Team 750C Nothing But Net Code Reference

Competition_2016-02-06_BCIT

Contents

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 joyState Struct Reference

Representation of the operator controller's instructions at a point in time.

```
#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

Representation of the operator controller's instructions at a point in time.

This state represents the values of the motors at a point in time. These instructions are played back at the rate polled to send the same commands the operator did.

Definition at line 67 of file autonrecorder.h.

3.1.2 Member Data Documentation

3.1.2.1 signed char joyState::dep

Speed of the lift deployment motor.

Definition at line 96 of file autonrecorder.h.

3.1.2.2 signed char joyState::intk

Speed of the intake motor.

Definition at line 86 of file autonrecorder.h.

3.1.2.3 signed char joyState::sht

Speed of the shooter motors.

Definition at line 81 of file autonrecorder.h.

3.1.2.4 signed char joyState::spd

Forward/backward speed of the drive motors.

Definition at line 71 of file autonrecorder.h.

3.1.2.5 signed char joyState::trans

Speed of the transmission motor.

Definition at line 91 of file autonrecorder.h.

3.1.2.6 signed char joyState::turn

Turning speed of the drive motors.

Definition at line 76 of file autonrecorder.h.

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

- include/autonrecorder.h

3.2 MotorGroup Struct Reference

Represents a logical motor grouping, to be used when testing motors.

`#include <lcddiag.h>`

Public Attributes

- bool motor [11]
- char name [LCD_MESSAGE_MAX_LENGTH+1]

3.2.1 Detailed Description

Represents a logical motor grouping, to be used when testing motors.

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

Definition at line 143 of file lcddiag.h.

3.2.2 Member Data Documentation

3.2.2.1 bool MotorGroup::motor[11]

Stores if each motor is contained in this group or not.

This array contains 11 values. Element 0 is ignored. Elements 1-10 represent the respective motor ports.

Definition at line 149 of file lcddiag.h.

3.2.2.2 char MotorGroup::name[LCD_MESSAGE_MAX_LENGTH+1]

The name of the motor group.

The name can be a maximum of 16 characters. The buffer is 17 characters to hold the null terminator.

Definition at line 157 of file lcddiag.h.

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

- include/lcddiag.h

4 File Documentation

4.1 include/autonrecorder.h File Reference

Header file for autonomous recorder functions and definitions.

Classes

- struct joyState

  Representation of the operator controller's instructions at a point in time.

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 9
- #define AUTON_POT_LOW 0
- #define AUTON_POT_HIGH 4095

Typedefs

- typedef struct joyState joyState

  Representation of the operator controller's instructions at a point in time.

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 9

Button for confirming selection of an autonomous routine.

Definition at line 49 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 39 of file autonrecorder.h.

4.1.2.3   #define AUTON_POT 1

Potentiometer for selecting which autonomous routine to load.

Definition at line 44 of file autonrecorder.h.

4.1.2.4   #define AUTON_POT_HIGH 4095

Upper limit of the autonomous routine selector potentiometer.

Definition at line 59 of file autonrecorder.h.

4.1.2.5   #define AUTON_POT_LOW 0

Lower limit of the autonomous routine selector potentiometer.

Definition at line 54 of file autonrecorder.h.

4.1.2.6   #define AUTON_TIME 15

Number of seconds the autonomous period lasts.

Definition at line 17 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 29 of file autonrecorder.h.

4.1.2.8   #define MAX_AUTON_SLOTS 10

Maximum number of autonomous routines to be stored.

Definition at line 34 of file autonrecorder.h.

4.1.2.9   #define PROGSKILL_TIME 60

Number of seconds the programming skills challenge lasts.

Definition at line 22 of file autonrecorder.h.

4.1.3   Typedef Documentation

4.1.3.1   typedef struct joyState joyState

Representation of the operator controller's instructions at a point in time.

This state represents the values of the motors at a point in time. These instructions are played back at the rate polled to send the same commands the operator did.

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 74 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 218 of file autonrecorder.c.

4.1.4.3   void playbackAuton (   )

Replays autonomous based on loaded values in states array.

Definition at line 317 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 90 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 135 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
00011 #ifndef AUTONRECORDER_H
00012 #define AUTONRECORDER_H
00013
00017 #define AUTON_TIME 15
00018
00022 #define PROGSKILL_TIME 60
00023
00029 #define JOY_POLL_FREQ 50
00030
00034 #define MAX_AUTON_SLOTS 10
00035
00039 #define AUTON_FILENAME_MAX_LENGTH 8
00040
00044 #define AUTON_POT 1
00045
00049 #define AUTON_BUTTON 9
00050
00054 #define AUTON_POT_LOW 0
00055
00059 #define AUTON_POT_HIGH 4095
00060
00067 typedef struct joyState {
00071     signed char spd;
```

```
00072
00076     signed char turn;
00077
00081     signed char sht;
00082
00086     signed char intk;
00087
00091     signed char trans;
00092
00096     signed char dep;
00097 } joyState;
00098
00103 extern joyState states[AUTON_TIME*JOY_POLL_FREQ];
00104
00108 extern int autonLoaded;
00109
00114 extern int progSkills;
00115
00119 void initAutonRecorder();
00120
00124 void recordAuton();
00125
00129 void saveAuton();
00130
00134 void loadAuton();
00135
00139 void playbackAuton();
00140
00141 #endif
00142
```

4.3 include/autonroutines.h File Reference

Header file for hard-coded autonomous routines.

Macros

- #define CLOSE_GOAL_ANGLE 15
- #define DISTANCE_TO_OTHER_SIDE 100

Functions

- void runHardCodedProgrammingSkills ()

4.3.1 Detailed Description

Header file for hard-coded autonomous routines.

This file contains function declarations and definitions for use in the hard-coded autonomous routines. These functions can be called from other files to run hard-coded autonomous routines.

Definition in file autonroutines.h.

4.3.2   Macro Definition Documentation

4.3.2.1   #define CLOSE_GOAL_ANGLE 15

Angle that the robot must be from the vertical to shoot into the close goal while still being able to turn without hitting the wall.

Definition at line 14 of file autonroutines.h.

4.3.2.2   #define DISTANCE_TO_OTHER_SIDE 100

Distance that the robot must travel to reach the other tile to shoot preloads.

Definition at line 19 of file autonroutines.h.

4.3.3   Function Documentation

4.3.3.1   void runHardCodedProgrammingSkills (   )

Runs a pre-written programming skills routine using sensors rather than the autonomous recorder.

Runs a programming skills routine using sensors rather than the autonomous recorder. Starts in the left side of the field shooting into the closer goal.

Definition at line 13 of file autonroutines.c.

4.4   autonroutines.h

```
00001
00008 #ifndef AUTONROUTINES_H
00009 #define AUTONROUTINES_H
00010
00014 #define CLOSE_GOAL_ANGLE 15
00015
00019 #define DISTANCE_TO_OTHER_SIDE 100
00020
00024 void runHardCodedProgrammingSkills();
00025
00026 #endif
00027
```

4.5   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))

4.5.1 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.

4.5.2 Macro Definition Documentation

4.5.2.1 #define bitClear(value, bit) ((value) &= $\sim$(1UL $<<$ (bit)))

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

Parameters

value	the variable to clear in
bit	the bit number to clear (0 being the rightmost)

Definition at line 35 of file bitwise.h.

4.5.2.2 #define bitRead(value, bit) (((value) $>>$ (bit)) & 0x01)

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

Parameters

value	the variable to read from
bit	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.

4.5.2.3 #define bitSet(value, bit) ((value) $|=$ (1UL $<<$ (bit)))

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

Parameters

value	the variable to set in
bit	the bit number to set (0 being the rightmost)

Definition at line 27 of file bitwise.h.

4.5.2.4   #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

value	the variable to write in
bit	the bit number to set (0 being the rightmost)
bitvalue	the value to write (1 or 0)

Definition at line 44 of file bitwise.h.

4.6   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
00047
```

4.7   include/constants.h File Reference

Header file for mathematical constants.

Macros

- #define MATH_PI 3.14159265358979323846264338327 9
- #define MATH_E 2.718281828459045
- #define PI 3.14159265358979323846264338327 95
- #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

4.7.1 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.

4.7.2 Macro Definition Documentation

4.7.2.1 #define DEG_TO_RAD 0.017453292519943295769236907684886

Conversion factor from degrees to radians.

Definition at line 44 of file constants.h.

4.7.2.2 #define HALF_PI 1.5707963267948966192313216916398

Half the value of pi.

Definition at line 29 of file constants.h.

4.7.2.3 #define MATH_E 2.718281828459045

The mathematical constant e (Euler's Number).

Definition at line 19 of file constants.h.

4.7.2.4 #define MATH_PI 3.14159265358979323846264338327950

The mathematical constant pi.

Definition at line 14 of file constants.h.

4.7.2.5 #define PI 3.14159265358979323846426433832795

The mathematical constant pi.

Definition at line 24 of file constants.h.

4.7.2.6 #define RAD_TO_DEG 57.295779513082320876798154814105

Conversion factor from radians to degrees.

Definition at line 49 of file constants.h.

4.7.2.7 #define ROTATION_DEG 360

Amount of degrees in a circle.

Definition at line 54 of file constants.h.

4.7.2.8 #define ROTATION_RAD TWO_PI

Amount of radians in a circle.

Definition at line 59 of file constants.h.

4.7.2.9 #define TAU 6.283185307179586476925286766559

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

Definition at line 39 of file constants.h.

4.7.2.10 #define TWO_PI 6.283185307179586476925286766559

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

Definition at line 34 of file constants.h.

4.8 constants.h

```
00001
00008 #ifndef CONSTANTS_H_
00009 #define CONSTANTS_H_
00010
00014 #define MATH_PI 3.141592653589793238462643383279
00015
00019 #define MATH_E 2.718281828459045
00020
00024 #define PI 3.14159265358979323846426433832795
00025
00029 #define HALF_PI 1.5707963267948966192313216916398
00030
00034 #define TWO_PI 6.283185307179586476925286766559
00035
00039 #define TAU 6.283185307179586476925286766559
00040
00044 #define DEG_TO_RAD 0.017453292519943295769236907684886
00045
00049 #define RAD_TO_DEG 57.295779513082320876798154814105
00050
00054 #define ROTATION_DEG 360
00055
00059 #define ROTATION_RAD TWO_PI
00060
00061 #endif
00062
```

4.9 include/friendly.h File Reference

Header file for human-readable definitions.

Macros

- #define PRESSED LOW
- #define UNPRESSED HIGH
- #define UNRELEASED LOW
- #define RELEASED HIGH

4.9.1 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.

4.9.2 Macro Definition Documentation

4.9.2.1 #define PRESSED LOW

More readable definition for button pressed/unreleased state.

Definition at line 15 of file friendly.h.

4.9.2.2 #define RELEASED HIGH

More readable definition for button released/unpressed state.

Definition at line 30 of file friendly.h.

4.9.2.3 #define UNPRESSED HIGH

More readable definition for button unpressed/released state.

Definition at line 20 of file friendly.h.

4.9.2.4 #define UNRELEASED LOW

More readable definition for button unreleased/pressed state.

Definition at line 25 of file friendly.h.

4.10   friendly.h

```
00001
00009 #ifndef FRIENDLY_H_
00010 #define FRIENDLY_H_
00011
00015 #define PRESSED LOW
00016
00020 #define UNPRESSED HIGH
00021
00025 #define UNRELEASED LOW
00026
00030 #define RELEASED HIGH
00031
00032 #endif
00033
```

4.11   include/lcddiag.h File Reference

Header file for LCD diagnostic menu functions and definitions.

Classes

- struct MotorGroup

  Represents a logical motor grouping, to be used when testing motors.

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

  Represents a logical motor grouping, to be used when testing motors.

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

4.11.1 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.

4.11.2 Macro Definition Documentation

4.11.2.1 #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.

4.11.2.2   #define MENU_AUTON 5

Menu ID number of the autonomous recorder status indicator.

Definition at line 108 of file lcddiag.h.

4.11.2.3   #define MENU_BACKLIGHT 6

Menu ID number of the backlight toggle.

Definition at line 113 of file lcddiag.h.

4.11.2.4   #define MENU_BATTERY 2

Menu ID number of the battery voltage readout.

Definition at line 93 of file lcddiag.h.

4.11.2.5   #define MENU_CONNECTION 3

Menu ID number of the joystick connection state indicator.

Definition at line 98 of file lcddiag.h.

4.11.2.6   #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.

4.11.2.7   #define MENU_MOTOR 0

Menu ID number of the motor testing menu.

Definition at line 83 of file lcddiag.h.

4.11.2.8   #define MENU_MOTOR_MGMT 1

Menu ID number of the motor group manager.

Definition at line 88 of file lcddiag.h.

4.11.2.9   #define MENU_ROBOT 4

Menu ID number of the robot sensory state indicator.

Definition at line 103 of file lcddiag.h.

4.11.2.10 #define MENU_SCREENSAVER 7

Menu ID number of the LCD message screensaver.

Definition at line 118 of file lcddiag.h.

4.11.3 Typedef Documentation

4.11.3.1 typedef struct MotorGroup MotorGroup

Represents a logical motor grouping, to be used when testing motors.

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

4.11.4 Function Documentation

4.11.4.1 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

ignore	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

ignore	does nothing - required by task definition

Definition at line 1298 of file lcddiag.c.

4.11.4.2 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.

4.11.4.3 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.

4.11.4.4   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

btn	the button to check

Returns

  true if pressed, false if not

Definition at line 44 of file lcddiag.h.

4.11.4.5   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

dest	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.

4.11.5   Variable Documentation

4.11.5.1 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.

4.11.5.2 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.

4.11.5.3 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.

4.11.5.4 char menuChoices[LCD_MENU_COUNT][LCD_MESSAGE_MAX_LENGTH+1]

Stores the top-level menu names.

Definition at line 60 of file lcddiag.c.

4.11.5.5 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.

4.12 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
00143 typedef struct MotorGroup {
00149      bool motor[11];
00150
00157      char name[LCD_MESSAGE_MAX_LENGTH+1];
00158 } MotorGroup;
00159
00165 extern MotorGroup *groups;
00166
00171 extern int numgroups;
00172
00177 extern bool disableOpControl;
00178
00183 void initGroups();
00184
00194 char* typeString(char* dest);
00195
00203 void formatLCDDisplay(void* ignore);
00204
00205 #endif
00206
```

4.13   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 48
- #define LCD_MESSAGE_MAX_LENGTH 16

Functions

- void randlcdmsg (FILE ∗lcdport, int line)
- void screensaver (FILE ∗lcdport)

Variables

- char ∗ lcdmsg [ ]

4.13.1 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.

