     bool done = false;
00083     int val;
00084     memset(dest, 0, sizeof(*dest));
00085     int i = 0;
00086     typedef enum Case {
00087         UPPER, LOWER, NUMBER
00088     } Case;
00089     Case c = UPPER;
00090     int mult = 28;
00091     int spacecode = 26;
00092     int endcode = 27;
00093     do {
00094         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00095         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00096         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00097
00098         switch(c){
00099             case UPPER:
00100                 case LOWER: mult = 28; spacecode = 26; endcode = 27; break;
00101                 case NUMBER: mult = 12; spacecode = 10; endcode = 11; break;
00102             }
00103         val = (float) ((float) analogRead(AUTON_POT) / (float)
AUTON_POT_HIGH) * mult;
00104         if(val > endcode){
00105             val = endcode;
00106         }
00107         if(val == spacecode){
00108             dest[i] = ' ';
00109         } else if(val == endcode) {
00110             dest[i] = '~';
00111         } else {
00112             switch(c){
00113                 case UPPER: dest[i] = val + 'A'; break;
00114                 case LOWER: dest[i] = val + 'a'; break;
00115                 case NUMBER: dest[i] = val + '0'; break;
00116             }
00117         }
00118
00119         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(dest))/2;
00120         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00121         for(int i = 0; i < spaces; i++){
00122             strcat(str, " ");
00123         }
00124         strcat(str, dest);
00125         for(int i = 0; i < spaces; i++){
00126             strcat(str, " ");
00127         }
00128
00129         lcdSetText(LCD_PORT, 1, str);
00130
00131         if(i > 0){

```

```

00132         if(c == UPPER){
00133             lcdSetText(LCD_PORT, 2, "DEL    SEL    abc");
00134         } else if(c == LOWER) {
00135             lcdSetText(LCD_PORT, 2, "DEL    SEL    123");
00136         } else { //NUMBER
00137             lcdSetText(LCD_PORT, 2, "DEL    SEL    ABC");
00138         }
00139     } else {
00140         if(c == UPPER){
00141             lcdSetText(LCD_PORT, 2, "|    SEL    abc");
00142         } else if(c == LOWER) {
00143             lcdSetText(LCD_PORT, 2, "|    SEL    123");
00144         } else { //NUMBER
00145             lcdSetText(LCD_PORT, 2, "|    SEL    ABC");
00146         }
00147     }
00148
00149     if((centerPressed) && val != endcode){
00150         i++;
00151     } else if(leftPressed && i > 0){
00152         dest[i] = 0;
00153         i--;
00154     } else if(rightPressed) {
00155         switch(c){
00156             case UPPER: c = LOWER; break;
00157             case LOWER: c = NUMBER; break;
00158             case NUMBER: c = UPPER; break;
00159         }
00160     }
00161
00162     done = ((centerPressed && val == endcode) || i == LCD_MESSAGE_MAX_LENGTH);
00163     delay(20);
00164 } while(!done);
00165 dest[i] = 0;
00166 return dest;
00167 }
00168
00173 void saveGroups(){
00174     FILE* group = fopen("grp", "w");
00175     taskPrioritySet(NULL, TASK_PRIORITY_HIGHEST-1);
00176     lcdSetText(LCD_PORT, 1, "Saving groups...");
00177     lcdSetText(LCD_PORT, 2, "");
00178     for(int i = 0; i < numgroups; i++){
00179         fwrite(groups[i].motor, sizeof(bool), sizeof(groups[i].motor) / sizeof(bool), group);
00180         fwrite("\n", sizeof(char), sizeof("\n") / sizeof(char), group);
00181         fwrite(groups[i].name, sizeof(char), sizeof(groups[i].name) / sizeof(char), group);
00182         if(i == numgroups-1){
00183             fwrite("\t", sizeof(char), sizeof("\t") / sizeof(char), group);
00184         } else {
00185             fwrite("\n", sizeof(char), sizeof("\n") / sizeof(char), group);
00186         }
00187     }
00188     taskPrioritySet(NULL, TASK_PRIORITY_DEFAULT);
00189     lcdSetText(LCD_PORT, 1, "Saved groups!");
00190     lcdSetText(LCD_PORT, 2, "");
00191     delay(1000);
00192 }
00193
00198 void loadGroups(){
00199     FILE* group = fopen("grp", "r");
00200     taskPrioritySet(NULL, TASK_PRIORITY_HIGHEST-1);
00201     lcdSetText(LCD_PORT, 1, "Loading groups...");
00202     lcdSetText(LCD_PORT, 2, "");
00203     if(groups != NULL){
00204         free(groups);
00205     }
00206     groups = NULL;
00207     int i = 0;
00208     bool done = false;
00209     while(!done){
00210         groups = (MotorGroup *) realloc(groups, sizeof(MotorGroup)*(i+1));
00211         fread(groups[i].motor, sizeof(bool), sizeof(groups[i].motor) / sizeof(bool), group);
00212         fgetc(group);
00213         fread(groups[i].name, sizeof(char), sizeof(groups[i].name) / sizeof(char), group);
00214         if(fgetc(group) == '\t'){
00215             done = true;
00216         }
00217         i++;
00218     }
00219     numgroups = i;
00220     taskPrioritySet(NULL, TASK_PRIORITY_DEFAULT);

```

```

00221     lcdSetText(LCD_PORT, 1, "Loaded groups!");
00222     lcdSetText(LCD_PORT, 2, "");
00223     delay(1000);
00224 }
00225
00230 void initGroups(){
00231     FILE* group = fopen("grp", "r");
00232     if(group == NULL){
00233         numgroups = 6; //LDRIVE, RDRIVE, DRIVE, SHOOT, INTK, TRANS
00234         groups = (MotorGroup*) malloc(sizeof(MotorGroup) *
numgroups);
00235         if(groups == NULL){
00236             return;
00237         }
00238         memset(groups, 0, sizeof(*groups));
00239         for(int i = 0; i<numgroups; i++){
00240             for(int j = 0; j<=10; j++){
00241                 groups[i].motor[j] = false;
00242             }
00243         }
00244         groups[0].motor[LEFT_MOTOR_TOP] = true;
00245         groups[0].motor[LEFT_MOTOR_BOT] = true;
00246         strcpy(groups[0].name, "Left Drive");
00247
00248         groups[1].motor[RIGHT_MOTOR_TOP] = true;
00249         groups[1].motor[RIGHT_MOTOR_BOT] = true;
00250         strcpy(groups[1].name, "Right Drive");
00251
00252         groups[2].motor[LEFT_MOTOR_TOP] = true;
00253         groups[2].motor[LEFT_MOTOR_BOT] = true;
00254         groups[2].motor[RIGHT_MOTOR_TOP] = true;
00255         groups[2].motor[RIGHT_MOTOR_BOT] = true;
00256         strcpy(groups[2].name, "Full Drive");
00257
00258         groups[3].motor[NAUTILUS_SHOOTER_MOTOR_LEFT] = true;
00259         groups[3].motor[NAUTILUS_SHOOTER_MOTOR_RIGHT] = true;
00260 #ifdef SHOOTER_HAS_THREE_MOTORS
00261         groups[3].motor[NAUTILUS_SHOOTER_MOTOR_CENTER] = true;
00262 #endif
00263         strcpy(groups[3].name, "Nautilus Shooter");
00264
00265         groups[4].motor[INTAKE_ROLLER_MOTOR] = true;
00266         strcpy(groups[4].name, "Intake");
00267
00268         groups[5].motor[TRANSMISSION_MOTOR] = true;
00269         strcpy(groups[5].name, "Transmission");
00270
00271         //saveGroups();
00272     } else {
00273         //loadGroups();
00274     }
00275 }
00276
00284 void formatMenuNameCenter(FILE* lcdport, int line, int index){
00285     int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(
menuChoices[index]))/2;
00286     char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00287     for(int i = 0; i < spaces; i++){
00288         strcat(str, " ");
00289     }
00290     strcat(str, menuChoices[index]);
00291     for(int i = 0; i < spaces; i++){
00292         strcat(str, " ");
00293     }
00294     lcdSetText(lcdport, line, str);
00295 }
00296
00302 int selectMenu() {
00303     bool done = false;
00304     int val = 0;
00305     do {
00306         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00307         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00308         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00309
00310         if(rightPressed && val != LCD_MENU_COUNT-1) val++;
00311         else if(rightPressed && val == LCD_MENU_COUNT-1) val = 0;
00312         else if(leftPressed && val != 0) val--;
00313         else if(leftPressed && val == 0) val = LCD_MENU_COUNT - 1;
00314
00315         formatMenuNameCenter(LCD_PORT, 1, val);

```

