EEL4914 ECE Design II
Final Report

SITH HAPPENS

by

Steven Paek (stevenpaek11@ufl.edu)
Daniel Suen (dsuen1@ufl.edu)

Department of Electrical and Computer Engineering
University of Florida

April 2019

Contents

List of Figures

Abstract

Our project is to build the "Star Wars R2-D2" using an RC Robot and Controller. The objective is to build a fun robot that people can play with that will behave like the robot from the Star Wars series. We hope that the robot will be entertaining, have motion and sound capabilities, and be easy to use. The roles for the robot and controller are described below.

For the controller, we have a physical controller board with multiple peripherals, and it will communicate with the robot via XBee. This controller will have a joystick to control the robot's movement. Additionally, it will have several buttons/controls to operate different functions of the robot such as audio output and other potential modes of operation such as manual and autonomous modes. The controller will be equipped with an 16x2 character LCD that depicts the robot's status for the user. The controller will also run off of a set of AA batteries.

For the robot, a microcontroller will drive PWM signals to control the speed of the DC motors; by doing so, the robot's basic movement will be achieved. The control will be received via XBee from the controller. For aesthetics, the R2-D2 robot will have an LED circuit which will be controlled by the microcontroller to display different colors on RGB LEDs. For example, the LEDs will default to "Blue", just like the original R2-D2, but will adjust based on changing modes. When in use, the robot will play R2-D2 sounds from the movies. This robot will be battery powered, running off of a set of AA batteries.

Chapter 1

Project Introduction

1.1 Objectives

In order to meet our goal of developing a remote-controlled car, we require both digital and analog circuitry. To allow the user to control the robot, we desire various inputs to the system; these inputs come in the form of sensors such as buttons, switches, and an analog joystick. At the same time, the robot needs to be able to receive commands from the controller and exercise those commands in real-time. As such, an XBee module is attached to both the robot and the controller for wireless communication, and the microcontrollers can interact with the XBee modules using UART.

The controller will have an LCD in order to let the user know the robot's status. On the robot end, we will have PWM and an H-Bridge circuit for motor control, an audio circuit including an external DAC to output sound effects from the Star Wars universe, and LEDs. We also include an ultrasonic sensor on the robot for potential obstacle avoidance. Both the controller and the robot require batteries for power.

Our specific desired functionalities can be broken down as follows:

Digital Objectives:

- XBee UART Communication
- PWM Control for Motors
- SPI for the DAC
- SPI for the RGB LEDs

- Digital Control for the LCD
- Microprocessor Controls for R2-D2 and Controller (GPIO)
- Ultrasonic Sensor

Analog Objectives:

- Battery Recharge Circuit
- H-Bridge Circuit
- Joystick Control
- Audio Amplifier Circuit

1.2 Technology Selection

The overall goal of the system is to have an interactive robot that can be entertaining. Initially, we were bouncing between two different options - using a traditional handheld remote controller or using a controller in the form of a glove with flex sensors.

We chose to implement a traditional handheld remote controller to operate the R2-D2 robot because it would be more intuitive to use, and the form factor was a big reason behind our decisions. This choice caused us to abandon the use of flex sensors and accelerometers and instead focus on more classical sensors such as digital buttons, switches, and a joystick.

Additionally, we chose to drive the robot using 4 DC motors connected to wheels. We decided to implement proportional steering and throttle because again, this would be the most intuitive to the end user. Ultimately, we also decided to put an LCD on the controller in order to update the user with the status; this also ended up helping us when we were working to debug our code. These specific design decisions led us to choose which specific hardware to implement.

Chapter 2

Analysis of Competitive Products

There are existing R2-D2 toy robots with different capabilities and price ranges. For example, Disney sells an R2-D2 figurine that spins, lights up, and makes realistic Star Wars sounds. Additionally, Hasbro also markets a similar robot that is a controlled by a user phone application and can dance to prerecorded music.

Figure 2.1 R2-D2 Figurine, sold by Disney

Figure 2.2 R2-D2 Robot, sold by Hasbro

These products are successful in providing functionality that is entertaining to the user. However, the first option lacks a fully manual driving mode while the second option is expensive. Because we would like to focus more on a hardware design, we choose to implement a hardware controller rather than develop a software app for a mobile phone. This also allows us to have more flexibility in that we can control the robot wirelessly without using other protocols such as Bluetooth or WiFi. Furthermore, as opposed to the Disney figurine, we choose to implement a fully manual driving mode to allow more flexibility for the user.

Chapter 3

Project Details

3.1 Project Architecture

The block diagram is divided into two parts: the controller and the R2-D2 robot.