4.13.2 Macro Definition Documentation

4.13.2.1 #define LCD_750C_TITLE " $$$ 750C $$$ "

Defines title string for LCD to display.

Definition at line 21 of file lcdmsg.h.

4.13.2.2 #define LCD_MESSAGE_COUNT 48

Defines the amount of LCD messages in the master list.

Definition at line 26 of file lcdmsg.h.

4.13.2.3 #define LCD_MESSAGE_MAX_LENGTH 16

Defines the max length for LCD messages.

Definition at line 31 of file lcdmsg.h.

4.13.2.4 #define LCD_PORT uart1

Defines the port the LCD is plugged into.

Definition at line 16 of file lcdmsg.h.

4.13.3 Function Documentation

4.13.3.1 void randlcdmsg (FILE ∗ *lcdport,* int *line*)

Displays a random LCD message from the master list.

Parameters

lcdport	the port the LCD is connected to
line	the line to display the message on

Definition at line 71 of file lcdmsg.c.

4.13.3.2   void screensaver (FILE ∗ *lcdport*)

Formats the LCD by displaying 750C title and message.

Parameters

lcdport	the port the LCD is connected to

Definition at line 90 of file lcdmsg.c.

4.13.4   Variable Documentation

4.13.4.1   char∗ lcdmsg[ ]

Master list of all LCD messages.

Definition at line 14 of file lcdmsg.c.

4.14   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 48
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
00054
```

4.15   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))

4.15.1 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.

4.15.2 Macro Definition Documentation

4.15.2.1 #define abs(x) ((x)>0?(x):-(x))

Returns the absolute value of the input value.

Parameters

x	the input value

Returns

the absolute value of x

Definition at line 57 of file macros.h.

4.15.2.2 #define constrain($amt,$ $low,$ $high$) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))

Constrains a value to a set of boundaries.

Parameters

amt	the value to constrain
low	the lower bound
$high$	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.

4.15.2.3 #define degrees(*rad*) ((rad)∗RAD_TO_DEG)

Converts an angle measure in degrees to radians.

Parameters

rad	the angle measure in radians

Returns

     the angle measure in degrees

Definition at line 104 of file macros.h.

4.15.2.4 #define max(*a, b*) ((a)>(b)?(a):(b))

Returns the maximum of the two values.

Parameters

a	the first input value
b	the second input value

Returns

     the greater of the two values

Definition at line 38 of file macros.h.

4.15.2.5 #define MAX(*a, b*) ((a)>(b)?(a):(b))

Returns the maximum of the two values.

Parameters

a	the first input value
b	the second input value

Returns

the greater of the two values

Definition at line 48 of file macros.h.

4.15.2.6 #define min(a, b) ((a)<(b)?(a):(b))

Returns the minimum of the two values.

Parameters

a	the first input value
b	the second input value

Returns

the lesser of the two values

Definition at line 18 of file macros.h.

4.15.2.7 #define MIN(a, b) ((a)<(b)?(a):(b))

Returns the minimum of the two values.

Parameters

a	the first input value
b	the second input value

Returns

the lesser of the two values

Definition at line 28 of file macros.h.

4.15.2.8 #define radians(deg) ((deg)∗DEG_TO_RAD)

Converts an angle measure in degrees to radians.

Parameters

deg	the angle measure in degrees

Returns

the angle measure in radians

Definition at line 95 of file macros.h.

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

Rounds a value to the nearest integer.

Parameters

x	the value to round

Returns

the rounded value

Definition at line 77 of file macros.h.

4.15.2.10 #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.

4.15.2.11 #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.

4.15.2.12 #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.

4.16 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
00125
```

4.17 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 <robot.h>
#include <lcdmsg.h>
#include <lcddiag.h>
#include <autonroutines.h>
#include <autonrecorder.h>
#include <opcontrol.h>
```

Functions

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

4.17.1   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 EXP↩
RESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERC↩
HANTABILITY 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.

4.17.2 Function Documentation

4.17.2.1 void autonomous ( )

C standard integer type header. C standard string function header. PROS main API header. Mathematical constant defi-
nitions. 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.
Robot physical constant definitions and function declarations. LCD definitions and function declarations. LCD diagnos-
tics menu definitions and function declarations. Hard-coded autonomous routine 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.

4.17.2.2 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.

4.17.2.3 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.

4.17.2.4 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.

4.18 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 <robot.h>
00093
00097 #include <lcdmsg.h>
00098
00102 #include <lcddiag.h>
00103
00107 #include <autonroutines.h>
```

```
00108
00112 #include <autonrecorder.h>
00113
00117 #include <opcontrol.h>
00118
00119 // Allow usage of this file in C++ programs
00120 #ifdef __cplusplus
00121 extern "C" {
00122 #endif
00123
00124 // A function prototype looks exactly like its declaration, but with a semicolon instead of
00125 // actual code. If a function does not match a prototype, compile errors will occur.
00126
00127 // Prototypes for initialization, operator control and autonomous
00128
00143 void autonomous();
00152 void initializeIO();
00166 void initialize();
00184 void operatorControl();
00185
00186 // End C++ export structure
00187 #ifdef __cplusplus
00188 }
00189 #endif
00190
00191 #endif
00192
```

4.19   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 5
- #define RIGHT_MOTOR_TOP 3
- #define RIGHT_MOTOR_BOT 4
- #define SHOOTER_HAS_THREE_MOTORS
- #define NAUTILUS_SHOOTER_MOTOR_LEFT 6
- #define NAUTILUS_SHOOTER_MOTOR_RIGHT 7
- #define NAUTILUS_SHOOTER_MOTOR_CENTER 8
- #define INTAKE_ROLLER_MOTOR 10
- #define ROBOT_HAS_LIFT_DEPLOY_MOTOR 1
- #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)

4.19.1   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.

4.19.2   Macro Definition Documentation

4.19.2.1   #define INTAKE_ROLLER_MOTOR 10

Defines motor ports for the intake mechanism.

Definition at line 107 of file motors.h.

4.19.2.2   #define LEFT_MOTOR_BOT 5

Definition at line 63 of file motors.h.

4.19.2.3   #define LEFT_MOTOR_TOP 2

Defines motor ports for the left side of the drivetrain.

Definition at line 62 of file motors.h.

4.19.2.4   #define LIFT_DEPLOY 9

Definition at line 121 of file motors.h.

4.19.2.5   #define MOTOR_MAX 127

Defines maximum motor speed value.

Definition at line 21 of file motors.h.

4.19.2.6   #define MOTOR_MIN -127

Defines motor minimum speed value.

Definition at line 26 of file motors.h.

4.19.2.7   #define NAUTILUS_SHOOTER_MOTOR_CENTER 8

Definition at line 91 of file motors.h.

4.19.2.8   #define NAUTILUS_SHOOTER_MOTOR_LEFT 6

Definition at line 89 of file motors.h.

4.19.2.9   #define NAUTILUS_SHOOTER_MOTOR_RIGHT 7

Definition at line 90 of file motors.h.

4.19.2.10   #define RIGHT_MOTOR_BOT 4

Definition at line 69 of file motors.h.

4.19.2.11   #define RIGHT_MOTOR_TOP 3

Defines motor ports for the right side of the drivetrain.

Definition at line 68 of file motors.h.

4.19.2.12   #define ROBOT_HAS_LIFT_DEPLOY_MOTOR 1

Definition at line 118 of file motors.h.

4.19.2.13   #define SHOOTER_HAS_THREE_MOTORS

Defines motor ports for the nautilus gear shooting mechanism.

Definition at line 87 of file motors.h.

4.19.2.14   #define TRANSMISSION_MOTOR 1

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

Definition at line 31 of file motors.h.

4.19.3   Function Documentation

4.19.3.1   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

gear	the gear to change to

Definition at line 55 of file motors.h.

4.19.3.2 void deploy (int *spd*) `[inline]`

Definition at line 123 of file motors.h.

4.19.3.3 void intake (int *spd*) `[inline]`

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

Parameters

spd	the speed to set the intake motors

Definition at line 114 of file motors.h.

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

Moves the robot by setting the drive motor values.

Parameters

spd	the forward/backward speed value
turn	the turning speed value

Definition at line 77 of file motors.h.

4.19.3.5 void shoot (int *spd*) `[inline]`

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

Parameters

spd	the speed to set the shooter motors

Definition at line 98 of file motors.h.

4.19.3.6 void transmission (int *spd*) `[inline]`

Runs the transmission motor by setting the motor value.

Parameters

spd	the speed to run the transmission motor

Definition at line 38 of file motors.h.

4.19.3.7 void transmissionSetPos (void ∗ pos)

Sets the position of the transmission.

Parameters

pos	the position to set the transmission to.

Definition at line 19 of file motors.c.

4.20 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,
     TASK_PRIORITY_DEFAULT);
00057 }
00058
00062 #define LEFT_MOTOR_TOP 2
00063 #define LEFT_MOTOR_BOT 5
00064
00068 #define RIGHT_MOTOR_TOP 3
00069 #define RIGHT_MOTOR_BOT 4
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 6
00090 #define NAUTILUS_SHOOTER_MOTOR_RIGHT 7
00091 #define NAUTILUS_SHOOTER_MOTOR_CENTER 8
00092
00098 inline void shoot(int spd){
00099     motorSet(NAUTILUS_SHOOTER_MOTOR_LEFT, -spd);
00100     motorSet(NAUTILUS_SHOOTER_MOTOR_RIGHT, spd);
00101     motorSet(NAUTILUS_SHOOTER_MOTOR_CENTER, spd);
00102 }
00103
```

```
00107 #define INTAKE_ROLLER_MOTOR 10
00108
00114 inline void intake(int spd){
00115     motorSet(INTAKE_ROLLER_MOTOR, spd);
00116 }
00117
00118 #define ROBOT_HAS_LIFT_DEPLOY_MOTOR 1
00119
00120 #ifdef ROBOT_HAS_LIFT_DEPLOY_MOTOR
00121 #define LIFT_DEPLOY 9
00122
00123 inline void deploy(int spd){
00124     motorSet(LIFT_DEPLOY, spd);
00125 }
00126 #endif
00127
00128 #ifndef ROBOT_HAS_LIFT_DEPLOY_MOTOR
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
00145
```

4.21   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

4.21.1   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.

4.21.2 **Function Documentation**

4.21.2.1 **void moveRobot ( )**

Moves the robot based on the motor state variables.

Definition at line 128 of file opcontrol.c.

4.21.2.2 **void recordJoyInfo ( )**

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

Definition at line 81 of file opcontrol.c.

4.21.3 **Variable Documentation**

4.21.3.1 **int dep**

Speed of the lift deployment motor.

Definition at line 65 of file opcontrol.c.

4.21.3.2 **int intk**

Speed of the intake motor.

Speed of the intake motors.

Definition at line 55 of file opcontrol.c.

4.21.3.3 **int sht**

Speed of the shooter motors.

Definition at line 50 of file opcontrol.c.

4.21.3.4 **int spd**

Forward/backward speed of the drive motors.

Definition at line 40 of file opcontrol.c.

4.21.3.5 **int trans**

Speed of the transmission motor.

Speed of the transmission motors.

Definition at line 60 of file opcontrol.c.

4.21.3.6   int turn

Turning speed of the drive motors.

Definition at line 45 of file opcontrol.c.

4.22   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
```

4.23   include/robot.h File Reference

File for robot constant definitions.

Macros

- #define DRIVE_WHEELBASE 16
- #define DRIVE_DIA 3
- #define DRIVE_GEARRATIO 1

4.23.1   Detailed Description

File for robot constant definitions.

This file contains definitions for constants regarding the robot. These definitions pertain to physical properties of the robot.

Definition in file robot.h.

4.23.2 Macro Definition Documentation

4.23.2.1 #define DRIVE_DIA 3

Defines the diameter of the robot's wheels, in inches.

Definition at line 20 of file robot.h.

4.23.2.2 #define DRIVE_GEARRATIO 1

Defines the drive's gear ratio.

Definition at line 25 of file robot.h.

4.23.2.3 #define DRIVE_WHEELBASE 16

Defines the robot's wheelbase, in inches.

This is equivalent to the distance between the midpoints of the wheels.

Definition at line 15 of file robot.h.

4.24 robot.h

```
00001
00007 #ifndef ROBOT_H_
00008 #define ROBOT_H_
00009
00015 #define DRIVE_WHEELBASE 16
00016
00020 #define DRIVE_DIA 3
00021
00025 #define DRIVE_GEARRATIO 1
00026
00027 #endif
```

4.25 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
- #define GYRO_P 10
- #define SHOOTER_LIMIT 10
- #define ULTRASONIC_ECHO_PORT 11
- #define ULTRASONIC_PING_PORT 12
- #define SONG_COUNT 17

Functions

- void clearDriveEncoders ()
- void lturn (int bodydegs)
- void rturn (int bodydegs)
- void goForward (int inches)
- void goBackward (int inches)
- unsigned int powerLevelExpander ()
- void playSpeaker (void ∗ignore)
- char ∗ randsong ()

Variables

- Encoder leftenc
- Encoder rightenc
- Gyro gyro
- Ultrasonic sonar
- char ∗ songs [ ]
- TaskHandle speakerTask

4.25.1   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.

4.25.2   Macro Definition Documentation

4.25.2.1   #define BATT_BKUP 1

Battery ID number of the robot's backup battery.

Definition at line 126 of file sensors.h.

4.25.2.2   #define BATT_MAIN 0

Battery ID number of the robot's main battery.

Definition at line 121 of file sensors.h.

4.25.2.3   #define BATT_PEXP 2

Battery ID number of the power expander's battery.

Definition at line 131 of file sensors.h.

4.25.2.4   #define GEAR_DRIVE 1860

Defines potentiometer values for drive gearing.

Definition at line 84 of file sensors.h.