```

00316         if(val == 0){
00317             lcdSetText(LCD_PORT, 2, "<      SEL      >");
00318         } else if(val == LCD_MENU_COUNT-1) {
00319             lcdSetText(LCD_PORT, 2, "<      SEL      >");
00320         } else {
00321             lcdSetText(LCD_PORT, 2, "<      SEL      >");
00322         }
00323         delay(20);
00324         done = centerPressed;
00325     } while(!done);
00326     printf("Selected menu choice: %d\n", val);
00327     return val;
00328 }
00329
00337 void runScreensaver(FILE *lcdport){
00338     int cycle = 0;
00339     do {
00340         if(cycle == 0){
00341             screensaver(LCD_PORT);
00342         }
00343         delay(20);
00344         cycle++;
00345         if(cycle==150){
00346             cycle=0;
00347         }
00348     } while(!lcdAnyButtonPressed());
00349 }
00350
00356 void runBattery(FILE *lcdport){
00357     bool done = false;
00358     int val = BATT_MAIN;
00359     do {
00360         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00361         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00362         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00363
00364         if(rightPressed && val != BATT_PEXP) val++;
00365         else if(rightPressed && val == BATT_PEXP) val = BATT_MAIN;
00366         else if(leftPressed && val != BATT_MAIN) val--;
00367         else if(leftPressed && val == BATT_MAIN) val = BATT_PEXP;
00368
00369         int beforepoint = 0;
00370         int afterpoint = 0;
00371         char battdisp[LCD_MESSAGE_MAX_LENGTH+1];
00372         char temp[LCD_MESSAGE_MAX_LENGTH+1];
00373         memset(battdisp, 0, sizeof(battdisp));
00374         memset(temp, 0, sizeof(temp));
00375
00376         switch(val){
00377             case BATT_MAIN: beforepoint = powerLevelMain()/1000;
00378                             afterpoint = powerLevelMain()%1000;
00379                             strcat(battdisp, "Mn Batt: ");
00380                             break;
00381             case BATT_BKUP: beforepoint = powerLevelBackup()/1000;
00382                             afterpoint = powerLevelBackup()%1000;
00383                             strcat(battdisp, "Bk Batt: ");
00384                             break;
00385             case BATT_PEXP: beforepoint = powerLevelExpander()/1000;
00386                             afterpoint = powerLevelExpander()%1000;
00387                             strcat(battdisp, "Ex Batt: ");
00388                             break;
00389         }
00390         sprintf(temp, "%d.%d V", beforepoint, afterpoint);
00391         strcat(battdisp, temp);
00392
00393         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(battdisp))/2;
00394         char str[LCD_MESSAGE_MAX_LENGTH+1];
00395         memset(str, 0, sizeof(str));
00396         for(int i = 0; i < spaces; i++){
00397             strcat(str, " ");
00398         }
00399         strcat(str, battdisp);
00400         for(int i = 0; i < spaces; i++){
00401             strcat(str, " ");
00402         }
00403         lcdSetText(LCD_PORT, 1, str);
00404
00405         if(val == BATT_MAIN){
00406             lcdSetText(LCD_PORT, 2, "EX      ESC      BK");
00407         } else if(val == BATT_PEXP) {
00408             lcdSetText(LCD_PORT, 2, "BK      ESC      MN");

```

```

00409         } else { //BATT_BKUP
00410             lcdSetText(LCD_PORT, 2, "MN      ESC      EX");
00411         }
00412         delay(20);
00413         done = centerPressed;
00414     } while(!done);
00415 }
00416
00424 int selectMotor(FILE *lcdport){
00425     bool done = false;
00426     int val = 1;
00427     do {
00428         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00429         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00430         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00431
00432         if(rightPressed && val != 10) val++;
00433         else if(rightPressed && val == 10) val = 0;
00434         else if(leftPressed && val != 0) val--;
00435         else if(leftPressed && val == 0) val = 10;
00436
00437         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00438         if(val != 0){
00439             char temp[LCD_MESSAGE_MAX_LENGTH+1];
00440             memset(motorstr, 0, sizeof(motorstr));
00441             memset(temp, 0, sizeof(temp));
00442             strcpy(motorstr, "Motor: ");
00443             sprintf(temp, "%d", val);
00444             strcat(motorstr, temp);
00445         } else {
00446             strcpy(motorstr, "Cancel");
00447         }
00448
00449         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00450         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00451         for(int i = 0; i < spaces; i++){
00452             strcat(str, " ");
00453         }
00454         strcat(str, motorstr);
00455         for(int i = 0; i < spaces; i++){
00456             strcat(str, " ");
00457         }
00458
00459         lcdSetText(LCD_PORT, 1, str);
00460
00461         done = centerPressed;
00462         if(val == 0){
00463             lcdSetText(LCD_PORT, 2, "10      SEL      1");
00464         } else if(val == 1) {
00465             lcdSetText(LCD_PORT, 2, "ESC      SEL      2");
00466         } else if(val == 10) {
00467             lcdSetText(LCD_PORT, 2, "9       SEL      ESC");
00468         } else if (val == 9) {
00469             lcdSetText(LCD_PORT, 2, "8       SEL      10");
00470         } else {
00471             char navstr[LCD_MESSAGE_MAX_LENGTH+1];
00472             memset(navstr, 0, sizeof(navstr));
00473             sprintf(navstr, "%c      SEL      %c", (val-1) + '0', (val+1) + '0');
00474             /*navstr[0] = (val-1) + '0';*/
00475             /*strcat(navstr, "      SEL      ");*/
00476             /*navstr[LCD_MESSAGE_MAX_LENGTH] = (val+1) + '0';*/
00477             lcdSetText(LCD_PORT, 2, navstr);
00478         }
00479         delay(20);
00480     } while(!done);
00481     printf("Testing motor %d.\n", val);
00482     return val;
00483 }
00484
00492 int selectSpd(int mtr) {
00493     bool done = false;
00494     int val=0;
00495     do {
00496         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00497         /*bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);*/
00498         /*bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);*/
00499
00500         val = (float) ((float) analogRead(AUTON_POT)/(float)
AUTON_POT_HIGH) * 254;
00501         val -= 127;
00502         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];

```

```

00503     char temp[LCD_MESSAGE_MAX_LENGTH+1];
00504     memset(motorstr, 0, sizeof(motorstr));
00505     memset(temp, 0, sizeof(temp));
00506     strcpy(motorstr, "Motor: ");
00507     sprintf(temp, "%d", mtr);
00508     strcat(motorstr, temp);
00509
00510     int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00511     char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00512     for(int i = 0; i < spaces; i++){
00513         strcat(str, " ");
00514     }
00515     strcat(str, motorstr);
00516     for(int i = 0; i < spaces; i++){
00517         strcat(str, " ");
00518     }
00519
00520     lcdSetText(LCD_PORT, 1, str);
00521
00522     char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00523     memset(speedstr, 0, sizeof(speedstr));
00524     memset(temp, 0, sizeof(temp));
00525     strcpy(speedstr, "Speed: ");
00526     sprintf(temp, "%d", val);
00527     strcat(speedstr, temp);
00528
00529     spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00530     strcpy(str, "");
00531     for(int i = 0; i < spaces; i++){
00532         strcat(str, " ");
00533     }
00534     strcat(str, speedstr);
00535     for(int i = 0; i < spaces; i++){
00536         strcat(str, " ");
00537     }
00538
00539     lcdSetText(LCD_PORT, 2, str);
00540
00541     done = (digitalRead(AUTON_BUTTON) == PRESSED || centerPressed);
00542     delay(20);
00543 } while(!done);
00544 printf("Using speed: %d\n", val);
00545 return val;
00546 }
00547
00557 void runIndivMotor(FILE *lcdport){
00558     bool done = false;
00559     int mtr = selectMotor(lcdport);
00560     if(mtr == 0) return;
00561     int spd = selectSpd(mtr);
00562     int val = spd;
00563     disableOpControl = true;
00564     motorStopAll();
00565     do {
00566         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00567         char temp[LCD_MESSAGE_MAX_LENGTH+1];
00568         memset(motorstr, 0, sizeof(motorstr));
00569         memset(temp, 0, sizeof(temp));
00570         strcpy(motorstr, "Motor: ");
00571         sprintf(temp, "%d", mtr);
00572         strcat(motorstr, temp);
00573
00574         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00575         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00576         for(int i = 0; i < spaces; i++){
00577             strcat(str, " ");
00578         }
00579         strcat(str, motorstr);
00580         for(int i = 0; i < spaces; i++){
00581             strcat(str, " ");
00582         }
00583
00584         lcdSetText(LCD_PORT, 1, str);
00585
00586         char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00587         memset(speedstr, 0, sizeof(speedstr));
00588         memset(temp, 0, sizeof(temp));
00589         strcpy(speedstr, "Run Speed: ");
00590         sprintf(temp, "%d", val);
00591         strcat(speedstr, temp);
00592

```