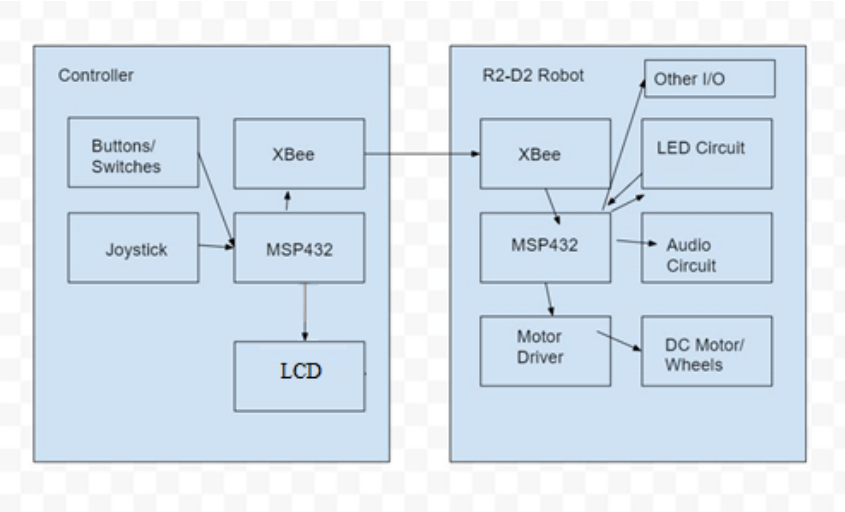

Figure 3.1 Block Diagram

The controller is powered by a microcontroller (MSP432), and it accepts different inputs including digital buttons, switches, and an analog joystick. The digital buttons and switches are for controlling which mode the robot is in (autonomous/-manual mode), motion, and sound. The analog joystick is used for determining which direction to move the robot. These control signals (joystick, switches, and buttons) are sent via XBee wireless communication to the R2-D2 robot. Additionally, the MSP432 on the controller end updates an LCD screen to inform the user of the robot's status.

The R2-D2 robot is also powered by the same microcontroller (MSP432). On the robot end, the XBee is used to receive the different user inputs, and each byte is parsed in code to determine the received data. Then, the MSP432 is used to drive various functions on the robot itself by communicating data via SPI to the

LEDs and the DAC for audio, and sending PWM signals to the H-bridge circuit for movement (which are sent to the robot's wheels). An ultrasonic sensor is also included on the robot, and the MSP432 interfaces it using GPIO and reads pulse widths using timers to determine distance to the closest object.

3.2 Hardware Selection

3.2.1 Microcontroller

Because the project consists of both a controller and a robot, we need two separate microcontrollers with the ability to communicate with each other. Since we do not require a microprocessor with specific capabilities, we select the MSP432 for both the ease of use and its versatility. The MSP432 served as the microcontroller on both the controller and robot sides.

This specific microcontroller was selected because it is low-power, relatively inexpensive, and provides much of the desired functionality (including UART, an ADC with sufficient resolution of 14 bits, built in SPI, and multiple timer channels for PWM) for our purposes. There is also sufficient GPIO for the task at hand. It can run at 48MHz as well, which is good when we are running computationally intensive tasks at the same time.

- MSP432: Up to 48MHz CPU, FPU, 14-bit ADC, UART, 16-bit Timers with PWM (Robot/Controller)

3.2.2 Wireless Communication

The XBee module series 2 is used to have two microcontrollers communicate with each others wirelessly through UART. This type of communication was selected because it is easy to use and interface with. We connect each module to 3.3 V and ground, and then connect the "data out" to the RX pin and the "data in" to the TX pin for the same UART channel on the MSP432. It works the same way as a

wire-to-wire transmission would. We also configure the XBees to be talking on the same channel with destination and source addresses set appropriately.

- XBee: RF Module IEEE 802.15.4 Standard, UART Communication (Robot/Controller)

3.2.3 Audio Circuit

To output sound from the microcontroller on the R2-D2 robot side, we use the MSP432 to send signals to a 10-bit external DAC (LTC1661). The output of the DAC is sent through a Class AB Amplifier circuit with a linear gain of 20 before sending it to a speaker. For the DAC, the LTC1661 was selected because we are both familiar with the chip, and it supports sufficient resolution (10-bit) for audio.

- LTC1661: 10-bit External DAC, 2.7V to 5.5V Supply (Robot)
- Audio Power Amplifier Circuit (Robot)

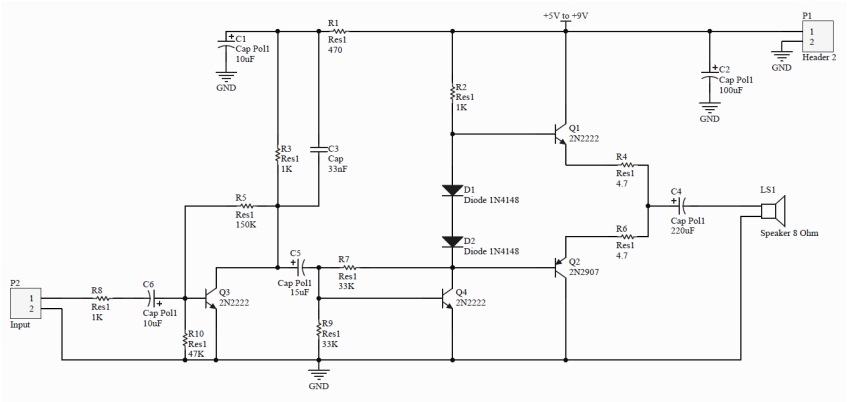

Figure 3.2 Class AB Amplifier

3.2.4 Motor Driver

The motor driver circuit consists of an H bridge to drive four motors for movement. This is essentially a switching circuit, and it controls how often voltage is allowed to be supplied to the motors. By changing the PWM duty cycle, we can change the total amount of power, thereby changing the speed. We can also alter the direction