4.25.2.5   #define GEAR_LIFT 4095

Defines potentiometer values for lift gearing.

Definition at line 89 of file sensors.h.

4.25.2.6   #define GYRO_NET_TARGET 0

Defines gyroscope target angle for the opposite corner net.

Definition at line 146 of file sensors.h.

4.25.2.7   #define GYRO_P 10

Defines the proportional error-correction term for the gyroscope alignment velocity control loop.

Definition at line 151 of file sensors.h.

4.25.2.8   #define GYRO_PORT 4

Defines the port for the gyroscope.

Definition at line 136 of file sensors.h.

4.25.2.9   #define GYRO_SENSITIVITY 0

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

Definition at line 141 of file sensors.h.

4.25.2.10   #define LEFT_ENC_BOT 2

Definition at line 22 of file sensors.h.

4.25.2.11   #define LEFT_ENC_TOP 1

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

Definition at line 21 of file sensors.h.

4.25.2.12   #define NUM_BATTS 3

The number of batteries present on the robot.

Definition at line 116 of file sensors.h.

4.25.2.13   #define POWER_EXPANDER_STATUS 3

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

Definition at line 95 of file sensors.h.

4.25.2.14   #define POWER_EXPANDER_VOLTAGE_DIVISOR 280

Defines power expander divisor. This varies by hardware revision. This value is for hardware revision A2. The sensor's value is divided by this to get the battery voltage.

Definition at line 102 of file sensors.h.

4.25.2.15   #define RIGHT_ENC_BOT 4

Definition at line 33 of file sensors.h.

4.25.2.16   #define RIGHT_ENC_TOP 3

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

Definition at line 32 of file sensors.h.

4.25.2.17   #define SHOOTER_LIMIT 10

Defines the port for the limit switch that is triggered when the shooter fires.

Definition at line 161 of file sensors.h.

4.25.2.18   #define SONG_COUNT 17

Defines the amount of songs in the master list.

Definition at line 181 of file sensors.h.

4.25.2.19   #define TRANSMISSION_POT 2

Defines the transmission potentiometer for position determination.

Definition at line 79 of file sensors.h.

4.25.2.20   #define ULTRASONIC_ECHO_PORT 11

Defines the port for the ultrasonic echo wire (orange).

Definition at line 166 of file sensors.h.

4.25.2.21   #define ULTRASONIC_PING_PORT 12

Defines the port for the ultrasonic ping wire (yellow).

Definition at line 171 of file sensors.h.

4.25.3   Function Documentation

4.25.3.1   void clearDriveEncoders ( ) `[inline]`

Clears the drive encoders by resetting their value to zero.

Definition at line 43 of file sensors.h.

4.25.3.2   void goBackward (int *inches*)

Moves the robot backward a specified distance.

Parameters

inches	the amount of inches to move backward

Definition at line 71 of file sensors.c.

4.25.3.3 void goForward (int *inches*)

Moves the robot forward a specified distance.

Parameters

inches	the amount of inches to move forward

Definition at line 55 of file sensors.c.

4.25.3.4 void lturn (int *bodydegs*)

Turns the robot left to a specified angle.

Parameters

bodydegs	the amount of degrees to turn the robot

Definition at line 40 of file sensors.c.

4.25.3.5 void playSpeaker (void ∗ *ignore*)

Plays a song over the speaker. This task plays a random song from the array of songs.

Parameters

ignore	does nothing - required by task definition

Definition at line 131 of file sensors.c.

4.25.3.6 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 109 of file sensors.h.

4.25.3.7 char∗ randsong (   )

Returns a random song from the master list.

Returns

a pointer to the song

Definition at line 113 of file sensors.c.

4.25.3.8 void rturn (int *bodydegs*)

Turns the robot right to a specified angle.

Parameters

bodydegs	the amount of degrees to turn the robot

Definition at line 25 of file sensors.c.

4.25.4 Variable Documentation

4.25.4.1 Gyro gyro

Object representing the gyroscope.

Definition at line 78 of file sensors.c.

4.25.4.2 Encoder leftenc

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

Definition at line 18 of file sensors.c.

4.25.4.3 Encoder rightenc

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

Definition at line 23 of file sensors.c.

4.25.4.4 Ultrasonic sonar

Object representing the ultrasonic sensor.

Definition at line 83 of file sensors.c.

4.25.4.5 char∗ songs[ ]

Master list of all songs.

Definition at line 88 of file sensors.c.

4.25.4.6 TaskHandle speakerTask

Object representing the speaker song task. The speaker song task runs in a separate thread from the operator control code. This prevents it from blocking the driving code from executing.

Definition at line 193 of file sensors.h.

4.26 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 inline void clearDriveEncoders(){
00044     encoderReset(leftenc);
00045     encoderReset(rightenc);
00046 }
00047
00053 void lturn(int bodydegs);
00054
00060 void rturn(int bodydegs);
00061
00067 void goForward(int inches);
00068
00074 void goBackward(int inches);
00075
00079 #define TRANSMISSION_POT 2
00080
00084 #define GEAR_DRIVE 1860
00085
00089 #define GEAR_LIFT 4095
00090
00095 #define POWER_EXPANDER_STATUS 3
00096
00102 #define POWER_EXPANDER_VOLTAGE_DIVISOR 280 //Hardware revision A2
00103
00109 inline unsigned int powerLevelExpander(){
00110     return analogRead(POWER_EXPANDER_STATUS)*1000/
    POWER_EXPANDER_VOLTAGE_DIVISOR;
00111 }
00112
00116 #define NUM_BATTS 3
00117
00121 #define BATT_MAIN 0
00122
00126 #define BATT_BKUP 1
00127
00131 #define BATT_PEXP 2
00132
00136 #define GYRO_PORT 4
00137
00141 #define GYRO_SENSITIVITY 0
00142
00146 #define GYRO_NET_TARGET 0
```

```
00147
00151 #define GYRO_P 10
00152
00156 extern Gyro gyro;
00157
00161 #define SHOOTER_LIMIT 10
00162
00166 #define ULTRASONIC_ECHO_PORT 11
00167
00171 #define ULTRASONIC_PING_PORT 12
00172
00176 extern Ultrasonic sonar;
00177
00181 #define SONG_COUNT 17
00182
00186 extern char* songs[];
00187
00193 TaskHandle speakerTask;
00194
00201 void playSpeaker(void *ignore);
00202
00208 char* randsong();
00209
00210 #endif
00211
```

4.27 src/auto.c File Reference

File for autonomous code.

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

Functions

- void autonomous ()

4.27.1 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 EXP↩
RESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERC↩
HANTABILITY 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.

4.27.2 Function Documentation

4.27.2.1 void autonomous ( )

C standard integer type header. C standard string function header. PROS main API header. Mathematical constant defi-
nitions. 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.
Robot physical constant definitions and function declarations. LCD definitions and function declarations. LCD diagnos-
tics menu definitions and function declarations. Hard-coded autonomous routine 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.

4.28 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 }
00054
```

4.29 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

4.29.1 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.

4.29.2 Function Documentation

4.29.2.1 void initAutonRecorder ( )

Initializes autonomous recorder by setting states array to zero.

Definition at line 74 of file autonrecorder.c.

4.29.2.2 void loadAuton ( )

Loads autonomous file contents into states array.

Definition at line 218 of file autonrecorder.c.

4.29.2.3 void playbackAuton ( )

Replays autonomous based on loaded values in states array.

Definition at line 317 of file autonrecorder.c.

4.29.2.4 void recordAuton ( )

Records driver joystick values into states array.

Definition at line 90 of file autonrecorder.c.

4.29.2.5 void saveAuton ( )

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

Definition at line 135 of file autonrecorder.c.

4.29.2.6 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.

4.29.3 Variable Documentation

4.29.3.1 int autonLoaded

Slot number of currently loaded autonomous routine.

Definition at line 24 of file autonrecorder.c.

4.29.3.2 int progSkills

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

Definition at line 29 of file autonrecorder.c.

4.29.3.3 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.

4.30 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+3);
00041         if(val > MAX_AUTON_SLOTS+2){
00042             val = MAX_AUTON_SLOTS+2;
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 if (val == MAX_AUTON_SLOTS+2) {
00049             lcdSetText(LCD_PORT, 2, "Hardcoded Skills");
00050         } else {
00051             char filename[AUTON_FILENAME_MAX_LENGTH];
00052             snprintf(filename, sizeof(filename)/sizeof(char), "a%d", val);
00053             FILE* autonFile = fopen(filename, "r");
00054             if(autonFile == NULL){
00055                 lcdPrint(LCD_PORT, 2, "Slot: %d (EMPTY)", val);
00056             } else {
00057                 char name[LCD_MESSAGE_MAX_LENGTH+1];
00058                 memset(name, 0, sizeof(name));
00059                 fread(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
00060                 lcdSetText(LCD_PORT, 2, name);
00061                 fclose(autonFile);
00062             }
00063         }
00064         done = (digitalRead(AUTON_BUTTON) == PRESSED);
00065         delay(20);
00066     } while(!done);
00067     printf("Selected autonomous: %d\n", val);
00068     return val;
00069 }
00070
00074 void initAutonRecorder() {
00075     printf("Beginning initialization of autonomous recorder...\n");
00076     lcdClear(LCD_PORT);
00077     lcdSetText(LCD_PORT, 1, "Init recorder...");
00078     lcdSetText(LCD_PORT, 2, "");
00079     memset(states, 0, sizeof(*states));
00080     printf("Completed initialization of autonomous recorder.\n");
00081     lcdSetText(LCD_PORT, 1, "Init-ed recorder!");
00082     lcdSetText(LCD_PORT, 2, "");
00083     autonLoaded = -1;
00084     progSkills = 0;
00085 }
00086
00090 void recordAuton() {
00091     lcdClear(LCD_PORT);
00092     for(int i = 3; i > 0; i--){
00093         lcdSetBacklight(LCD_PORT, true);
00094         printf("Beginning autonomous recording in %d...\n", i);
00095         lcdSetText(LCD_PORT, 1, "Recording auton");
00096         lcdPrint(LCD_PORT, 2, "in %d...", i);
00097         delay(1000);
00098     }
00099     printf("Ready to begin autonomous recording.\n");
00100     lcdSetText(LCD_PORT, 1, "Recording auton...");
00101     lcdSetText(LCD_PORT, 2, "");
00102     bool lightState = false;
00103     for (int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00104         printf("Recording state %d...\n", i);
00105         lcdSetBacklight(LCD_PORT, lightState);
00106         lightState = !lightState;
00107         recordJoyInfo();
00108         states[i].spd = spd;
00109         states[i].turn = turn;
```

```
00110              states[i].sht = sht;
00111              states[i].intk = intk;
00112              states[i].trans = trans;
00113              states[i].dep = dep;
00114              if (joystickGetDigital(1, 7, JOY_UP)) {
00115                  printf("Autonomous recording manually cancelled.\n");
00116                  lcdSetText(LCD_PORT, 1, "Cancelled record.");
00117                  lcdSetText(LCD_PORT, 2, "");
00118                  memset(states + i + 1, 0, sizeof(joyState) * (AUTON_TIME * JOY_POLL_FREQ - i
       - 1));
00119                  i = AUTON_TIME * JOY_POLL_FREQ;
00120              }
00121              moveRobot();
00122              delay(1000 / JOY_POLL_FREQ);
00123          }
00124      printf("Completed autonomous recording.\n");
00125      lcdSetText(LCD_PORT, 1, "Recorded auton!");
00126      lcdSetText(LCD_PORT, 2, "");
00127      motorStopAll();
00128      delay(1000);
00129      autonLoaded = 0;
00130 }
00131
00135 void saveAuton() {
00136      printf("Waiting for file selection...\n");
00137      lcdClear(LCD_PORT);
00138      lcdSetText(LCD_PORT, 1, "Save to?");
00139      lcdSetText(LCD_PORT, 2, "");
00140      int autonSlot;
00141      if(progSkills == 0) {
00142          autonSlot = selectAuton();
00143      } else {
00144          printf("Currently in the middle of a programming skills run.\n");
00145          autonSlot = MAX_AUTON_SLOTS + 1;
00146      }
00147      char name[LCD_MESSAGE_MAX_LENGTH+1];
00148      memset(name, 0, sizeof(name));
00149      if(autonSlot == 0) {
00150          printf("Not saving this autonomous!\n");
00151          return;
00152      } else if(autonSlot != MAX_AUTON_SLOTS+1) {
00153          typeString(name);
00154      }
00155      lcdSetText(LCD_PORT, 1, "Saving auton...");
00156      char filename[AUTON_FILENAME_MAX_LENGTH];
00157      if(autonSlot != MAX_AUTON_SLOTS + 1) {
00158          printf("Not doing programming skills, recording to slot %d.\n",autonSlot);
00159          snprintf(filename, sizeof(filename)/sizeof(char), "a%d", autonSlot);
00160          //lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00161          lcdPrint(LCD_PORT, 2, "%s", name);
00162      } else {
00163          printf("Doing programming skills, recording to section %d.\n",
       progSkills);
00164          snprintf(filename, sizeof(filename)/sizeof(char), "p%d", progSkills);
00165          lcdPrint(LCD_PORT, 2, "Skills Part: %d", progSkills+1);
00166      }
00167      printf("Saving to file %s...\n",filename);
00168      FILE *autonFile = fopen(filename, "w");
00169      if (autonFile == NULL) {
00170          printf("Error saving autonomous in file %s!\n", filename);
00171          lcdSetText(LCD_PORT, 1, "Error saving!");
00172          if(autonSlot != MAX_AUTON_SLOTS + 1){
00173              printf("Not doing programming skills, error saving auton in slot %d!\n", autonSlot);
00174              lcdSetText(LCD_PORT, 1, "Error saving!");
00175              lcdPrint(LCD_PORT, 2,    "Slot: %d", autonSlot);
00176          } else {
00177              printf("Doing programming skills, error saving auton in section 0!\n");
00178              lcdSetText(LCD_PORT, 1, "Error saving!");
00179              lcdSetText(LCD_PORT, 2, "Prog. Skills");
00180          }
00181          delay(1000);
00182          return;
00183      }
00184      if(autonSlot != MAX_AUTON_SLOTS+1){
00185          fwrite(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
00186      }
00187      for (int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00188          printf("Recording state %d to file %s...\n", i, filename);
00189          signed char write[6] = {states[i].spd, states[i].turn, states[i].
       sht, states[i].intk, states[i].trans,
00190                                   states[i].dep};
```