```

00593     spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00594     strcpy(str, "");
00595     for(int i = 0; i < spaces; i++){
00596         strcat(str, " ");
00597     }
00598     strcat(str, speedstr);
00599     for(int i = 0; i < spaces; i++){
00600         strcat(str, " ");
00601     }
00602     lcdSetText(LCD_PORT, 2, str);
00603
00604     done = lcdAnyButtonPressed();
00605     motorSet(mtr, val);
00606
00607     delay(20);
00608 } while(!done);
00609 disableOpControl = false;
00610 }
00611 }
00612
00620 int selectMotorGroup(FILE *lcdport){
00621     bool done = false;
00622     int val = 0;
00623     do {
00624         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00625         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00626         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00627
00628         if(rightPressed && val != numgroups-1) val++;
00629         else if(rightPressed && val == numgroups-1) val = -1;
00630         else if(leftPressed && val != -1) val--;
00631         else if(leftPressed && val == -1) val = numgroups-1;
00632
00633         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00634         if(val != -1){
00635             memset(motorstr, 0, sizeof(motorstr));
00636             strcpy(motorstr, groups[val].name);
00637         } else {
00638             strcpy(motorstr, "Cancel");
00639         }
00640
00641         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00642         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00643         for(int i = 0; i < spaces; i++){
00644             strcat(str, " ");
00645         }
00646         strcat(str, motorstr);
00647         for(int i = 0; i < spaces; i++){
00648             strcat(str, " ");
00649         }
00650
00651         lcdSetText(LCD_PORT, 1, str);
00652
00653         done = centerPressed;
00654         lcdSetText(LCD_PORT, 2, "<      SEL      >");
00655         delay(20);
00656     } while(!done);
00657     return val;
00658 }
00659
00667 int selectSpdGroup(int mtr) {
00668     bool done = false;
00669     int val;
00670     do {
00671         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00672         /*bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);*/
00673         /*bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);*/
00674
00675         val = (float) ((float) analogRead(AUTON_POT)/(float)
AUTON_POT_HIGH) * 254;
00676         val -= 127;
00677         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00678         char temp[LCD_MESSAGE_MAX_LENGTH+1];
00679         memset(motorstr, 0, sizeof(motorstr));
00680         memset(temp, 0, sizeof(temp));
00681         strcpy(motorstr, groups[mtr].name);
00682
00683         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00684         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00685         for(int i = 0; i < spaces; i++){
00686             strcat(str, " ");

```



```

00687     }
00688     strcat(str, motorstr);
00689     for(int i = 0; i < spaces; i++){
00690         strcat(str, " ");
00691     }
00692
00693     lcdSetText(LCD_PORT, 1, str);
00694
00695     char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00696     memset(speedstr, 0, sizeof(speedstr));
00697     memset(temp, 0, sizeof(temp));
00698     strcpy(speedstr, "Speed: ");
00699     sprintf(temp, "%d", val);
00700     strcat(speedstr, temp);
00701
00702     spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00703     strcpy(str, "");
00704     for(int i = 0; i < spaces; i++){
00705         strcat(str, " ");
00706     }
00707     strcat(str, speedstr);
00708     for(int i = 0; i < spaces; i++){
00709         strcat(str, " ");
00710     }
00711
00712     lcdSetText(LCD_PORT, 2, str);
00713
00714     done = (digitalRead(AUTON_BUTTON) == PRESSED || centerPressed);
00715     delay(20);
00716 } while(!done);
00717 return val;
00718 }
00719
00729 void runGroupMotor(FILE *lcdport){
00730     bool done = false;
00731     int mtr = selectMotorGroup(lcdport);
00732     if(mtr == -1) return;
00733     int spd = selectSpdGroup(mtr);
00734     int val = spd;
00735     disableOpControl = true;
00736     motorStopAll();
00737     do {
00738         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00739         char temp[LCD_MESSAGE_MAX_LENGTH+1];
00740         memset(motorstr, 0, sizeof(motorstr));
00741         memset(temp, 0, sizeof(temp));
00742         strcpy(motorstr, groups[mtr].name);
00743
00744         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00745         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00746         for(int i = 0; i < spaces; i++){
00747             strcat(str, " ");
00748         }
00749         strcat(str, motorstr);
00750         for(int i = 0; i < spaces; i++){
00751             strcat(str, " ");
00752         }
00753
00754         lcdSetText(LCD_PORT, 1, str);
00755
00756         char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00757         memset(speedstr, 0, sizeof(speedstr));
00758         memset(temp, 0, sizeof(temp));
00759         strcat(speedstr, "Run Speed: ");
00760         sprintf(temp, "%d", val);
00761         strcat(speedstr, temp);
00762
00763         spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00764         strcpy(str, "");
00765         for(int i = 0; i < spaces; i++){
00766             strcat(str, " ");
00767         }
00768         strcat(str, speedstr);
00769         for(int i = 0; i < spaces; i++){
00770             strcat(str, " ");
00771         }
00772
00773         lcdSetText(LCD_PORT, 2, str);
00774
00775         for(int i = 1; i <= 10; i++){
00776             if(groups[mtr].motor[i]){

```

```

00777         motorSet(i, val);
00778     }
00779 }
00780
00781     done = lcdAnyButtonPressed();
00782     delay(20);
00783 } while(!done);
00784 disableOpControl = false;
00785 }
00786
00793 void runMotor(FILE *lcdport){
00794     bool done = false;
00795     int val = 0;
00796     do {
00797         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00798         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00799         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00800
00801         if(rightPressed && val != 1) val++;
00802         else if(leftPressed && val != 0) val--;
00803
00804         if(val){
00805             lcdSetText(lcdport, 1, "Indiv Motor Test");
00806         } else {
00807             lcdSetText(lcdport, 1, "Group Motor Test");
00808         }
00809         done = centerPressed;
00810         if(val == 0){
00811             lcdSetText(LCD_PORT, 2, "|      SEL      >");
00812         } else if(val == 1) {
00813             lcdSetText(LCD_PORT, 2, "<      SEL      |");
00814         }
00815         delay(20);
00816     } while(!done);
00817     if(val){
00818         runIndivMotor(lcdport);
00819     } else {
00820         runGroupMotor(lcdport);
00821     }
00822 }
00823
00831 bool* selectMotorGroupMembers(bool *motor){
00832     bool done = false;
00833     int val = 1;
00834     do {
00835         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00836         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00837         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00838
00839         if(rightPressed && val != 10) val++;
00840         else if(rightPressed && val == 10) val = 0;
00841         else if(leftPressed && val != 0) val--;
00842         else if(leftPressed && val == 0) val = 10;
00843
00844         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00845         if(val != 0){
00846             char temp[LCD_MESSAGE_MAX_LENGTH+1];
00847             memset(motorstr, 0, sizeof(motorstr));
00848             memset(temp, 0, sizeof(temp));
00849             strcpy(motorstr, "Motor ");
00850             sprintf(temp, "%d:", val);
00851             strcat(motorstr, temp);
00852             if(motor[val]){
00853                 strcat(motorstr, " On");
00854             } else {
00855                 strcat(motorstr, " Off");
00856             }
00857         } else {
00858             strcpy(motorstr, "Confirm");
00859         }
00860         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00861         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00862         for(int i = 0; i < spaces; i++){
00863             strcat(str, " ");
00864         }
00865         strcat(str, motorstr);
00866         for(int i = 0; i < spaces; i++){
00867             strcat(str, " ");
00868         }
00869
00870         lcdSetText(LCD_PORT, 1, str);

```