```
00191            fwrite(write, sizeof(char), sizeof(write) / sizeof(char), autonFile);
00192            delay(10);
00193        }
00194        fclose(autonFile);
00195        printf("Completed saving autonomous to file %s.\n", filename);
00196        lcdSetText(LCD_PORT, 1, "Saved auton!");
00197        if(autonSlot != MAX_AUTON_SLOTS + 1) {
00198            printf("Not doing programming skills, recorded to slot %d.\n",autonSlot);
00199            lcdPrint(LCD_PORT, 2, "Slot: %d", autonSlot);
00200        } else {
00201            printf("Doing programming skills, recorded to section %d.\n", progSkills);
00202            lcdPrint(LCD_PORT, 2, "Skills Part: %d", progSkills+1);
00203        }
00204        delay(1000);
00205        if(autonSlot == MAX_AUTON_SLOTS + 1) {
00206            printf("Proceeding to next programming skills section (%d).\n", ++
     progSkills);
00207        }
00208        if(progSkills == PROGSKILL_TIME/AUTON_TIME) {
00209            printf("Finished recording programming skills (all parts).\n");
00210            progSkills = 0;
00211        }
00212        autonLoaded = autonSlot;
00213 }
00214
00218 void loadAuton() {
00219        lcdClear(LCD_PORT);
00220        bool done = false;
00221        int autonSlot;
00222        FILE* autonFile;
00223        char filename[AUTON_FILENAME_MAX_LENGTH];
00224        do {
00225            printf("Waiting for file selection...\n");
00226            lcdSetText(LCD_PORT, 1, "Load from?");
00227            lcdSetText(LCD_PORT, 2, "");
00228            autonSlot = selectAuton();
00229            if(autonSlot == 0) {
00230                printf("Not loading an autonomous!\n");
00231                lcdSetText(LCD_PORT, 1, "Not loading!");
00232                lcdSetText(LCD_PORT,   2, "");
00233                autonLoaded = 0;
00234                return;
00235            } else if(autonSlot == MAX_AUTON_SLOTS + 1){
00236                printf("Performing programming skills.\n");
00237                lcdSetText(LCD_PORT, 1, "Loading skills...");
00238                lcdPrint(LCD_PORT,   2, "Skills Part: 1");
00239                autonLoaded = MAX_AUTON_SLOTS + 1;
00240            } else if (autonSlot == MAX_AUTON_SLOTS + 2) {
00241                printf("Performing hard-coded programming skills.\n");
00242                lcdSetText(LCD_PORT, 1, "Loading skills...");
00243                lcdPrint(LCD_PORT,   2, "Hardcoded Skills");
00244                autonLoaded = MAX_AUTON_SLOTS + 2;
00245                return;
00246            } else if(autonSlot == autonLoaded) {
00247                printf("Autonomous %d is already loaded.\n", autonSlot);
00248                lcdSetText(LCD_PORT, 1, "Loaded auton!");
00249                lcdPrint(LCD_PORT,   2, "Slot: %d", autonSlot);
00250                return;
00251            }
00252            printf("Loading autonomous from slot %d...\n", autonSlot);
00253            lcdSetText(LCD_PORT, 1, "Loading auton...");
00254            if(autonSlot != MAX_AUTON_SLOTS + 1){
00255                lcdPrint(LCD_PORT, 2,   "Slot: %d", autonSlot);
00256            }
00257            if(autonSlot != MAX_AUTON_SLOTS + 1){
00258                printf("Not doing programming skills, loading slot %d\n", autonSlot);
00259                snprintf(filename, sizeof(filename)/sizeof(char), "a%d", autonSlot);
00260            } else {
00261                printf("Doing programming skills, loading section 0.\n");
00262                snprintf(filename, sizeof(filename)/sizeof(char), "p0");
00263            }
00264            printf("Loading from file %s...\n",filename);
00265            autonFile = fopen(filename, "r");
00266            if (autonFile == NULL) {
00267                printf("No autonomous was saved in file %s!\n", filename);
00268                lcdSetText(LCD_PORT, 1, "No auton saved!");
00269                if(autonSlot != MAX_AUTON_SLOTS + 1){
00270                    printf("Not doing programming skills, no auton in slot %d!\n", autonSlot);
00271                    lcdSetText(LCD_PORT, 1, "No auton saved!");
00272                    lcdPrint(LCD_PORT, 2,   "Slot: %d", autonSlot);
00273                } else {
```

```
00274                        printf("Doing programming skills, no auton in section 0!\n");
00275                        lcdSetText(LCD_PORT, 1, "No skills saved!");
00276                    }
00277                    delay(1000);
00278                } else {
00279                    done = true;
00280                }
00281            } while(!done);
00282        fseek(autonFile, 0, SEEK_SET);
00283        char name[LCD_MESSAGE_MAX_LENGTH+1];
00284        memset(name, 0, sizeof(name));
00285        if(autonSlot != MAX_AUTON_SLOTS + 1){
00286            fread(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
00287        }
00288        for (int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00289            printf("Loading state %d from file %s...\n", i, filename);
00290            char read[6] = {0, 0, 0, 0, 0, 0};
00291            fread(read, sizeof(char), sizeof(read) / sizeof(char), autonFile);
00292            states[i].spd = (signed char) read[0];
00293            states[i].turn = (signed char) read[1];
00294            states[i].sht = (signed char) read[2];
00295            states[i].intk = (signed char) read[3];
00296            states[i].trans = (signed char) read[4];
00297            states[i].dep = (signed char) read[5];
00298            delay(10);
00299        }
00300        fclose(autonFile);
00301        printf("Completed loading autonomous from file %s.\n", filename);
00302        lcdSetText(LCD_PORT, 1, "Loaded auton!");
00303        if(autonSlot != MAX_AUTON_SLOTS + 1){
00304            printf("Not doing programming skills, loaded from slot %d.\n", autonSlot);
00305            //lcdPrint(LCD_PORT,   2, "Slot: %d", autonSlot);
00306            lcdPrint(LCD_PORT, 2, "%s", name);
00307        } else {
00308            printf("Doing programming skills, loaded from section %d.\n", progSkills);
00309            lcdSetText(LCD_PORT, 2, "Skills Section: 1");
00310        }
00311        autonLoaded = autonSlot;
00312 }
00313
00317 void playbackAuton() { //must load autonomous first!
00318        if (autonLoaded == -1 /* nothing in memory */) {
00319            printf("No autonomous loaded, entering loadAuton()\n");
00320            loadAuton();
00321        }
00322        if(autonLoaded == 0) {
00323            printf("autonLoaded = 0, doing nothing.\n");
00324            return;
00325        } else if (autonLoaded == MAX_AUTON_SLOTS + 2) {
00326            runHardCodedProgrammingSkills();
00327            return;
00328        }
00329        printf("Beginning playback...\n");
00330        lcdSetText(LCD_PORT, 1, "Playing back...");
00331        lcdSetText(LCD_PORT, 2, "");
00332        lcdSetBacklight(LCD_PORT, true);
00333        int file=0;
00334        do{
00335            FILE* nextFile = NULL;
00336            lcdPrint(LCD_PORT, 2, "File: %d", file+1);
00337            char filename[AUTON_FILENAME_MAX_LENGTH];
00338            if(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
      PROGSKILL_TIME/AUTON_TIME - 1){
00339                printf("Next section: %d\n", file+1);
00340                snprintf(filename, sizeof(filename)/sizeof(char), "p%d", file+1);
00341                nextFile = fopen(filename, "r");
00342            }
00343            for(int i = 0; i < AUTON_TIME * JOY_POLL_FREQ; i++) {
00344                printf("Playing back state %d...\n", i);
00345                spd = states[i].spd;
00346                turn = states[i].turn;
00347                sht = states[i].sht;
00348                intk = states[i].intk;
00349                trans = states[i].trans;
00350                dep = states[i].dep;
00351                if (joystickGetDigital(1, 7, JOY_UP) && !isOnline()) {
00352                    printf("Playback manually cancelled.\n");
00353                    lcdSetText(LCD_PORT, 1, "Cancelled playback.");
00354                    lcdSetText(LCD_PORT, 2, "");
00355                    i = AUTON_TIME * JOY_POLL_FREQ;
00356                    file = PROGSKILL_TIME/AUTON_TIME;
```

```
00357               }
00358               moveRobot();
00359               if(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
        PROGSKILL_TIME/AUTON_TIME - 1){
00360                   printf("Loading state %d from file %s...\n", i, filename);
00361                   char read[6] = {0, 0, 0, 0, 0, 0};
00362                   fread(read, sizeof(char), sizeof(read) / sizeof(char), nextFile);
00363                   states[i].spd = (signed char) read[0];
00364                   states[i].turn = (signed char) read[1];
00365                   states[i].sht = (signed char) read[2];
00366                   states[i].intk = (signed char) read[3];
00367                   states[i].trans = (signed char) read[4];
00368                   states[i].dep = (signed char) read[5];
00369               }
00370               delay(1000 / JOY_POLL_FREQ);
00371           }
00372           if(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
        PROGSKILL_TIME/AUTON_TIME - 1){
00373               printf("Finished with section %d, closing file.\n", file+1);
00374               fclose(nextFile);
00375           }
00376           file++;
00377       } while(autonLoaded == MAX_AUTON_SLOTS + 1 && file <
        PROGSKILL_TIME/AUTON_TIME);
00378       motorStopAll();
00379       printf("Completed playback.\n");
00380       lcdSetText(LCD_PORT, 1, "Played back!");
00381       lcdSetText(LCD_PORT, 2, "");
00382       delay(1000);
00383 }
00384
```

4.31 src/autonroutines.c File Reference

File for hard-coded autonomous routines.

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

Functions

- void runHardCodedProgrammingSkills ()

4.31.1 Detailed Description

File for hard-coded autonomous routines.

This file contains code for hard-coded autonomous routines that use sensors.

Definition in file autonroutines.c.

4.31.2 Function Documentation

4.31.2.1 void runHardCodedProgrammingSkills ( )

Runs a programming skills routine using sensors rather than the autonomous recorder. Starts in the left side of the field shooting into the closer goal.

Definition at line 13 of file autonroutines.c.

4.32   autonroutines.c

```
00001
00007 #include "main.h"
00008
00013 void runHardCodedProgrammingSkills() {
00014     int numShots = 0;
00015     bool shooterLimitPressed = PRESSED;
00016     shoot(-127);
00017     while (numShots - 1 <= 32 /* one shot subtracted because the shooter starts uncocked */) {
00018         if (shooterLimitPressed == PRESSED && digitalRead(SHOOTER_LIMIT) ==
    UNPRESSED) {
00019             numShots++;
00020         }
00021         shooterLimitPressed = digitalRead(SHOOTER_LIMIT);
00022         delay(20);
00023     }
00024     shoot(0);
00025
00026     while (gyroGet(gyro) % ROTATION_DEG != -90 -
    CLOSE_GOAL_ANGLE /* turning right */) {
00027         int error = (-90 - CLOSE_GOAL_ANGLE) - gyroGet(gyro) %
    ROTATION_DEG;
00028         move(0, error * GYRO_P);
00029     }
00030     while (ultrasonicGet(sonar) < DISTANCE_TO_OTHER_SIDE - 30) {
00031         move(127, 0);
00032     }
00033     while (ultrasonicGet(sonar) < DISTANCE_TO_OTHER_SIDE) {
00034         move(64, 0);
00035     }
00036     while (gyroGet(gyro) % ROTATION_DEG != -2*CLOSE_GOAL_ANGLE /* turning
     left */) {
00037         int error = (-2*CLOSE_GOAL_ANGLE) - gyroGet(gyro) %
    ROTATION_DEG;
00038         move(0, error * GYRO_P);
00039     }
00040
00041     numShots = 0;
00042     shooterLimitPressed = digitalRead(SHOOTER_LIMIT);
00043     shoot(-127);
00044     while (numShots - 1 <= 32 /* one shot subtracted because the shooter starts uncocked */) {
00045         if (shooterLimitPressed == PRESSED && digitalRead(SHOOTER_LIMIT) ==
    UNPRESSED) {
00046             numShots++;
00047         }
00048         shooterLimitPressed = digitalRead(SHOOTER_LIMIT);
00049         delay(20);
00050     }
00051     shoot(0);
00052 }
00053
```

4.33   src/init.c File Reference

File for initialization code.

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

Functions

- void initializeIO ()
- void initialize ()

4.33.1 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 EXP↩
RESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERC↩
HANTABILITY 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.

4.33.2 Function Documentation

4.33.2.1 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.

4.33.2.2 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.

4.34 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     pinMode(SHOOTER_LIMIT, INPUT);
00068     leftenc = encoderInit(LEFT_ENC_TOP, LEFT_ENC_BOT, false);
00069     rightenc = encoderInit(RIGHT_ENC_TOP, RIGHT_ENC_BOT, false);
00070     lcdInit(LCD_PORT);
00071     lcdClear(LCD_PORT);
00072     lcdSetBacklight(LCD_PORT, true);
00073     lcdSetText(LCD_PORT, 1, "Init-ing gyro...");
00074     gyro = gyroInit(GYRO_PORT, GYRO_SENSITIVITY);
00075     sonar = ultrasonicInit(ULTRASONIC_ECHO_PORT,
     ULTRASONIC_PING_PORT);
00076     speakerInit();
00077     delay(1100);
00078     gyroReset(gyro);
00079     lcdSetText(LCD_PORT, 1, "Init-ed gyro!");
00080     initAutonRecorder();
00081     initGroups();
00082     if(isOnline()){
00083         loadAuton();
00084     }
00085 }
00086
```

4.35 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]

4.35.1 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.

4.35.2 Function Documentation

4.35.2.1 void addMotorGroup ( )

Adds a new motor group to the dynamic array.

Definition at line 899 of file lcddiag.c.

4.35.2.2 void delMotorGroup (int *mtr*)

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

Parameters

mtr	the ID number to delete

Definition at line 970 of file lcddiag.c.

4.35.2.3 **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

choice	the selected menu to run

Definition at line 1276 of file lcddiag.c.

4.35.2.4 **void editMotorGroup (int** *mtr* **)**

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

Parameters

mtr	the ID number of the motor group to edit

Definition at line 915 of file lcddiag.c.

4.35.2.5 **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

ignore	does nothing - required by task definition

Definition at line 1298 of file lcddiag.c.

4.35.2.6 **void formatMenuNameCenter (FILE** ∗ *lcdport,* **int** *line,* **int** *index* **)**

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

Parameters

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

Definition at line 282 of file lcddiag.c.

4.35.2.7 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.

4.35.2.8 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.

4.35.2.9 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

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 1168 of file lcddiag.c.

4.35.2.10 void runBattery (FILE * *lcdport*)

Displays the battery voltages, allowing for switching between batteries.

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 354 of file lcddiag.c.

4.35.2.11 void runConnection (FILE * *lcdport*)

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

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 1068 of file lcddiag.c.

4.35.2.12   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

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 1233 of file lcddiag.c.

4.35.2.13   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

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 727 of file lcddiag.c.

4.35.2.14   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

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 555 of file lcddiag.c.

4.35.2.15   void runMotor (FILE ∗ *lcdport*)

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

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 791 of file lcddiag.c.

4.35.2.16   void runMotorGroupMgmt (FILE ∗ *lcdport*)

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

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 1027 of file lcddiag.c.

4.35.2.17   void runRobot (FILE ∗ *lcdport*)

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

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 1117 of file lcddiag.c.

4.35.2.18   void runScreensaver (FILE ∗ *lcdport*)

Runs the screensaver that displays LCD messages.

See also

>   lcdmsg.c

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Definition at line 335 of file lcddiag.c.

4.35.2.19   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.

4.35.2.20   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 300 of file lcddiag.c.

4.35.2.21   int selectMotor (FILE ∗ *lcdport*)

Prompts the user to select an individual motor to test.

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Returns

the motor number selected

Definition at line 422 of file lcddiag.c.

4.35.2.22   int selectMotorGroup (FILE ∗ *lcdport*)

Selects the motor group to test.

Parameters

lcdport	the LCD screen's port (either UART1 or UART2)

Returns

the ID number of the group to test

Definition at line 618 of file lcddiag.c.

4.35.2.23   bool∗ selectMotorGroupMembers (bool ∗ *motor*)

Selects the motors that constitute the specified motor group.

Parameters

motor	a boolean array that stores the states of each motor in the group

Returns

a pointer to the array passed

Definition at line 829 of file lcddiag.c.

4.35.2.24   int selectSpd (int *mtr*)

Selects the speed of the motor to test.

Parameters

mtr	the motor number that is being tested

Returns

the speed selected

Definition at line 490 of file lcddiag.c.

4.35.2.25   int selectSpdGroup (int *mtr*)

Selects the speed of the motor group to be tested.

Parameters

mtr	the ID number of the motor group to be tested

Returns

the speed selected

Definition at line 665 of file lcddiag.c.

4.35.2.26   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

dest	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.

4.35.3 Variable Documentation

4.35.3.1 bool backlight = true

Boolean representing the LCD screen's backlight state.

Definition at line 36 of file lcddiag.c.

4.35.3.2 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.

4.35.3.3 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.

4.35.3.4 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.

4.35.3.5 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.

4.35.3.6 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.

4.36 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          groups[3].motor[NAUTILUS_SHOOTER_MOTOR_CENTER] = true;
00261          strcpy(groups[3].name, "Nautilus Shooter");
00262
00263          groups[4].motor[INTAKE_ROLLER_MOTOR] = true;
00264          strcpy(groups[4].name, "Intake");
00265
00266          groups[5].motor[TRANSMISSION_MOTOR] = true;
00267          strcpy(groups[5].name, "Transmission");
00268
00269          //saveGroups();
00270      } else {
00271          //loadGroups();
00272      }
00273 }
00274
00282 void formatMenuNameCenter(FILE* lcdport, int line, int index){
00283      int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(
      menuChoices[index]))/2;
00284      char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00285      for(int i = 0; i < spaces; i++){
00286          strcat(str, " ");
00287      }
00288      strcat(str, menuChoices[index]);
00289      for(int i = 0; i < spaces; i++){
00290          strcat(str, " ");
00291      }
00292      lcdSetText(lcdport, line, str);
00293 }
00294
00300 int selectMenu() {
00301      bool done = false;
00302      int val = 0;
00303      do {
```

```
00304          bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00305          bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00306          bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00307
00308          if(rightPressed && val != LCD_MENU_COUNT-1) val++;
00309          else if(rightPressed && val == LCD_MENU_COUNT-1) val = 0;
00310          else if(leftPressed && val != 0) val--;
00311          else if(leftPressed && val == 0) val = LCD_MENU_COUNT - 1;
00312
00313          formatMenuNameCenter(LCD_PORT, 1, val);
00314          if(val == 0){
00315              lcdSetText(LCD_PORT, 2, "<      SEL      >");
00316          } else if(val == LCD_MENU_COUNT-1) {
00317              lcdSetText(LCD_PORT, 2, "<      SEL      >");
00318          } else {
00319              lcdSetText(LCD_PORT, 2, "<      SEL      >");
00320          }
00321          delay(20);
00322          done = centerPressed;
00323      } while(!done);
00324      printf("Selected menu choice: %d\n", val);
00325      return val;
00326 }
00327
00335 void runScreensaver(FILE *lcdport){
00336      int cycle = 0;
00337      do {
00338          if(cycle == 0){
00339              screensaver(LCD_PORT);
00340          }
00341          delay(20);
00342          cycle++;
00343          if(cycle==150){
00344              cycle=0;
00345          }
00346      } while(!lcdAnyButtonPressed());
00347 }
00348
00354 void runBattery(FILE *lcdport){
00355      bool done = false;
00356      int val = BATT_MAIN;
00357      do {
00358          bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00359          bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00360          bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00361
00362          if(rightPressed && val != BATT_PEXP) val++;
00363          else if(rightPressed && val == BATT_PEXP) val = BATT_MAIN;
00364          else if(leftPressed && val != BATT_MAIN) val--;
00365          else if(leftPressed && val == BATT_MAIN) val = BATT_PEXP;
00366
00367          int beforepoint = 0;
00368          int afterpoint = 0;
00369          char battdisp[LCD_MESSAGE_MAX_LENGTH+1];
00370          char temp[LCD_MESSAGE_MAX_LENGTH+1];
00371          memset(battdisp, 0, sizeof(battdisp));
00372          memset(temp, 0, sizeof(temp));
00373
00374          switch(val){
00375              case BATT_MAIN: beforepoint = powerLevelMain()/1000;
00376                              afterpoint = powerLevelMain()%1000;
00377                              strcat(battdisp, "Mn Batt: ");
00378                              break;
00379              case BATT_BKUP: beforepoint = powerLevelBackup()/1000;
00380                              afterpoint = powerLevelBackup()%1000;
00381                              strcat(battdisp, "Bk Batt: ");
00382                              break;
00383              case BATT_PEXP: beforepoint = powerLevelExpander()/1000;
00384                              afterpoint = powerLevelExpander()%1000;
00385                              strcat(battdisp, "Ex Batt: ");
00386                              break;
00387          }
00388          sprintf(temp, "%d.%d V", beforepoint, afterpoint);
00389          strcat(battdisp, temp);
00390
00391          int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(battdisp))/2;
00392          char str[LCD_MESSAGE_MAX_LENGTH+1];
00393          memset(str, 0, sizeof(str));
00394          for(int i = 0; i < spaces; i++){
00395              strcat(str, " ");
00396          }
```

```
00397             strcat(str, battdisp);
00398             for(int i = 0; i < spaces; i++){
00399                 strcat(str, " ");
00400             }
00401             lcdSetText(LCD_PORT, 1, str);
00402
00403             if(val == BATT_MAIN){
00404                 lcdSetText(LCD_PORT, 2, "EX      ESC      BK");
00405             } else if(val == BATT_PEXP) {
00406                 lcdSetText(LCD_PORT, 2, "BK      ESC      MN");
00407             } else { //BATT_BKUP
00408                 lcdSetText(LCD_PORT, 2, "MN      ESC      EX");
00409             }
00410             delay(20);
00411             done = centerPressed;
00412         } while(!done);
00413 }
00414
00422 int selectMotor(FILE *lcdport){
00423     bool done = false;
00424     int val = 1;
00425     do {
00426         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00427         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00428         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00429
00430         if(rightPressed && val != 10) val++;
00431         else if(rightPressed && val == 10) val = 0;
00432         else if(leftPressed && val != 0) val--;
00433         else if(leftPressed && val == 0) val = 10;
00434
00435         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00436         if(val != 0){
00437             char temp[LCD_MESSAGE_MAX_LENGTH+1];
00438             memset(motorstr, 0, sizeof(motorstr));
00439             memset(temp, 0, sizeof(temp));
00440             strcpy(motorstr, "Motor: ");
00441             sprintf(temp, "%d", val);
00442             strcat(motorstr, temp);
00443         } else {
00444             strcpy(motorstr, "Cancel");
00445         }
00446
00447         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00448         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00449         for(int i = 0; i < spaces; i++){
00450             strcat(str, " ");
00451         }
00452         strcat(str, motorstr);
00453         for(int i = 0; i < spaces; i++){
00454             strcat(str, " ");
00455         }
00456
00457         lcdSetText(LCD_PORT, 1, str);
00458
00459         done = centerPressed;
00460         if(val == 0){
00461             lcdSetText(LCD_PORT, 2, "10      SEL      1");
00462         } else if(val == 1) {
00463             lcdSetText(LCD_PORT, 2, "ESC     SEL      2");
00464         } else if(val == 10) {
00465             lcdSetText(LCD_PORT, 2, "9       SEL    ESC");
00466         } else if (val == 9) {
00467             lcdSetText(LCD_PORT, 2, "8       SEL     10");
00468         } else {
00469             char navstr[LCD_MESSAGE_MAX_LENGTH+1];
00470             memset(navstr, 0, sizeof(navstr));
00471             sprintf(navstr, "%c      SEL      %c",(val-1) + '0', (val+1) + '0');
00472             /*navstr[0] = (val-1) + '0';*/
00473             /*strcat(navstr, "      SEL      ");*/
00474             /*navstr[LCD_MESSAGE_MAX_LENGTH] = (val+1) + '0';*/
00475             lcdSetText(LCD_PORT, 2, navstr);
00476         }
00477         delay(20);
00478     } while(!done);
00479     printf("Testing motor %d.\n", val);
00480     return val;
00481 }
00482
00490 int selectSpd(int mtr) {
00491     bool done = false;
```

```
00492      int val=0;
00493      do {
00494          bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00495          /*bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);*/
00496          /*bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);*/
00497
00498          val = (float) ((float) analogRead(AUTON_POT)/(float)
      AUTON_POT_HIGH) * 254;
00499          val -= 127;
00500          char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00501          char temp[LCD_MESSAGE_MAX_LENGTH+1];
00502          memset(motorstr, 0, sizeof(motorstr));
00503          memset(temp, 0, sizeof(temp));
00504          strcpy(motorstr, "Motor: ");
00505          sprintf(temp, "%d", mtr);
00506          strcat(motorstr, temp);
00507
00508          int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00509          char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00510          for(int i = 0; i < spaces; i++){
00511              strcat(str, " ");
00512          }
00513          strcat(str, motorstr);
00514          for(int i = 0; i < spaces; i++){
00515              strcat(str, " ");
00516          }
00517
00518          lcdSetText(LCD_PORT, 1, str);
00519
00520          char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00521          memset(speedstr, 0, sizeof(speedstr));
00522          memset(temp, 0, sizeof(temp));
00523          strcpy(speedstr, "Speed: ");
00524          sprintf(temp, "%d", val);
00525          strcat(speedstr, temp);
00526
00527          spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00528          strcpy(str, "");
00529          for(int i = 0; i < spaces; i++){
00530              strcat(str, " ");
00531          }
00532          strcat(str, speedstr);
00533          for(int i = 0; i < spaces; i++){
00534              strcat(str, " ");
00535          }
00536
00537          lcdSetText(LCD_PORT, 2, str);
00538
00539          done = (digitalRead(AUTON_BUTTON) == PRESSED || centerPressed);
00540          delay(20);
00541      } while(!done);
00542      printf("Using speed: %d\n", val);
00543      return val;
00544 }
00545
00555 void runIndivMotor(FILE *lcdport){
00556      bool done = false;
00557      int mtr = selectMotor(lcdport);
00558      if(mtr == 0) return;
00559      int spd = selectSpd(mtr);
00560      int val = spd;
00561      disableOpControl = true;
00562      motorStopAll();
00563      do {
00564          char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00565          char temp[LCD_MESSAGE_MAX_LENGTH+1];
00566          memset(motorstr, 0, sizeof(motorstr));
00567          memset(temp, 0, sizeof(temp));
00568          strcpy(motorstr, "Motor: ");
00569          sprintf(temp, "%d", mtr);
00570          strcat(motorstr, temp);
00571
00572          int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00573          char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00574          for(int i = 0; i < spaces; i++){
00575              strcat(str, " ");
00576          }
00577          strcat(str, motorstr);
00578          for(int i = 0; i < spaces; i++){
00579              strcat(str, " ");
00580          }
```