```

00871
00872     done = (centerPressed && val == 0);
00873     if(centerPressed && val != 0){
00874         motor[val] = !motor[val];
00875     }
00876     if(val == 0){
00877         lcdSetText(LCD_PORT, 2, "10      SEL      1");
00878     } else if(val == 1) {
00879         lcdSetText(LCD_PORT, 2, "ESC      SEL      2");
00880     } else if(val == 10) {
00881         lcdSetText(LCD_PORT, 2, "9        SEL   ESC");
00882     } else if (val == 9) {
00883         lcdSetText(LCD_PORT, 2, "8        SEL   10");
00884     } else {
00885         char navstr[LCD_MESSAGE_MAX_LENGTH+1];
00886         memset(navstr, 0, sizeof(navstr));
00887         sprintf(navstr, "%c      SEL      %c", (val-1) + '0', (val+1) + '0');
00888         /*navstr[0] = (val-1) + '0';*/
00889         /*strcat(navstr, "      SEL      ");*/
00890         /*navstr[LCD_MESSAGE_MAX_LENGTH] = (val+1) + '0';*/
00891         lcdSetText(LCD_PORT, 2, navstr);
00892     }
00893     delay(20);
00894 } while(!done);
00895 return motor;
00896 }
00897
00901 void addMotorGroup(){
00902     MotorGroup *temp = (MotorGroup*) realloc(groups, sizeof(
00903     MotorGroup)*(numgroups+1));
00904     if(temp == NULL) return;
00905     groups = temp;
00906     numgroups++;
00907     memset(&groups[numgroups-1], 0, sizeof(groups[numgroups-1]));
00908     typeString(groups[numgroups-1].name);
00909     selectMotorGroupMembers(groups[numgroups-1].motor);
00910 }
00917 void editMotorGroup(int mtr){
00918     if(mtr == -1) return;
00919     bool done = false;
00920     int val = 0;
00921     FILE *lcdport = LCD_PORT;
00922     do {
00923         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00924         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00925         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00926
00927         if(rightPressed && val != 1) val++;
00928         else if(leftPressed && val != 0) val--;
00929
00930         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00931         memset(motorstr, 0, sizeof(motorstr));
00932
00933         if(val){
00934             strcpy(motorstr, "Edit Name");
00935         } else {
00936             strcpy(motorstr, "Edit Motors");
00937         }
00938
00939         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00940         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00941         for(int i = 0; i < spaces; i++){
00942             strcat(str, " ");
00943         }
00944         strcat(str, motorstr);
00945         for(int i = 0; i < spaces; i++){
00946             strcat(str, " ");
00947         }
00948
00949         lcdSetText(lcdport, 1, str);
00950
00951         done = centerPressed;
00952         if(val == 0){
00953             lcdSetText(LCD_PORT, 2, "|      SEL      >");
00954         } else if(val == 1) {
00955             lcdSetText(LCD_PORT, 2, "<      SEL      |");
00956         }
00957         delay(20);
00958     } while(!done);
00959     if(val){

```

```

00960         typeString(groups[mtr].name);
00961     } else {
00962         selectMotorGroupMembers(groups[mtr].motor);
00963     }
00964 }
00965
00972 void delMotorGroup(int mtr){
00973     if(mtr == -1) return;
00974     int val = 0;
00975     FILE *lcdport = LCD_PORT;
00976     bool done = false;
00977     do {
00978         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00979         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00980         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00981
00982         if(rightPressed && val != 1) val++;
00983         else if(leftPressed && val != 0) val--;
00984
00985         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00986         memset(motorstr, 0, sizeof(motorstr));
00987
00988         if(val){
00989             strcpy(motorstr, "Delete Forever");
00990         } else {
00991             strcpy(motorstr, "Cancel Delete");
00992         }
00993
00994         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00995         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00996         for(int i = 0; i < spaces; i++){
00997             strcat(str, " ");
00998         }
00999         strcat(str, motorstr);
01000         for(int i = 0; i < spaces; i++){
01001             strcat(str, " ");
01002         }
01003
01004         lcdSetText(lcdport, 1, str);
01005
01006         done = centerPressed;
01007         if(val == 0){
01008             lcdSetText(LCD_PORT, 2, "|      SEL      >");
01009         } else if(val == 1) {
01010             lcdSetText(LCD_PORT, 2, "<      SEL      |");
01011         }
01012         delay(20);
01013     } while(!done);
01014     if(val){
01015         memset(&groups[mtr], 0, sizeof(groups[mtr]));
01016         for(int i = mtr; i < numgroups - 1; i++){
01017             memcpy(&groups[i], &groups[i+1], sizeof(groups[i]));
01018         }
01019         numgroups--;
01020     }
01021 }
01022
01029 void runMotorGroupMgmt(FILE *lcdport){
01030     bool done = false;
01031     int val = 0;
01032     do {
01033         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
01034         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
01035         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
01036
01037         if(rightPressed && val != 3) val++;
01038         else if(rightPressed && val == 3) val = 0;
01039         else if(leftPressed && val != 0) val--;
01040         else if(leftPressed && val == 0) val = 3;
01041
01042         switch(val){
01043             case 0: lcdSetText(lcdport, 1, "Add Motor Group"); break;
01044             case 1: lcdSetText(lcdport, 1, "Edit Motor Group"); break;
01045             case 2: lcdSetText(lcdport, 1, "Del Motor Group"); break;
01046             case 3: lcdSetText(lcdport, 1, "Cancel Grp. Mgmt"); break;
01047         }
01048         done = centerPressed;
01049         if(val == 0){
01050             lcdSetText(LCD_PORT, 2, "<      SEL      >");
01051         } else {
01052             lcdSetText(LCD_PORT, 2, "<      SEL      >");

```

```

01053     }
01054     delay(20);
01055 } while(!done);
01056 switch(val){
01057     case 0: addMotorGroup(); break;
01058     case 1: editMotorGroup(selectMotorGroup(lcdport)); break;
01059     case 2: delMotorGroup(selectMotorGroup(lcdport)); break;
01060     case 3: break;
01061 }
01062 }
01063
01070 void runConnection(FILE *lcdport){
01071     do {
01072         char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01073         char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01074         if(isJoystickConnected(1)){
01075             strcat(strjoy1, "J1: Connected");
01076         } else {
01077             strcat(strjoy1, "J1: Disconnected");
01078         }
01079         if(isJoystickConnected(2)){
01080             strcat(strjoy2, "J2: Connected");
01081         } else {
01082             strcat(strjoy2, "J2: Disconnected");
01083         }
01084
01085         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01086         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01087         for(int i = 0; i < spaces; i++){
01088             strcat(str, " ");
01089         }
01090         strcat(str, strjoy1);
01091         for(int i = 0; i < spaces; i++){
01092             strcat(str, " ");
01093         }
01094
01095         lcdSetText(lcdport, 1, str);
01096
01097         spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01098         strcpy(str, "");
01099         for(int i = 0; i < spaces; i++){
01100             strcat(str, " ");
01101         }
01102         strcat(str, strjoy2);
01103         for(int i = 0; i < spaces; i++){
01104             strcat(str, " ");
01105         }
01106
01107         lcdSetText(lcdport, 2, str);
01108
01109         delay(20);
01110     } while(!lcdAnyButtonPressed());
01111 }
01112
01119 void runRobot(FILE *lcdport){
01120     do {
01121         char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01122         char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01123         if(isOnline()){
01124             strcat(strjoy1, "Competition Mode");
01125         } else {
01126             strcat(strjoy1, "Practice Mode");
01127         }
01128         /*if(isEnabled()){
01129             strcat(strjoy2, "Robot Enabled");
01130         } else {
01131             strcat(strjoy2, "Robot Disabled");
01132         }*/
01133         sprintf(strjoy2, "Angle: %d", gyroGet(gyro));
01134
01135         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01136         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01137         for(int i = 0; i < spaces; i++){
01138             strcat(str, " ");
01139         }
01140         strcat(str, strjoy1);
01141         for(int i = 0; i < spaces; i++){
01142             strcat(str, " ");
01143         }
01144
01145         lcdSetText(lcdport, 1, str);

```