```
00581
00582          lcdSetText(LCD_PORT, 1, str);
00583
00584          char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00585          memset(speedstr, 0, sizeof(speedstr));
00586          memset(temp, 0, sizeof(temp));
00587          strcpy(speedstr, "Run Speed: ");
00588          sprintf(temp, "%d", val);
00589          strcat(speedstr, temp);
00590
00591          spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00592          strcpy(str, "");
00593          for(int i = 0; i < spaces; i++){
00594              strcat(str, " ");
00595          }
00596          strcat(str, speedstr);
00597          for(int i = 0; i < spaces; i++){
00598              strcat(str, " ");
00599          }
00600
00601          lcdSetText(LCD_PORT, 2, str);
00602
00603          done = lcdAnyButtonPressed();
00604          motorSet(mtr, val);
00605
00606          delay(20);
00607      } while(!done);
00608      disableOpControl = false;
00609 }
00610
00618 int selectMotorGroup(FILE *lcdport){
00619      bool done = false;
00620      int val = 0;
00621      do {
00622          bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00623          bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00624          bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00625
00626          if(rightPressed && val != numgroups-1) val++;
00627          else if(rightPressed && val == numgroups-1) val = -1;
00628          else if(leftPressed && val != -1) val--;
00629          else if(leftPressed && val == -1) val = numgroups-1;
00630
00631          char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00632          if(val != -1){
00633              memset(motorstr, 0, sizeof(motorstr));
00634              strcpy(motorstr, groups[val].name);
00635          } else {
00636              strcpy(motorstr, "Cancel");
00637          }
00638
00639          int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00640          char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00641          for(int i = 0; i < spaces; i++){
00642              strcat(str, " ");
00643          }
00644          strcat(str, motorstr);
00645          for(int i = 0; i < spaces; i++){
00646              strcat(str, " ");
00647          }
00648
00649          lcdSetText(LCD_PORT, 1, str);
00650
00651          done = centerPressed;
00652          lcdSetText(LCD_PORT, 2, "<      SEL      >");
00653          delay(20);
00654      } while(!done);
00655      return val;
00656 }
00657
00665 int selectSpdGroup(int mtr) {
00666      bool done = false;
00667      int val;
00668      do {
00669          bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00670          /*bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);*/
00671          /*bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);*/
00672
00673          val = (float) ((float) analogRead(AUTON_POT)/(float)
     AUTON_POT_HIGH) * 254;
00674          val -= 127;
```

```
00675          char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00676          char temp[LCD_MESSAGE_MAX_LENGTH+1];
00677          memset(motorstr, 0, sizeof(motorstr));
00678          memset(temp, 0, sizeof(temp));
00679          strcpy(motorstr, groups[mtr].name);
00680
00681          int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00682          char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00683          for(int i = 0; i < spaces; i++){
00684              strcat(str, " ");
00685          }
00686          strcat(str, motorstr);
00687          for(int i = 0; i < spaces; i++){
00688              strcat(str, " ");
00689          }
00690
00691          lcdSetText(LCD_PORT, 1, str);
00692
00693          char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00694          memset(speedstr, 0, sizeof(speedstr));
00695          memset(temp, 0, sizeof(temp));
00696          strcpy(speedstr, "Speed: ");
00697          sprintf(temp, "%d", val);
00698          strcat(speedstr, temp);
00699
00700          spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00701          strcpy(str, "");
00702          for(int i = 0; i < spaces; i++){
00703              strcat(str, " ");
00704          }
00705          strcat(str, speedstr);
00706          for(int i = 0; i < spaces; i++){
00707              strcat(str, " ");
00708          }
00709
00710          lcdSetText(LCD_PORT, 2, str);
00711
00712          done = (digitalRead(AUTON_BUTTON) == PRESSED || centerPressed);
00713          delay(20);
00714      } while(!done);
00715      return val;
00716 }
00717
00727 void runGroupMotor(FILE *lcdport){
00728      bool done = false;
00729      int mtr = selectMotorGroup(lcdport);
00730      if(mtr == -1) return;
00731      int spd = selectSpdGroup(mtr);
00732      int val = spd;
00733      disableOpControl = true;
00734      motorStopAll();
00735      do {
00736          char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00737          char temp[LCD_MESSAGE_MAX_LENGTH+1];
00738          memset(motorstr, 0, sizeof(motorstr));
00739          memset(temp, 0, sizeof(temp));
00740          strcpy(motorstr, groups[mtr].name);
00741
00742          int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00743          char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00744          for(int i = 0; i < spaces; i++){
00745              strcat(str, " ");
00746          }
00747          strcat(str, motorstr);
00748          for(int i = 0; i < spaces; i++){
00749              strcat(str, " ");
00750          }
00751
00752          lcdSetText(LCD_PORT, 1, str);
00753
00754          char speedstr[LCD_MESSAGE_MAX_LENGTH+1];
00755          memset(speedstr, 0, sizeof(speedstr));
00756          memset(temp, 0, sizeof(temp));
00757          strcat(speedstr, "Run Speed: ");
00758          sprintf(temp, "%d", val);
00759          strcat(speedstr, temp);
00760
00761          spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(speedstr))/2;
00762          strcpy(str, "");
00763          for(int i = 0; i < spaces; i++){
00764              strcat(str, " ");
```

```
00765            }
00766            strcat(str, speedstr);
00767            for(int i = 0; i < spaces; i++){
00768                strcat(str, " ");
00769            }
00770
00771            lcdSetText(LCD_PORT, 2, str);
00772
00773            for(int i = 1; i <= 10; i++){
00774                if(groups[mtr].motor[i]){
00775                    motorSet(i, val);
00776                }
00777            }
00778
00779            done = lcdAnyButtonPressed();
00780            delay(20);
00781        } while(!done);
00782        disableOpControl = false;
00783 }
00784
00791 void runMotor(FILE *lcdport){
00792     bool done = false;
00793     int val = 0;
00794     do {
00795         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00796         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00797         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00798
00799         if(rightPressed && val != 1) val++;
00800         else if(leftPressed && val != 0) val--;
00801
00802         if(val){
00803             lcdSetText(lcdport, 1, "Indiv Motor Test");
00804         } else {
00805             lcdSetText(lcdport, 1, "Group Motor Test");
00806         }
00807         done = centerPressed;
00808         if(val == 0){
00809             lcdSetText(LCD_PORT, 2, "|      SEL      >");
00810         } else if(val == 1) {
00811             lcdSetText(LCD_PORT, 2, "<      SEL      |");
00812         }
00813         delay(20);
00814     } while(!done);
00815     if(val){
00816         runIndivMotor(lcdport);
00817     } else {
00818         runGroupMotor(lcdport);
00819     }
00820 }
00821
00829 bool* selectMotorGroupMembers(bool *motor){
00830     bool done = false;
00831     int val = 1;
00832     do {
00833         bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00834         bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00835         bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00836
00837         if(rightPressed && val != 10) val++;
00838         else if(rightPressed && val == 10) val = 0;
00839         else if(leftPressed && val != 0) val--;
00840         else if(leftPressed && val == 0) val = 10;
00841
00842         char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00843         if(val != 0){
00844             char temp[LCD_MESSAGE_MAX_LENGTH+1];
00845             memset(motorstr, 0, sizeof(motorstr));
00846             memset(temp, 0, sizeof(temp));
00847             strcpy(motorstr, "Motor ");
00848             sprintf(temp, "%d:", val);
00849             strcat(motorstr, temp);
00850             if(motor[val]){
00851                 strcat(motorstr, " On");
00852             } else {
00853                 strcat(motorstr, " Off");
00854             }
00855         } else {
00856             strcpy(motorstr, "Confirm");
00857         }
00858         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
```

```
00859            char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00860            for(int i = 0; i < spaces; i++){
00861                strcat(str, " ");
00862            }
00863            strcat(str, motorstr);
00864            for(int i = 0; i < spaces; i++){
00865                strcat(str, " ");
00866            }
00867
00868            lcdSetText(LCD_PORT, 1, str);
00869
00870            done = (centerPressed && val == 0);
00871            if(centerPressed && val != 0){
00872                motor[val] = !motor[val];
00873            }
00874            if(val == 0){
00875                lcdSetText(LCD_PORT, 2, "10     SEL     1");
00876            } else if(val == 1) {
00877                lcdSetText(LCD_PORT, 2, "ESC    SEL     2");
00878            } else if(val == 10) {
00879                lcdSetText(LCD_PORT, 2, "9      SEL   ESC");
00880            } else if (val == 9) {
00881                lcdSetText(LCD_PORT, 2, "8      SEL    10");
00882            } else {
00883                char navstr[LCD_MESSAGE_MAX_LENGTH+1];
00884                memset(navstr, 0, sizeof(navstr));
00885                sprintf(navstr, "%c      SEL      %c",(val-1) + '0', (val+1) + '0');
00886                /*navstr[0] = (val-1) + '0';*/
00887                /*strcat(navstr, "      SEL      ");*/
00888                /*navstr[LCD_MESSAGE_MAX_LENGTH] = (val+1) + '0';*/
00889                lcdSetText(LCD_PORT, 2, navstr);
00890            }
00891            delay(20);
00892        } while(!done);
00893        return motor;
00894 }
00895
00899 void addMotorGroup(){
00900     MotorGroup *temp = (MotorGroup*) realloc(groups, sizeof(
    MotorGroup)*(numgroups+1));
00901     if(temp == NULL) return;
00902     groups = temp;
00903     numgroups++;
00904     memset(&groups[numgroups-1], 0, sizeof(groups[numgroups-1]));
00905     typeString(groups[numgroups-1].name);
00906     selectMotorGroupMembers(groups[numgroups-1].motor);
00907 }
00908
00915 void editMotorGroup(int mtr){
00916     if(mtr == -1) return;
00917     bool done = false;
00918     int val = 0;
00919     FILE *lcdport = LCD_PORT;
00920     do {
00921        bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00922        bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00923        bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00924
00925        if(rightPressed && val != 1) val++;
00926        else if(leftPressed && val != 0) val--;
00927
00928        char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00929        memset(motorstr, 0, sizeof(motorstr));
00930
00931        if(val){
00932            strcpy(motorstr, "Edit Name");
00933        } else {
00934            strcpy(motorstr, "Edit Motors");
00935        }
00936
00937        int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00938        char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00939        for(int i = 0; i < spaces; i++){
00940            strcat(str, " ");
00941        }
00942        strcat(str, motorstr);
00943        for(int i = 0; i < spaces; i++){
00944            strcat(str, " ");
00945        }
00946
00947        lcdSetText(lcdport, 1, str);
```

```
00948
00949            done = centerPressed;
00950            if(val == 0){
00951                lcdSetText(LCD_PORT, 2, "|      SEL      >");
00952            } else if(val == 1) {
00953                lcdSetText(LCD_PORT, 2, "<      SEL      |");
00954            }
00955            delay(20);
00956        } while(!done);
00957        if(val){
00958            typeString(groups[mtr].name);
00959        } else {
00960            selectMotorGroupMembers(groups[mtr].motor);
00961        }
00962 }
00963
00970 void delMotorGroup(int mtr){
00971        if(mtr == -1) return;
00972        int val = 0;
00973        FILE *lcdport = LCD_PORT;
00974        bool done = false;
00975        do {
00976            bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
00977            bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
00978            bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
00979
00980            if(rightPressed && val != 1) val++;
00981            else if(leftPressed && val != 0) val--;
00982
00983            char motorstr[LCD_MESSAGE_MAX_LENGTH+1];
00984            memset(motorstr, 0, sizeof(motorstr));
00985
00986            if(val){
00987                strcpy(motorstr, "Delete Forever");
00988            } else {
00989                strcpy(motorstr, "Cancel Delete");
00990            }
00991
00992            int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(motorstr))/2;
00993            char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00994            for(int i = 0; i < spaces; i++){
00995                strcat(str, " ");
00996            }
00997            strcat(str, motorstr);
00998            for(int i = 0; i < spaces; i++){
00999                strcat(str, " ");
01000            }
01001
01002            lcdSetText(lcdport, 1, str);
01003
01004            done = centerPressed;
01005            if(val == 0){
01006                lcdSetText(LCD_PORT, 2, "|      SEL      >");
01007            } else if(val == 1) {
01008                lcdSetText(LCD_PORT, 2, "<      SEL      |");
01009            }
01010            delay(20);
01011        } while(!done);
01012        if(val){
01013            memset(&groups[mtr], 0, sizeof(groups[mtr]));
01014            for(int i = mtr; i < numgroups - 1; i++){
01015                memcpy(&groups[i], &groups[i+1], sizeof(groups[i]));
01016            }
01017            numgroups--;
01018        }
01019 }
01020
01027 void runMotorGroupMgmt(FILE *lcdport){
01028        bool done = false;
01029        int val = 0;
01030        do {
01031            bool centerPressed = lcdButtonPressed(LCD_BTN_CENTER);
01032            bool leftPressed = lcdButtonPressed(LCD_BTN_LEFT);
01033            bool rightPressed = lcdButtonPressed(LCD_BTN_RIGHT);
01034
01035            if(rightPressed && val != 3) val++;
01036            else if(rightPressed && val == 3) val = 0;
01037            else if(leftPressed && val != 0) val--;
01038            else if(leftPressed && val == 0) val = 3;
01039
01040            switch(val){
```

```
01041            case 0: lcdSetText(lcdport, 1, "Add Motor Group"); break;
01042            case 1: lcdSetText(lcdport, 1, "Edit Motor Group"); break;
01043            case 2: lcdSetText(lcdport, 1, "Del Motor Group"); break;
01044            case 3: lcdSetText(lcdport, 1, "Cancel Grp. Mgmt"); break;
01045        }
01046        done = centerPressed;
01047        if(val == 0){
01048            lcdSetText(LCD_PORT, 2, "<      SEL      >");
01049        } else {
01050            lcdSetText(LCD_PORT, 2, "<      SEL      >");
01051        }
01052        delay(20);
01053    } while(!done);
01054    switch(val){
01055        case 0: addMotorGroup(); break;
01056        case 1: editMotorGroup(selectMotorGroup(lcdport)); break;
01057        case 2: delMotorGroup(selectMotorGroup(lcdport)); break;
01058        case 3: break;
01059    }
01060 }
01061
01068 void runConnection(FILE *lcdport){
01069    do {
01070        char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01071        char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01072        if(isJoystickConnected(1)){
01073            strcat(strjoy1, "J1: Connected");
01074        } else {
01075            strcat(strjoy1, "J1: Disconnected");
01076        }
01077        if(isJoystickConnected(2)){
01078            strcat(strjoy2, "J2: Connected");
01079        } else {
01080            strcat(strjoy2, "J2: Disconnected");
01081        }
01082
01083        int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01084        char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01085        for(int i = 0; i < spaces; i++){
01086            strcat(str, " ");
01087        }
01088        strcat(str, strjoy1);
01089        for(int i = 0; i < spaces; i++){
01090            strcat(str, " ");
01091        }
01092
01093        lcdSetText(lcdport, 1, str);
01094
01095        spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01096        strcpy(str, "");
01097        for(int i = 0; i < spaces; i++){
01098            strcat(str, " ");
01099        }
01100        strcat(str, strjoy2);
01101        for(int i = 0; i < spaces; i++){
01102            strcat(str, " ");
01103        }
01104
01105        lcdSetText(lcdport, 2, str);
01106
01107        delay(20);
01108    } while(!lcdAnyButtonPressed());
01109 }
01110
01117 void runRobot(FILE *lcdport){
01118    do {
01119        char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01120        char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01121        if(isOnline()){
01122            strcat(strjoy1, "Competition Mode");
01123        } else {
01124            strcat(strjoy1, "Practice Mode");
01125        }
01126        /*if(isEnabled()){
01127            strcat(strjoy2, "Robot Enabled");
01128        } else {
01129            strcat(strjoy2, "Robot Disabled");
01130        }*/
01131        sprintf(strjoy2, "Angle: %d", gyroGet(gyro));
01132
01133        int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
```