```

01146
01147     spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01148     strcpy(str, "");
01149     for(int i = 0; i < spaces; i++){
01150         strcat(str, " ");
01151     }
01152     strcat(str, strjoy2);
01153     for(int i = 0; i < spaces; i++){
01154         strcat(str, " ");
01155     }
01156
01157     lcdSetText(lcdport, 2, str);
01158
01159     delay(20);
01160 } while(!lcdAnyButtonPressed());
01161 }
01162
01170 void runAuton(FILE *lcdport){
01171     do {
01172         char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01173         char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01174         if(autonLoaded == -1){
01175             strcat(strjoy1, "No Auton Loaded");
01176         } else if(autonLoaded == 0){
01177             strcat(strjoy1, "Empty Auton");
01178         } else if(autonLoaded == MAX_AUTON_SLOTS + 1){
01179             strcat(strjoy1, "Prog. Skills");
01180         } else {
01181             FILE* autonFile;
01182             char filename[AUTON_FILENAME_MAX_LENGTH];
01183             snprintf(filename, sizeof(filename)/sizeof(char), "a%d", autonLoaded);
01184             autonFile = fopen(filename, "r");
01185             char name[LCD_MESSAGE_MAX_LENGTH+1];
01186             memset(name, 0, sizeof(name));
01187             fread(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
01188             strcpy(strjoy1, name);
01189             fclose(autonFile);
01190         }
01191         if(isOnline()){
01192             strcat(strjoy2, "Recorder Off");
01193         } else {
01194             strcat(strjoy2, "Recorder On");
01195         }
01196
01197         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01198         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01199         for(int i = 0; i < spaces; i++){
01200             strcat(str, " ");
01201         }
01202         strcat(str, strjoy1);
01203         for(int i = 0; i < spaces; i++){
01204             strcat(str, " ");
01205         }
01206
01207         lcdSetText(lcdport, 1, str);
01208
01209         spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01210         strcpy(str, "");
01211         for(int i = 0; i < spaces; i++){
01212             strcat(str, " ");
01213         }
01214         strcat(str, strjoy2);
01215         for(int i = 0; i < spaces; i++){
01216             strcat(str, " ");
01217         }
01218
01219         lcdSetText(lcdport, 2, str);
01220
01221         delay(20);
01222     } while(!lcdAnyButtonPressed());
01223 }
01224
01235 void runCredits(FILE *lcdport){
01236     do {
01237         char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01238         char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01239         strcat(strjoy1, "Thanks Team 750W");
01240         strcat(strjoy2, "Akram Sandhu");
01241
01242         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01243         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";

```

```

01244         for(int i = 0; i < spaces; i++){
01245             strcat(str, " ");
01246         }
01247         strcat(str, strjoy1);
01248         for(int i = 0; i < spaces; i++){
01249             strcat(str, " ");
01250         }
01251
01252         lcdSetText(lcdport, 1, str);
01253
01254         spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01255         strcpy(str, "");
01256         for(int i = 0; i < spaces; i++){
01257             strcat(str, " ");
01258         }
01259         strcat(str, strjoy2);
01260         for(int i = 0; i < spaces; i++){
01261             strcat(str, " ");
01262         }
01263
01264         lcdSetText(lcdport, 2, str);
01265
01266         delay(20);
01267     } while(!lcdAnyButtonPressed());
01268 }
01269
01278 void doMenuChoice(int choice){
01279     switch(choice){
01280         case MENU_SCREENSAVER: runScreensaver(
LCD_PORT); break;
01281         case MENU_BATTERY: runBattery(LCD_PORT); break;
01282         case MENU_MOTOR: runMotor(LCD_PORT); break;
01283         case MENU_MOTOR_MGMT: runMotorGroupMgmt(
LCD_PORT); break;
01284         case MENU_CONNECTION: runConnection(LCD_PORT); break;
01285         case MENU_ROBOT: runRobot(LCD_PORT); break;
01286         case MENU_AUTON: runAuton(LCD_PORT); break;
01287         case MENU_BACKLIGHT: lcdSetBacklight(LCD_PORT, !
backlight); backlight = !backlight; break;
01288         case MENU_CREDITS: runCredits(LCD_PORT); break;
01289     }
01290 }
01291
01300 void formatLCDDisplay(void *ignore){
01301     lcdSetBacklight(LCD_PORT, backlight);
01302     doMenuChoice(MENU_SCREENSAVER);
01303     while(true){
01304         doMenuChoice(selectMenu());
01305     }
01306 }

```

4.31 src/lcdmsg.c File Reference

File for LCD message code.

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

Functions

- void [randlcdmsg](#) (FILE *lcdport, int line)
- void [screensaver](#) (FILE *lcdport)

Variables

- char * [lcdmsg](#) []

4.31.1 Detailed Description

File for LCD message code.

This file contains the code for the LCD screensaver messages. These messages display randomly while the LCD diagnostic menu is set to the screensaver mode. These messages are mainly inside jokes among the team.

Definition in file [lcdmsg.c](#).

4.31.2 Function Documentation

4.31.2.1 void randlcdmsg (FILE * *lcdport*, int *line*)

Displays a random LCD message from the master list.

Parameters

<i>lcdport</i>	the port the LCD is connected to
<i>line</i>	the line to display the message on

Definition at line [67](#) of file [lcdmsg.c](#).

4.31.2.2 void screensaver (FILE * *lcdport*)

Formats the LCD by displaying 750C title and message.

Parameters

<i>lcdport</i>	the port the LCD is connected to
----------------	----------------------------------

Definition at line [86](#) of file [lcdmsg.c](#).

4.31.3 Variable Documentation

4.31.3.1 char* lcdmsg[]

Master list of all LCD messages.

Definition at line [14](#) of file [lcdmsg.c](#).

4.32 lcdmsg.c

```

00001
00009 #include "main.h"
00010
00014 char *lcdmsg[] = {
00015     "Hint of Lime",
00016     "Review Committee",
00017     "Exacerbate",
00018     "Get Schwifty",
00019     "Hot Knife",
00020     "Mikel",
00021     "Michael",
00022     "Donald J. Trump",
00023     "Nautilus Gears",
00024     "Drop and Pop",
00025     "More Flip",
00026     "Alfred & Robin",
00027     "Quinnipiac",
00028     "Yeah",
00029     "YEAH",
00030     "YEAH!",
00031     "Did You Get That",
00032     "On Video",
00033     "MGNT",
00034     "LC/DC",
00035     "You Seem Tense",
00036     "Root Theta Gears",
00037     "Money Shot",
00038     "Jaskirat Vig",
00039     "Torsionify",
00040     "Chortler",
00041     "750 Crust",
00042     "Crusty Teeth",
00043     "Moneyballs",
00044     "dooDAH",
00045     "Wobbly Pobbly",
00046     "Hallen Key",
00047     "24 Motors",
00048     "Complicate THIS!",
00049     "Meshing",
00050     "Shikhar Crustogi",
00051     "Neural Network",
00052     "easyAuton()",
00053     "Loctite",
00054     "Skin Irritant",
00055     "Akram Sandhu",
00056     "Therapy Dogs",
00057     "Staples Bands",
00058     "itoa() Sucks"
00059 };
00060
00067 void randlcdmsg(FILE *lcdport, int line){
00068     int index = rand() % LCD_MESSAGE_COUNT;
00069     int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(lcdmsg[index]))/2;
00070     char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00071     for(int i = 0; i < spaces; i++){
00072         strcat(str, " ");
00073     }
00074     strcat(str, lcdmsg[index]);
00075     for(int i = 0; i < spaces; i++){
00076         strcat(str, " ");
00077     }
00078     lcdSetText(lcdport, line, str);
00079 }
00080
00086 void screensaver(FILE *lcdport) {
00087     lcdSetText(lcdport, 1, LCD_750C_TITLE);
00088     randlcdmsg(lcdport, 2);
00089 }

```

4.33 src/motors.c File Reference

File for important motor functions.

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

Functions

- void [transmissionSetPos](#) (void *pos)

4.33.1 Detailed Description

File for important motor functions.

This file contains the code for functions regarding motor status. These functions are too complex to be defined as inline functions in the [motors.h](#) file.

See the [motors.h](#) file for the basic movement functions.

See also

[motors.h](#)

Definition in file [motors.c](#).

4.33.2 Function Documentation

4.33.2.1 void transmissionSetPos (void * pos)

Sets the position of the transmission.

Parameters

<i>pos</i>	the position to set the transmission to.
------------	--

Definition at line 19 of file [motors.c](#).

4.34 motors.c

```

00001
00012 #include "main.h"
00013
00019 void transmissionSetPos(void *pos){
00020     int pot = (intptr_t) pos;
00021     printf("Target: %d\n", pot);
00022     if(analogRead(TRANSMISSION_POT) < pot) {
00023         while(analogRead(TRANSMISSION_POT) < pot){
00024             printf("Current: %d, Target: %d\n", analogRead(TRANSMISSION_POT), pot);
00025             transmission(sign(analogRead(TRANSMISSION_POT)-pot)*50);
00026             delay(20);
00027         }
00028     } else if(analogRead(TRANSMISSION_POT) > pot){
00029         while(analogRead(TRANSMISSION_POT) > pot){
00030             printf("Current: %d, Target: %d\n", analogRead(TRANSMISSION_POT), pot);
00031             transmission(sign(analogRead(TRANSMISSION_POT)-pot)*50);
00032             delay(20);
00033         }
00034     }
00035     printf("Task loop completed.\n");
00036     transmission(0);
00037 }
```

4.35 src/opcontrol.c File Reference

File for operator control code.

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

Functions

- void [targetNet](#) (int target)
- void [recordJoyInfo](#) ()
- void [moveRobot](#) ()
- void [operatorControl](#) ()

Variables

- int [spd](#)
- int [turn](#)
- int [sht](#)
- int [intk](#)
- int [trans](#)
- int [dep](#)

4.35.1 Detailed Description

File for operator control code.

This file should contain the user [operatorControl\(\)](#) function and any functions related to it.

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

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

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

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

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

Definition in file [opcontrol.c](#).

4.35.2 Function Documentation

4.35.2.1 void moveRobot ()

Moves the robot based on the motor state variables.

Definition at line 128 of file [opcontrol.c](#).

4.35.2.2 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 153 of file [opcontrol.c](#).

4.35.2.3 void recordJoyInfo ()

Populates the motor state variables based on the joystick's current values.

Definition at line 81 of file [opcontrol.c](#).

4.35.2.4 void targetNet (int *target*)

Faces the robot towards the desired gyroscope angle by turning it. This function implements a simple PID control loop in order to correct for error.

Parameters

<i>target</i>	the desired robot angle
---------------	-------------------------

Definition at line 73 of file [opcontrol.c](#).

4.35.3 Variable Documentation

4.35.3.1 int dep

Speed of the lift deployment motor.

Definition at line 65 of file [opcontrol.c](#).

4.35.3.2 int intk

Speed of the intake motors.

Definition at line 55 of file [opcontrol.c](#).

4.35.3.3 int sht

Speed of the shooter motors.

Definition at line 50 of file [opcontrol.c](#).

4.35.3.4 int spd

Forward/backward speed of the drive motors.

Definition at line 40 of file [opcontrol.c](#).

4.35.3.5 int trans

Speed of the transmission motors.

Definition at line 60 of file [opcontrol.c](#).

4.35.3.6 int turn

Turning speed of the drive motors.

Definition at line 45 of file [opcontrol.c](#).

4.36 opcontrol.c

```
00001
00035 #include "main.h"
00036
00040 int spd;
00041
00045 int turn;
00046
00050 int sht;
00051
00055 int intk;
00056
00060 int trans;
00061
00065 int dep;
00066
00073 void targetNet(int target){
00074     int error = gyroGet(gyro) % ROTATION_DEG - target;
00075     turn = error * 20;
00076 }
00077
00081 void recordJoyInfo(){
00082     spd = joystickGetAnalog(1, 3);
00083     turn = joystickGetAnalog(1, 1);
```

```

00084     sht = 0;
00085     intk = 0;
00086     trans = 0;
00087     if(joystickGetDigital(1, 6, JOY_UP) || joystickGetDigital(2, 6, JOY_UP)){
00088         sht = 127;
00089     } else if(joystickGetDigital(1, 6, JOY_DOWN) || joystickGetDigital(2, 6, JOY_DOWN)){
00090         sht = -127;
00091     } else {
00092         sht = 0;
00093     }
00094     if(joystickGetDigital(1, 5, JOY_UP) || joystickGetDigital(2, 5, JOY_UP)){
00095         intk = 127;
00096     } else if(joystickGetDigital(1, 5, JOY_DOWN) || joystickGetDigital(2, 5, JOY_DOWN)){
00097         intk = -127;
00098     } else {
00099         intk = 0;
00100     }
00101     if(joystickGetDigital(1, 8, JOY_LEFT)){
00102         trans = 80;
00103         /*changeGear(GEAR_LIFT);*/
00104     } else if(joystickGetDigital(1, 8, JOY_RIGHT)){
00105         trans = -80;
00106         /*changeGear(GEAR_DRIVE);*/
00107     } else {
00108         trans = 0;
00109     }
00110     if(joystickGetDigital(1, 8, JOY_UP) || joystickGetDigital(2, 8, JOY_UP)){
00111         dep = 127;
00112     } else if(joystickGetDigital(1, 8, JOY_DOWN) || joystickGetDigital(2, 8, JOY_DOWN)){
00113         dep = -127;
00114     } else {
00115         dep = 0;
00116     }
00117
00118     if(joystickGetDigital(1, 7, JOY_DOWN)){
00119         targetNet(GYRO_NET_TARGET);
00120     } else if(joystickGetDigital(1, 7, JOY_UP)){
00121         gyroReset(gyro);
00122     }
00123 }
00124
00128 void moveRobot(){
00129     move(spd, turn);
00130     shoot(sht);
00131     intake(intk);
00132     transmission(trans);
00133     deploy(dep);
00134 }
00135
00153 void operatorControl() {
00154     lcdSetBacklight(LCD_PORT, true);
00155     while (true) {
00156         if(isOnline() || progSkills == 0){
00157             if(lcdDiagTask == NULL){
00158                 lcdDiagTask = taskCreate(formatLCDDisplay,
TASK_DEFAULT_STACK_SIZE, NULL, TASK_PRIORITY_DEFAULT);
00159             } else if(taskGetState(lcdDiagTask) == TASK_SUSPENDED){
00160                 taskResume(lcdDiagTask);
00161             }
00162             if(!disableOpControl){
00163                 recordJoyInfo();
00164                 if (joystickGetDigital(1, 7, JOY_RIGHT) && !isOnline()) {
00165                     taskSuspend(lcdDiagTask);
00166                     recordAuton();
00167                     lcdSetBacklight(LCD_PORT, true);
00168                     saveAuton();
00169                 } else if (joystickGetDigital(1, 7, JOY_LEFT) && !isOnline()) {
00170                     taskSuspend(lcdDiagTask);
00171                     lcdSetBacklight(LCD_PORT, true);
00172                     loadAuton();
00173                     playbackAuton();
00174                 }
00175                 moveRobot();
00176             }
00177         } else {
00178             motorStopAll();
00179             lcdSetText(LCD_PORT, 1, "Press 7R");
00180             lcdPrint(LCD_PORT, 2, "Last Skills: %d", progSkills);
00181             if (joystickGetDigital(1, 7, JOY_RIGHT) && !isOnline()) {
00182                 recordAuton();
00183                 saveAuton();

```

```
00184         } else if(joystickGetDigital(1, 7, JOY_UP) && !isOnline()) {  
00185             progSkills = 0;  
00186         }  
00187     }  
00188     delay(20);  
00189 }  
00190 }
```

4.37 src/sensors.c File Reference

File for important sensor declarations and functions.

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

Variables

- Encoder [leftenc](#)
- Encoder [rightenc](#)
- Gyro [gyro](#)

4.37.1 Detailed Description

File for important sensor declarations and functions.

This file contains the code for declarations and functions regarding sensors. The definitions contained herein define objects representing more complex sensors. The functions contained herein are too complex to be defined as inline functions in the [sensors.h](#) file.

See the [sensors.h](#) file for the basic sensory definitions and functions.

See also

[sensors.h](#)

Definition in file [sensors.c](#).

4.37.2 Variable Documentation

4.37.2.1 Gyro gyro

Object representing the gyroscope.

Definition at line 28 of file [sensors.c](#).

4.37.2.2 Encoder leftenc

Object representing the encoder on the left side of the drivetrain.

Definition at line 18 of file [sensors.c](#).

4.37.2.3 Encoder rightenc

Object representing the encoder on the right side of the drivetrain.

Definition at line 23 of file [sensors.c](#).

4.38 sensors.c