```
01134            char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01135            for(int i = 0; i < spaces; i++){
01136                strcat(str, " ");
01137            }
01138            strcat(str, strjoy1);
01139            for(int i = 0; i < spaces; i++){
01140                strcat(str, " ");
01141            }
01142
01143            lcdSetText(lcdport, 1, str);
01144
01145            spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01146            strcpy(str, "");
01147            for(int i = 0; i < spaces; i++){
01148                strcat(str, " ");
01149            }
01150            strcat(str, strjoy2);
01151            for(int i = 0; i < spaces; i++){
01152                strcat(str, " ");
01153            }
01154
01155            lcdSetText(lcdport, 2, str);
01156
01157            delay(20);
01158        } while(!lcdAnyButtonPressed());
01159 }
01160
01168 void runAuton(FILE *lcdport){
01169     do {
01170            char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01171            char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01172            if(autonLoaded == -1){
01173                strcat(strjoy1, "No Auton Loaded");
01174            } else if(autonLoaded == 0){
01175                strcat(strjoy1, "Empty Auton");
01176            } else if(autonLoaded == MAX_AUTON_SLOTS + 1){
01177                strcat(strjoy1, "Prog. Skills");
01178            } else {
01179                FILE* autonFile;
01180                char filename[AUTON_FILENAME_MAX_LENGTH];
01181                snprintf(filename, sizeof(filename)/sizeof(char), "a%d", autonLoaded);
01182                autonFile = fopen(filename, "r");
01183                char name[LCD_MESSAGE_MAX_LENGTH+1];
01184                memset(name, 0, sizeof(name));
01185                fread(name, sizeof(char), sizeof(name) / sizeof(char), autonFile);
01186                strcpy(strjoy1, name);
01187                fclose(autonFile);
01188            }
01189            if(isOnline()){
01190                strcat(strjoy2, "Recorder Off");
01191            } else {
01192                strcat(strjoy2, "Recorder On");
01193            }
01194
01195            int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01196            char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01197            for(int i = 0; i < spaces; i++){
01198                strcat(str, " ");
01199            }
01200            strcat(str, strjoy1);
01201            for(int i = 0; i < spaces; i++){
01202                strcat(str, " ");
01203            }
01204
01205            lcdSetText(lcdport, 1, str);
01206
01207            spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01208            strcpy(str, "");
01209            for(int i = 0; i < spaces; i++){
01210                strcat(str, " ");
01211            }
01212            strcat(str, strjoy2);
01213            for(int i = 0; i < spaces; i++){
01214                strcat(str, " ");
01215            }
01216
01217            lcdSetText(lcdport, 2, str);
01218
01219            delay(20);
01220        } while(!lcdAnyButtonPressed());
01221 }
```

```
01222
01233 void runCredits(FILE *lcdport){
01234     do {
01235         char strjoy1[LCD_MESSAGE_MAX_LENGTH+1] = "";
01236         char strjoy2[LCD_MESSAGE_MAX_LENGTH+1] = "";
01237         strcat(strjoy1, "Thanks Team 750W");
01238         strcat(strjoy2, "Akram Sandhu");
01239
01240         int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy1))/2;
01241         char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
01242         for(int i = 0; i < spaces; i++){
01243             strcat(str, " ");
01244         }
01245         strcat(str, strjoy1);
01246         for(int i = 0; i < spaces; i++){
01247             strcat(str, " ");
01248         }
01249
01250         lcdSetText(lcdport, 1, str);
01251
01252         spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(strjoy2))/2;
01253         strcpy(str, "");
01254         for(int i = 0; i < spaces; i++){
01255             strcat(str, " ");
01256         }
01257         strcat(str, strjoy2);
01258         for(int i = 0; i < spaces; i++){
01259             strcat(str, " ");
01260         }
01261
01262         lcdSetText(lcdport, 2, str);
01263
01264         delay(20);
01265     } while(!lcdAnyButtonPressed());
01266 }
01267
01276 void doMenuChoice(int choice){
01277     switch(choice){
01278         case MENU_SCREENSAVER: runScreensaver(
    LCD_PORT); break;
01279         case MENU_BATTERY: runBattery(LCD_PORT); break;
01280         case MENU_MOTOR: runMotor(LCD_PORT); break;
01281         case MENU_MOTOR_MGMT: runMotorGroupMgmt(
    LCD_PORT); break;
01282         case MENU_CONNECTION: runConnection(LCD_PORT); break;
01283         case MENU_ROBOT: runRobot(LCD_PORT); break;
01284         case MENU_AUTON: runAuton(LCD_PORT); break;
01285         case MENU_BACKLIGHT: lcdSetBacklight(LCD_PORT, !
    backlight); backlight = !backlight; break;
01286         case MENU_CREDITS: runCredits(LCD_PORT); break;
01287     }
01288 }
01289
01298 void formatLCDDisplay(void *ignore){
01299     lcdSetBacklight(LCD_PORT, backlight);
01300     doMenuChoice(MENU_SCREENSAVER);
01301     while(true){
01302         doMenuChoice(selectMenu());
01303     }
01304 }
01305
```

4.37 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 [ ]

4.37.1   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.

4.37.2   Function Documentation

4.37.2.1   void randlcdmsg (FILE ∗ *lcdport,* int *line*)

Displays a random LCD message from the master list.

Parameters

lcdport	the port the LCD is connected to
line	the line to display the message on

Definition at line 71 of file lcdmsg.c.

4.37.2.2   void screensaver (FILE ∗ *lcdport*)

Formats the LCD by displaying 750C title and message.

Parameters

lcdport	the port the LCD is connected to

Definition at line 90 of file lcdmsg.c.

4.37.3   Variable Documentation

4.37.3.1   char∗ lcdmsg[ ]

Master list of all LCD messages.

Definition at line 14 of file lcdmsg.c.

4.38 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     "New Nautilus",
00059     "Nauty Gears",
00060     "New 'n' Nauty",
00061     "TURBO GEARS",
00062     "itoa() Sucks"
00063 };
00064
00071 void randlcdmsg(FILE *lcdport, int line){
00072     int index = rand() % LCD_MESSAGE_COUNT;
00073     int spaces = (LCD_MESSAGE_MAX_LENGTH - strlen(lcdmsg[index]))/2;
00074     char str[LCD_MESSAGE_MAX_LENGTH+1] = "";
00075     for(int i = 0; i < spaces; i++){
00076         strcat(str, " ");
00077     }
00078     strcat(str, lcdmsg[index]);
00079     for(int i = 0; i < spaces; i++){
00080         strcat(str, " ");
00081     }
00082     lcdSetText(lcdport, line, str);
00083 }
00084
00090 void screensaver(FILE *lcdport) {
00091     lcdSetText(lcdport, 1, LCD_750C_TITLE);
00092     randlcdmsg(lcdport, 2);
00093 }
00094
```

4.39   src/motors.c File Reference

File for important motor functions.

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

Functions

- void transmissionSetPos (void ∗pos)

4.39.1   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.

4.39.2   Function Documentation

4.39.2.1   void transmissionSetPos (void ∗ *pos*)

Sets the position of the transmission.

Parameters

pos	the position to set the transmission to.

Definition at line 19 of file motors.c.

4.40   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 }
00038
```

4.41 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

4.41.1 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 EXP↩RESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERC↩HANTABILITY 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.

4.41.2 Function Documentation

4.41.2.1 void moveRobot ( )

Moves the robot based on the motor state variables.

Definition at line 128 of file opcontrol.c.

4.41.2.2 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.

4.41.2.3 void recordJoyInfo ( )

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

Definition at line 81 of file opcontrol.c.

4.41.2.4 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

target	the desired robot angle

Definition at line 73 of file opcontrol.c.

4.41.3 Variable Documentation

4.41.3.1 int dep

Speed of the lift deployment motor.

Definition at line 65 of file opcontrol.c.

4.41.3.2 int intk

Speed of the intake motors.

Definition at line 55 of file opcontrol.c.

4.41.3.3 int sht

Speed of the shooter motors.

Definition at line 50 of file opcontrol.c.

4.41.3.4 int spd

Forward/backward speed of the drive motors.

Definition at line 40 of file opcontrol.c.

4.41.3.5 int trans

Speed of the transmission motors.

Definition at line 60 of file opcontrol.c.

4.41.3.6 int turn

Turning speed of the drive motors.

Definition at line 45 of file opcontrol.c.

4.42 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 * GYRO_P;
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     bool speakerPlay = false;
00156     bool speakerButtonPressed = false;
00157     speakerTask = NULL;
00158     while (true) {
```

```
00159          if(isOnline() || progSkills == 0){
00160              if(joystickGetDigital(1, 8, JOY_UP) || joystickGetDigital(1, 8, JOY_DOWN)) {
00161                  if(!speakerButtonPressed) {
00162                      speakerPlay = !speakerPlay;
00163                      speakerButtonPressed = true;
00164                  }
00165              } else {
00166                  speakerButtonPressed = false;
00167              }
00168              if(speakerPlay){
00169                  if(speakerTask == NULL){
00170                      speakerTask = taskCreate(playSpeaker, TASK_DEFAULT_STACK_SIZE,
      NULL, TASK_PRIORITY_DEFAULT);
00171                      speakerPlay = false;
00172                  }
00173              }
00174              if(lcdDiagTask == NULL){
00175                  lcdDiagTask = taskCreate(formatLCDDisplay,
      TASK_DEFAULT_STACK_SIZE, NULL, TASK_PRIORITY_DEFAULT);
00176              } else if(taskGetState(lcdDiagTask) == TASK_SUSPENDED){
00177                  taskResume(lcdDiagTask);
00178              }
00179              if(!disableOpControl){
00180                  recordJoyInfo();
00181                  if (joystickGetDigital(1, 7, JOY_RIGHT) && !isOnline()) {
00182                      taskSuspend(lcdDiagTask);
00183                      recordAuton();
00184                      lcdSetBacklight(LCD_PORT, true);
00185                      saveAuton();
00186                  } else if (joystickGetDigital(1, 7, JOY_LEFT) && !isOnline()) {
00187                      taskSuspend(lcdDiagTask);
00188                      lcdSetBacklight(LCD_PORT, true);
00189                      loadAuton();
00190                      playbackAuton();
00191                  }
00192                  moveRobot();
00193              }
00194          } else {
00195              motorStopAll();
00196              lcdSetText(LCD_PORT, 1, "Press 7R");
00197              lcdPrint(LCD_PORT, 2,   "Last Skills: %d", progSkills);
00198              if (joystickGetDigital(1, 7, JOY_RIGHT) && !isOnline()) {
00199                  recordAuton();
00200                  saveAuton();
00201              } else if(joystickGetDigital(1, 7, JOY_UP) && !isOnline()) {
00202                  progSkills = 0;
00203              }
00204          }
00205          delay(20);
00206      }
00207 }
00208
```

4.43 src/sensors.c File Reference

File for important sensor declarations and functions.

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

Functions

- void rturn (int bodydegs)
- void lturn (int bodydegs)
- void goForward (int inches)
- void goBackward (int inches)
- char ∗ randsong ()
- void playSpeaker (void ∗ignore)

Variables

- Encoder leftenc
- Encoder rightenc
- Gyro gyro
- Ultrasonic sonar
- char ∗ songs [ ]
- TaskHandle speakerTask

4.43.1   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.

4.43.2   Function Documentation

4.43.2.1   void goBackward (int *inches*)

Moves the robot backward a specified distance.

Parameters

inches	the amount of inches to move backward

Definition at line 71 of file sensors.c.

4.43.2.2   void goForward (int *inches*)

Moves the robot forward a specified distance.

Parameters

inches	the amount of inches to move forward

Definition at line 55 of file sensors.c.

4.43.2.3   void lturn (int *bodydegs*)

Turns the robot left to a specified angle.

Parameters

bodydegs	the amount of degrees to turn the robot

Definition at line 40 of file sensors.c.

4.43.2.4   void playSpeaker (void ∗ *ignore*)

Plays a song over the speaker. This task plays a random song from the array of songs.

Parameters

ignore	does nothing - required by task definition

Definition at line 131 of file sensors.c.

4.43.2.5   char∗ randsong (   )

Returns a random song from the master list.

Returns

a pointer to the song

Definition at line 113 of file sensors.c.

4.43.2.6   void rturn (int *bodydegs*)

Turns the robot right to a specified angle.

Parameters

bodydegs	the amount of degrees to turn the robot

Definition at line 25 of file sensors.c.

4.43.3   Variable Documentation

4.43.3.1   Gyro gyro

Object representing the gyroscope.

Definition at line 78 of file sensors.c.

4.43.3.2   Encoder leftenc

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

Definition at line 18 of file sensors.c.

4.43.3.3   Encoder rightenc

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

Definition at line 23 of file sensors.c.

4.43.3.4   Ultrasonic sonar

Object representing the ultrasonic sensor.

Definition at line 83 of file sensors.c.