```
00001
00013 #include "main.h"
00014
00018 Encoder leftenc;
00019
00023 Encoder rightenc;
00024
00028 Gyro gyro;
```


Index

- AUTON_BUTTON
 - autonrecorder.h, 6
- AUTON_FILENAME_MAX_LENGTH
 - autonrecorder.h, 6
- AUTON_POT_HIGH
 - autonrecorder.h, 6
- AUTON_POT_LOW
 - autonrecorder.h, 6
- AUTON_POT
 - autonrecorder.h, 6
- AUTON_TIME
 - autonrecorder.h, 6
- abs
 - macros.h, 23
- addMotorGroup
 - lcddiag.c, 54
- auto.c
 - autonomous, 43
- autonLoaded
 - autonrecorder.c, 45
 - autonrecorder.h, 8
- autonomous
 - auto.c, 43
 - main.h, 29
- autonrecorder.c
 - autonLoaded, 45
 - initAutonRecorder, 45
 - loadAuton, 45
 - playbackAuton, 45
 - progSkills, 45
 - recordAuton, 45
 - saveAuton, 45
 - selectAuton, 45
 - states, 46
- autonrecorder.h
 - AUTON_BUTTON, 6
 - AUTON_FILENAME_MAX_LENGTH, 6
 - AUTON_POT_HIGH, 6
 - AUTON_POT_LOW, 6
 - AUTON_POT, 6
 - AUTON_TIME, 6
 - autonLoaded, 8
 - initAutonRecorder, 7
 - JOY_POLL_FREQ, 7
 - joyState, 7
 - loadAuton, 7
 - MAX_AUTON_SLOTS, 7
 - PROGSKILL_TIME, 7
 - playbackAuton, 7
 - progSkills, 8
 - recordAuton, 8
 - saveAuton, 8
 - states, 8
- BATT_BKUP
 - sensors.h, 39
- BATT_MAIN
 - sensors.h, 39
- BATT_PEXP
 - sensors.h, 39
- backlight
 - lcddiag.c, 60
- bitClear
 - bitwise.h, 10
- bitRead
 - bitwise.h, 10
- bitSet
 - bitwise.h, 10
- bitWrite
 - bitwise.h, 10
- bitwise.h
 - bitClear, 10
 - bitRead, 10
 - bitSet, 10
 - bitWrite, 10
- changeGear
 - motors.h, 33
- constants.h
 - DEG_TO_RAD, 12
 - HALF_PI, 12
 - MATH_PI, 12
 - MATH_E, 12
 - PI, 12
 - RAD_TO_DEG, 12
 - ROTATION_DEG, 12
 - ROTATION_RAD, 12
 - TAU, 12
 - TWO_PI, 13
- constrain
 - macros.h, 24
- DEG_TO_RAD
 - constants.h, 12
- degrees
 - macros.h, 24
- delMotorGroup
 - lcddiag.c, 54
- dep
 - joyState, 3
 - opcontrol.c, 80
 - opcontrol.h, 37
- deploy

- motors.h, 34
- disableOpControl
 - lcddiag.c, 60
 - lcddiag.h, 19
- doMenuChoice
 - lcddiag.c, 54
- editMotorGroup
 - lcddiag.c, 54
- formatLCDDisplay
 - lcddiag.c, 55
 - lcddiag.h, 17
- formatMenuNameCenter
 - lcddiag.c, 55
- friendly.h
 - PRESSED, 14
 - RELEASED, 14
 - UNPRESSED, 14
 - UNRELEASED, 14
- GEAR_DRIVE
 - sensors.h, 39
- GEAR_LIFT
 - sensors.h, 40
- GYRO_NET_TARGET
 - sensors.h, 40
- GYRO_PORT
 - sensors.h, 40
- GYRO_SENSITIVITY
 - sensors.h, 40
- groups
 - lcddiag.c, 61
 - lcddiag.h, 19
- gyro
 - sensors.c, 83
 - sensors.h, 41
- HALF_PI
 - constants.h, 12
- INTAKE_ROLLER_MOTOR
 - motors.h, 32
- include/autonrecorder.h, 5, 9
- include/bitwise.h, 9, 11
- include/constants.h, 11, 13
- include/friendly.h, 13, 14
- include/lcddiag.h, 15, 20
- include/lcdmsg.h, 21, 23
- include/macros.h, 23, 27
- include/main.h, 28, 31
- include/motors.h, 31, 35
- include/opcontrol.h, 36, 38
- include/sensors.h, 38, 42
- init.c
 - initialize, 51
 - initializeIO, 51
- initAutonRecorder
 - autonrecorder.c, 45
 - autonrecorder.h, 7
- initGroups
 - lcddiag.c, 55
 - lcddiag.h, 18
- initialize
 - init.c, 51
 - main.h, 29
- initializeIO
 - init.c, 51
 - main.h, 30
- intake
 - motors.h, 34
- intk
 - joyState, 3
 - opcontrol.c, 81
 - opcontrol.h, 37
- JOY_POLL_FREQ
 - autonrecorder.h, 7
- joyState, 3
 - autonrecorder.h, 7
 - dep, 3
 - intk, 3
 - sht, 3
 - spd, 4
 - trans, 4
 - turn, 4
- LCD_750C_TITLE
 - lcdmsg.h, 21
- LCD_MENU_COUNT
 - lcddiag.h, 16
- LCD_MESSAGE_COUNT
 - lcdmsg.h, 21
- LCD_MESSAGE_MAX_LENGTH
 - lcdmsg.h, 22
- LCD_PORT
 - lcdmsg.h, 22
- LEFT_ENC_BOT
 - sensors.h, 40
- LEFT_ENC_TOP
 - sensors.h, 40
- LEFT_MOTOR_BOT
 - motors.h, 32
- LEFT_MOTOR_TOP
 - motors.h, 32
- LIFT_DEPLOY
 - motors.h, 33
- lcdAnyButtonPressed
 - lcddiag.h, 18
- lcdButtonPressed

- lcddiag.h, 18
- lcdDiagTask
 - lcddiag.c, 61
 - lcddiag.h, 19
- lcddiag.c
 - addMotorGroup, 54
 - backlight, 60
 - delMotorGroup, 54
 - disableOpControl, 60
 - doMenuChoice, 54
 - editMotorGroup, 54
 - formatLCDDisplay, 55
 - formatMenuNameCenter, 55
 - groups, 61
 - initGroups, 55
 - lcdDiagTask, 61
 - loadGroups, 55
 - menuChoices, 61
 - numgroups, 61
 - runAuton, 55
 - runBattery, 56
 - runConnection, 56
 - runCredits, 56
 - runGroupMotor, 56
 - runIndivMotor, 57
 - runMotor, 57
 - runMotorGroupMgmt, 57
 - runRobot, 57
 - runScreensaver, 58
 - saveGroups, 58
 - selectMenu, 58
 - selectMotor, 58
 - selectMotorGroup, 59
 - selectMotorGroupMembers, 59
 - selectSpd, 59
 - selectSpdGroup, 60
 - typeString, 60
- lcddiag.h
 - disableOpControl, 19
 - formatLCDDisplay, 17
 - groups, 19
 - initGroups, 18
 - LCD_MENU_COUNT, 16
 - lcdAnyButtonPressed, 18
 - lcdButtonPressed, 18
 - lcdDiagTask, 19
 - MENU_AUTON, 16
 - MENU_BACKLIGHT, 16
 - MENU_BATTERY, 16
 - MENU_CONNECTION, 17
 - MENU_CREDITS, 17
 - MENU_MOTOR_MGMT, 17
 - MENU_MOTOR, 17
 - MENU_ROBOT, 17
- MENU_SCREENSAVER, 17
 - menuChoices, 19
 - MotorGroup, 17
 - numgroups, 20
 - typeString, 19
- lcdmsg
 - lcdmsg.c, 76
 - lcdmsg.h, 22
- lcdmsg.c
 - lcdmsg, 76
 - randlcdmsg, 76
 - screensaver, 76
- lcdmsg.h
 - LCD_750C_TITLE, 21
 - LCD_MESSAGE_COUNT, 21
 - LCD_MESSAGE_MAX_LENGTH, 22
 - LCD_PORT, 22
 - lcdmsg, 22
 - randlcdmsg, 22
 - screensaver, 22
- leftenc
 - sensors.c, 83
 - sensors.h, 41
- loadAuton
 - autonrecorder.c, 45
 - autonrecorder.h, 7
- loadGroups
 - lcddiag.c, 55
- MATH_PI
 - constants.h, 12
- MATH_E
 - constants.h, 12
- MAX_AUTON_SLOTS
 - autonrecorder.h, 7
- MAX
 - macros.h, 25
- MENU_AUTON
 - lcddiag.h, 16
- MENU_BACKLIGHT
 - lcddiag.h, 16
- MENU_BATTERY
 - lcddiag.h, 16
- MENU_CONNECTION
 - lcddiag.h, 17
- MENU_CREDITS
 - lcddiag.h, 17
- MENU_MOTOR_MGMT
 - lcddiag.h, 17
- MENU_MOTOR
 - lcddiag.h, 17
- MENU_ROBOT
 - lcddiag.h, 17
- MENU_SCREENSAVER

- lccdiag.h, 17
- MIN
 - macros.h, 25
- MOTOR_MAX
 - motors.h, 33
- MOTOR_MIN
 - motors.h, 33
- macros.h
 - abs, 23
 - constrain, 24
 - degrees, 24
 - MAX, 25
 - MIN, 25
 - max, 24
 - min, 25
 - radians, 26
 - round, 26
 - sign, 26
 - SQ, 27
 - sq, 27
- main.h
 - autonomous, 29
 - initialize, 29
 - initializeIO, 30
 - operatorControl, 30
- max
 - macros.h, 24
- menuChoices
 - lccdiag.c, 61
 - lccdiag.h, 19
- min
 - macros.h, 25
- motor
 - MotorGroup, 4
- MotorGroup, 4
 - lccdiag.h, 17
 - motor, 4
 - name, 4
- motors.c
 - transmissionSetPos, 78
- motors.h
 - changeGear, 33
 - deploy, 34
 - INTAKE_ROLLER_MOTOR, 32
 - intake, 34
 - LEFT_MOTOR_BOT, 32
 - LEFT_MOTOR_TOP, 32
 - LIFT_DEPLOY, 33
 - MOTOR_MAX, 33
 - MOTOR_MIN, 33
 - move, 34
 - NAUTILUS_SHOOTER_MOTOR_CENTER, 33
 - NAUTILUS_SHOOTER_MOTOR_LEFT, 33
 - NAUTILUS_SHOOTER_MOTOR_RIGHT, 33
 - RIGHT_MOTOR_BOT, 33
 - RIGHT_MOTOR_TOP, 33
 - SHOOTER_HAS_THREE_MOTORS, 33
 - shoot, 34
 - TRANSMISSION_MOTOR, 33
 - transmission, 34
 - transmissionSetPos, 35
- move
 - motors.h, 34
- moveRobot
 - opcontrol.c, 80
 - opcontrol.h, 37
- NAUTILUS_SHOOTER_MOTOR_CENTER
 - motors.h, 33
- NAUTILUS_SHOOTER_MOTOR_LEFT
 - motors.h, 33
- NAUTILUS_SHOOTER_MOTOR_RIGHT
 - motors.h, 33
- NUM_BATTS
 - sensors.h, 40
- name
 - MotorGroup, 4
- numgroups
 - lccdiag.c, 61
 - lccdiag.h, 20
- opcontrol.c
 - dep, 80
 - intk, 81
 - moveRobot, 80
 - operatorControl, 80
 - recordJoyInfo, 80
 - sht, 81
 - spd, 81
 - targetNet, 80
 - trans, 81
 - turn, 81
- opcontrol.h
 - dep, 37
 - intk, 37
 - moveRobot, 37
 - recordJoyInfo, 37
 - sht, 37
 - spd, 37
 - trans, 37
 - turn, 37
- operatorControl
 - main.h, 30
 - opcontrol.c, 80
- POWER_EXPANDER_STATUS
 - sensors.h, 40
- POWER_EXPANDER_VOLTAGE_DIVISOR
 - sensors.h, 41

PRESSED
 friendly.h, 14
PROGSKILL_TIME
 autonrecorder.h, 7
PI
 constants.h, 12
playbackAuton
 autonrecorder.c, 45
 autonrecorder.h, 7
powerLevelExpander
 sensors.h, 41
progSkills
 autonrecorder.c, 45
 autonrecorder.h, 8

RAD_TO_DEG
 constants.h, 12
RELEASED
 friendly.h, 14
RIGHT_ENC_BOT
 sensors.h, 41
RIGHT_ENC_TOP
 sensors.h, 41
RIGHT_MOTOR_BOT
 motors.h, 33
RIGHT_MOTOR_TOP
 motors.h, 33
ROTATION_DEG
 constants.h, 12
ROTATION_RAD
 constants.h, 12
radians
 macros.h, 26
randLcdmsg
 lcdmsg.c, 76
 lcdmsg.h, 22
recordAuton
 autonrecorder.c, 45
 autonrecorder.h, 8
recordJoyInfo
 opcontrol.c, 80
 opcontrol.h, 37
rightenc
 sensors.c, 84
 sensors.h, 42
round
 macros.h, 26
runAuton
 lcddiag.c, 55
runBattery
 lcddiag.c, 56
runConnection
 lcddiag.c, 56
runCredits
 lcddiag.c, 56
runGroupMotor
 lcddiag.c, 56
runIndivMotor
 lcddiag.c, 57
runMotor
 lcddiag.c, 57
runMotorGroupMgmt
 lcddiag.c, 57
runRobot
 lcddiag.c, 57
runScreensaver
 lcddiag.c, 58

SHOOTER_HAS_THREE_MOTORS
 motors.h, 33
saveAuton
 autonrecorder.c, 45
 autonrecorder.h, 8
saveGroups
 lcddiag.c, 58
screensaver
 lcdmsg.c, 76
 lcdmsg.h, 22
selectAuton
 autonrecorder.c, 45
selectMenu
 lcddiag.c, 58
selectMotor
 lcddiag.c, 58
selectMotorGroup
 lcddiag.c, 59
selectMotorGroupMembers
 lcddiag.c, 59
selectSpd
 lcddiag.c, 59
selectSpdGroup
 lcddiag.c, 60
sensors.c
 gyro, 83
 leftenc, 83
 rightenc, 84
sensors.h
 BATT_BKUP, 39
 BATT_MAIN, 39
 BATT_PEXP, 39
 GEAR_DRIVE, 39
 GEAR_LIFT, 40
 GYRO_NET_TARGET, 40
 GYRO_PORT, 40
 GYRO_SENSITIVITY, 40
 gyro, 41
 LEFT_ENC_BOT, 40
 LEFT_ENC_TOP, 40

- leftenc, [41](#)
- NUM_BATTS, [40](#)
- POWER_EXPANDER_STATUS, [40](#)
- POWER_EXPANDER_VOLTAGE_DIVISOR, [41](#)
- powerLevelExpander, [41](#)
- RIGHT_ENC_BOT, [41](#)
- RIGHT_ENC_TOP, [41](#)
- rightenc, [42](#)
- TRANSMISSION_POT, [41](#)
- shoot
 - motors.h, [34](#)
- sht
 - joyState, [3](#)
 - opcontrol.c, [81](#)
 - opcontrol.h, [37](#)
- sign
 - macros.h, [26](#)
- spd
 - joyState, [4](#)
 - opcontrol.c, [81](#)
 - opcontrol.h, [37](#)
- SQ
 - macros.h, [27](#)
- sq
 - macros.h, [27](#)
- src/auto.c, [42](#), [44](#)
- src/autonrecorder.c, [44](#), [46](#)
- src/init.c, [50](#), [52](#)
- src/lcddiag.c, [52](#), [62](#)
- src/lcdmsg.c, [75](#), [77](#)
- src/motors.c, [77](#), [78](#)
- src/opcontrol.c, [79](#), [81](#)
- src/sensors.c, [83](#), [84](#)
- states
 - autonrecorder.c, [46](#)
 - autonrecorder.h, [8](#)
- TAU
 - constants.h, [12](#)
- TRANSMISSION_MOTOR
 - motors.h, [33](#)
- TRANSMISSION_POT
 - sensors.h, [41](#)
- TWO_PI
 - constants.h, [13](#)
- targetNet
 - opcontrol.c, [80](#)
- trans
 - joyState, [4](#)
 - opcontrol.c, [81](#)
 - opcontrol.h, [37](#)
- transmission
 - motors.h, [34](#)
- transmissionSetPos
 - motors.c, [78](#)
 - motors.h, [35](#)
- turn
 - joyState, [4](#)
 - opcontrol.c, [81](#)
 - opcontrol.h, [37](#)
- typeString
 - lcddiag.c, [60](#)
 - lcddiag.h, [19](#)
- UNPRESSED
 - friendly.h, [14](#)
- UNRELEASED
 - friendly.h, [14](#)