4.43.3.5   char∗ songs[ ]

Initial value:

```
= {
    "Batman:d=8,o=5,b=180:d,d,c#,c#,c,c,c#,c#,d,d,c#,c#,c,c,c#,c#,d,d#,c,c#,c,c,c#,c#,f,p,4f",
    "
      Spiderman:d=4,o=6,b=200:c,8d#,g.,p,f#,8d#,c.,p,c,8d#,g,8g#,g,f#,8d#,c.,p,f,8g#,c.7,p,a#,8g#,f.,p,c,8d#,g.,p,f#,8d#,c,p,8g#,2
    "Star
      Wars:d=8,o=6,b=180:f5,f5,f5,2a#5.,2f.,d#,d,c,2a#.,4f.,d#,d,c,2a#.,4f.,d#,d,d#,2c,4p,f5,f5,f5,2a#5.,2f.,d#,d,c,2a#.,4f.,d#,d,
    "Final
      Countdown:d=16,o=5,b=125:b,a,4b,4e,4p,8p,c6,b,8c6,8b,4a,4p,8p,c6,b,4c6,4e,4p,8p,a,g,8a,8g,8f#,8a,4g.,f#,g,4a.,g,a,8b,8a,8g,
    "Deep Purple-Smoke on the
      Water:d=4,o=4,b=112:c,d#,f.,c,d#,8f#,f,p,c,d#,f.,d#,c,2p,8p,c,d#,f.,c,d#,8f#,f,p,c,d#,f.,d#,c",
    "Ghost Busters:d=8,o=5,b=145:16c6,32p,16c6,e6,c6,d6,a#,2p,32c6,32p,32c6,32p,c6,a#,c6",
    "Fresh Prince Of Bel Air Theme:d=4,o=5,b=112:16a, 16p, 16a, 8p, 8a, 32d7, 32d#7, 16e7, 8p, 8e7, 8p,
      32c7, 32c#7, 8d7, 16d#7, 16d7, 16p, 16c7, 16d6, 16p, 16c6, 16p, 8b, 8p, 16a, 16p, 16a, 8p, 8a6, 32d7, 32d#7,
      16e7, 8p, 8e7, 8p, 32c7, 32c#7, 8d7, 16d#7, 16d7, 16p, 16c7, 16a, 16p, 16e6, 16p, 16a, 8p, 32b, 32c6, 32b,
      32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b,
      32c6",
    "Funky Town:d=8,o=4,b=125:c6,c6,a#5,c6,p,g5,p,g5,c6,f6,e6,c6,2p,c6,c6,a#5,c6,p,g5,p,g5,c6,f6,e6,c6",
    "
      Macarena:d=8,o=5,b=180:f,f,f,4f,f,f,f,f,f,f,f,a,c,c,4f,f,f,4f,f,f,f,f,f,d,c,4p,4f,f,f,4f,f,f,f,f,f,a,4p,2c.6,4a,c6,a,
    "Mission
      Impossible:d=16,o=5,b=100:32d,32d#,32d,32d#,32d,32d#,32d,32d#,32d,32d,32d#,32e,32f,32f#,32g,g,8
     p,g,8p,a#,p,c6,p,g,8p,g,8p,f,p,f#,p,g,8p,g,8p,a#,p,c6,p,g,8p,g,8p,f,p,f#,p,a#,g,2d,32p,a#,g,2c#,32p,a#,g,2c,p,a#4,c",
    "USA National
      Anthem:d=8,o=5,b=120:e.,d,4c,4e,4g,4c6.,p,e6.,d6,4c6,4e,4f#,4g.,p,4g,4e6.,d6,4c6,2b,a,4b,c6.,16p,4c6,4g,4e,32p,4c",
    "
      Bond:d=4,o=5,b=80:32p,16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,16d#6,16d
     #6,16d#6,16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,16d#6,16d6,16c#6,16c#7,c.7,16g#6,16f#6,g#.6
    "
      GoodBad:d=4,o=5,b=56:32p,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,d#,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,c#6,32a#,32d#6,32a
    "
      MetalGear:d=8,o=6,b=125:4e5,4d5,2c5,d5,e5,a4,4e5,2d5,c5,d5,4e.5,a5,g5,e5,4c5,2d5,e5,a5,2c,b5,c,d,4c,2a5
     ,g5,a5,4b.5,c,4b5,a5,g5,1a5,4b5,4a5,2g5,a5,b5,e5,4b5,2a5,g5,a5,4b.5,e,d,b5,4g5,2a5,b5,e,2g,f#,g,a,4g,2e,d,e,4f#.,g,4f#,e,d,1
    "
      Jeopardy:d=4,o=6,b=125:c,f,c,f5,c,f,2c,c,f,c,f,a.,8g,8f,8e,8d,8c#,c,f,c,f5,c,f,2c,f.,8d,c,a#5,a5,g5,f5,
     p,d#,g#,d#,g#5,d#,g#,2d#,d#,g#,d#,g#,c.7,8a#,8g#,8g,8f,8e,d#,g#,d#,g#5,d#,g#,2d#,g#.,8f,d#,c#,c,p,a#5,p,g#.5,d#,g#",
    "MichaelJackson_Beat_It:d=4,o=5,b=138:g#7, p, d#6, d6, d#6, d6, d#6, d6, c6, d6, d#6, d6, d#6, d6, c6,
      d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, c6, d6, d#6, d6, d#6,
      d6, d#6, d6, c6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, 16c,
      16c, 16c, 16c, 1",
    "Michael Jackson - Thriller:d=31,o=5,b=112:24b., 15p, 16d.6, p, 24b., 15p, 9e.6, p, 2d.6, 3p, 9d.6, p,
      16c#.6, 10p, 6b., 3p, 24b., 15p, 16b., p, 49a., 10p, 16a., p, 49g., p, 9g., p, 24e., 15p, 12g., 153, 24a.,
      15p, 16b., p, 24a., 15p, 16a., p, 24g., 153, 16b., 1"
}
```

Master list of all songs.

Definition at line 88 of file sensors.c.

4.43.3.6   TaskHandle speakerTask

Object representing the speaker song task. The speaker song task runs in a separate thread from the operator control code. This prevents it from blocking the driving code from executing.

Definition at line 123 of file sensors.c.

4.44   sensors.c

```
00001
00013 #include "main.h"
00014
00018 Encoder leftenc;
00019
00023 Encoder rightenc;
00024
00025 void rturn(int bodydegs) {
00026     clearDriveEncoders();
00027     float turndeg;
00028     float encdegperbodydeg = DRIVE_WHEELBASE / (DRIVE_DIA *
     DRIVE_GEARRATIO);
00029     turndeg = encdegperbodydeg * bodydegs;
00030
00031     while(abs(encoderGet(rightenc)) < abs(turndeg)) {
00032         move(0, MOTOR_MAX);
00033         delay(20);
00034     }
00035
00036     move(0, 0);
00037     clearDriveEncoders();
00038 }
00039
00040 void lturn(int bodydegs) {
00041     clearDriveEncoders();
00042     float turndeg;
00043     float encdegperbodydeg = DRIVE_WHEELBASE / (DRIVE_DIA *
     DRIVE_GEARRATIO);
00044     turndeg = encdegperbodydeg * bodydegs;
00045
00046     while(abs(encoderGet(leftenc)) < abs(turndeg)) {
00047         move(0, -MOTOR_MAX);
00048         delay(20);
00049     }
00050
00051     move(0, 0);
00052     clearDriveEncoders();
00053 }
00054
00055 void goForward(int inches) {
00056     int deg;
00057     float encperinch;
00058     encperinch = 360/(DRIVE_DIA * PI * DRIVE_GEARRATIO);
00059     deg = encperinch * (float)inches;
00060
00061     clearDriveEncoders();
00062     while(abs(encoderGet(rightenc)) <  abs(deg))
00063     {
00064         move(sign(inches) * 127, 0);
00065         delay(20);
00066     }
00067     move(0,0);
00068     clearDriveEncoders();
00069 }
00070
00071 void goBackward(int inches) {
00072     goForward(-inches);
00073 }
```

```
00074
00078 Gyro gyro;
00079
00083 Ultrasonic sonar;
00084
00088 char* songs[] = {
00089     "Batman:d=8,o=5,b=180:d,d,c#,c#,c,c,c#,c#,d,d,c#,c#,c,c,c#,c#,d,d#,c,c#,c,c,c#,c#,f,p,4f",
00090     "
    Spiderman:d=4,o=6,b=200:c,8d#,g.,p,f#,8d#,c.,p,c,8d#,g,8g#,g,f#,8d#,c.,p,f,8g#,c.7,p,a#,8g#,f.,p,c,8d#,g.,p,f#,8d#,c,p,8g#,2
00091     "Star
    Wars:d=8,o=6,b=180:f5,f5,f5,2a#5.,2f.,d#,d,c,2a#.,4f.,d#,d,c,2a#.,4f.,d#,d,d#,2c,4p,f5,f5,f5,2a#5.,2f.,d#,d,c,2a#.,4f.,d#,c
00092     "Final
    Countdown:d=16,o=5,b=125:b,a,4b,4e,4p,8p,c6,b,8c6,8b,4a,4p,8p,c6,b,4c6,4e,4p,8p,a,g,8a,8g,8f#,8a,4g.,f#,g,4a.,g,a,8b,8a,8g
00093     "Deep Purple-Smoke on the
    Water:d=4,o=4,b=112:c,d#,f.,c,d#,8f#,f,p,c,d#,f.,d#,c,2p,8p,c,d#,f.,c,d#,8f#,f,p,c,d#,f.,d#,c",
00094     "Ghost Busters:d=8,o=5,b=145:16c6,32p,16c6,e6,c6,d6,a#,2p,32c6,32p,32c6,32p,c6,a#,c6",
00095     "Fresh Prince Of Bel Air Theme:d=4,o=5,b=112:16a, 16p, 16a, 8p, 8a, 32d7, 32d#7, 16e7, 8p, 8e7, 8p,
    32c7, 32c#7, 8d7, 16d#7, 16d7, 16p, 16c7, 16d6, 16p, 16c6, 16p, 8b, 8p, 16a, 16p, 16a, 8p, 8a6, 32d7, 32d#7,
    16e7, 8p, 8e7, 8p, 32c7, 32c#7, 8d7, 16d#7, 16d7, 16p, 16c7, 16a, 16p, 16e6, 16p, 16a, 8p, 32b, 32c6, 32b,
    32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b, 32c6, 32b,
    32c6",
00096     "Funky Town:d=8,o=4,b=125:c6,c6,a#5,c6,p,g5,p,g5,c6,f6,e6,c6,2p,c6,c6,a#5,c6,p,g5,p,g5,c6,f6,e6,c6",
00097     "
    Macarena:d=8,o=5,b=180:f,f,f,4f,f,f,f,f,f,f,f,a,c,c,4f,f,f,4f,f,f,f,f,f,d,c,4p,4f,f,f,4f,f,f,f,f,f,f,a,4p,2c.6,4a,c6,a,
00098     "Mission
    Impossible:d=16,o=5,b=100:32d,32d#,32d,32d#,32d,32d#,32d,32d#,32d,32d,32d#,32e,32f,32f#,32g,g,8
    p,g,8p,a#,p,c6,p,g,8p,g,8p,f,p,f#,p,g,8p,g,8p,a#,p,c6,p,g,8p,g,8p,f,p,f#,p,a#,g,2d,32p,a#,g,2c#,32p,a#,g,2c,p,a#4,c",
00099     "USA National
    Anthem:d=8,o=5,b=120:e.,d,4c,4e,4g,4c6.,p,e6.,d6,4c6,4e,4f#,4g.,p,4g,4e6.,d6,4c6,2b,a,4b,c6.,16p,4c6,4g,4e,32p,4c",
00100     "
    Bond:d=4,o=5,b=80:32p,16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,16d#6,16d
    #6,16d#6,16c#6,32d#6,32d#6,16d#6,8d#6,16c#6,16c#6,16c#6,16c#6,32e6,32e6,16e6,8e6,16d#6,16d6,16c#6,16c#7,c.7,16g#6,16f#6,g#.
00101     "
    GoodBad:d=4,o=5,b=56:32p,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,d#,32a#,32d#6,32a#,32d#6,8a#.,16f#.,16g#.,c#6,32a#,32d#6,32a
00102     "
    MetalGear:d=8,o=6,b=125:4e5,4d5,2c5,d5,e5,a4,4e5,2d5,c5,d5,4e.5,a5,g5,e5,4c5,2d5,e5,a5,2c,b5,c,d,4c,2a5
    ,g5,a5,4b.5,c,4b5,a5,g5,1a5,4b5,4a5,2g5,a5,b5,e5,4b5,2a5,g5,a5,4b.5,e,d,b5,4g5,2a5,b5,e,2g,f#,g,a,4g,2e,d,e,4f#.,g,4f#,e,d,1
00103     "
    Jeopardy:d=4,o=6,b=125:c,f,c,f5,c,f,2c,c,f,c,f,a.,8g,8f,8e,8d,8c#,c,f,c,f5,c,f,2c,f.,8d,c,a#5,a5,g5,f5,
    p,d#,g#,d#,g#5,d#,g#,2d#,d#,g#,d#,g#,c.7,8a#,8g#,8g,8f,8e,d#,g#,d#,g#5,d#,g#,2d#,g#.,8f,d#,c#,c,p,a#5,p,g#.5,d#,g#",
00104     "MichaelJackson_Beat_It:d=4,o=5,b=138:g#7, p, d#6, d6, d#6, d6, c6, d6, d#6, d6, d#6, d6, d#6, d6, c6,
    d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, c6, d6, d#6, d6, d#6,
    d6, d#6, d6, c6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, d#6, d6, 16c,
    16c, 16c, 16c, 1",
00105     "Michael Jackson - Thriller:d=31,o=5,b=112:24b., 15p, 16d.6, p, 24b., 15p, 9e.6, p, 2d.6, 3p, 9d.6, p,
    16c#.6, 10p, 6b., 3p, 24b., 15p, 16b., p, 49a., 10p, 16a., p, 49g., p, 9g., p, 24e., 15p, 12g., 153, 24a.,
    15p, 16b., p, 24a., 15p, 16a., p, 24g., 153, 16b., 1"
00106 };
00107
00113 char* randsong() {
00114     int index = rand() % SONG_COUNT;
00115     return songs[index];
00116 }
00117
00123 TaskHandle speakerTask;
00124
00131 void playSpeaker(void *ignore){
00132     char *rtttl = randsong();
00133     speakerPlayRtttl(rtttl);
00134     speakerTask = NULL;
00135 }
```

Index