of current flow to the motors. When the left and right motors turn at different speeds or even different directions, we will achieve turning or pivoting motions.

- H-Bridge Circuit (Robot)

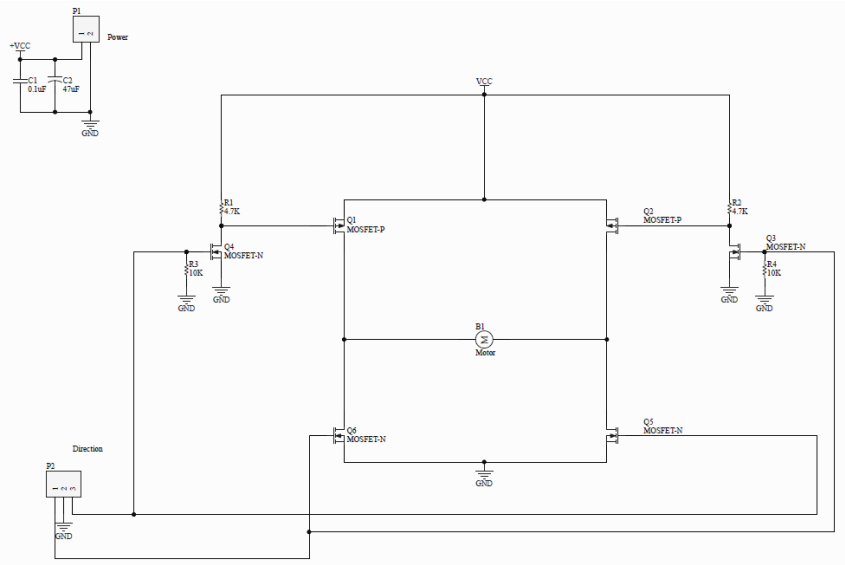

Figure 3.3 H-Bridge Circuit

3.2.5 Battery Charging Circuit

We developed a battery charging circuit using a Schmitt trigger. The charger was designed so that if the battery voltage dropped below 3.8 V, current would flow into the battery, charging it. If the battery voltage exceeded 4.18 V, then the current would stop flowing to prevent the battery from overcharging.

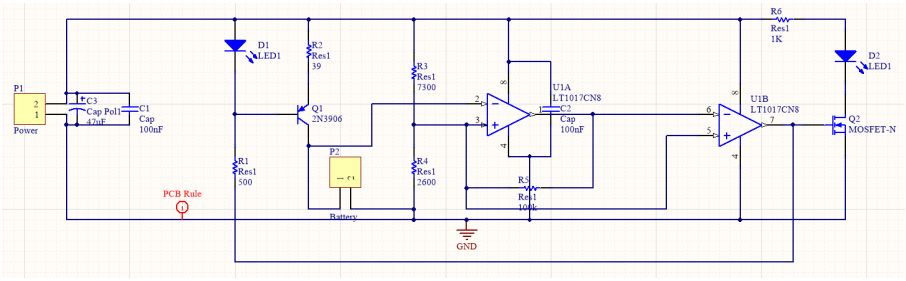

Figure 3.4 Battery Charger Circuit

3.2.6 LED Circuit

We included the use of the APA102 LEDs, and there are 32 bits used to control an LED. We send this data via SPI, and from there we can adjust the R, G, and B registers to create an overall color. These LEDs were selected mainly for the ease of use.

3.2.7 Ultrasonic Sensor

We are using an HC-SR04 ultrasonic sensor on the robot for obstacle detection, and we interface it using GPIO. We send a high pulse to trigger an ultrasonic pulse to be sent, and we read the incoming pulse widths using timers. The length of the incoming pulse is proportional to the distance away from the closest object, and we use a preset threshold to determine how sensitive the robot is.

3.2.8 Controller Inputs

The following digital buttons and switches and analog joystick will serve to provide user input to the system for robot control.

- Push Buttons
- Switches
- Analog Joystick

3.3 Software

We include rough flowcharts to describe the functionality of the software design at a high level. The first flowchart describes the functionality of the controller.
At the start, the controller initializes all of the systems including the digital ports, UART, LCD, and ADC. The software consists primarily of a while-true loop in which the program alternates between reading the analog joystick pins and the digital inputs and then making decisions. The analog joystick determines which

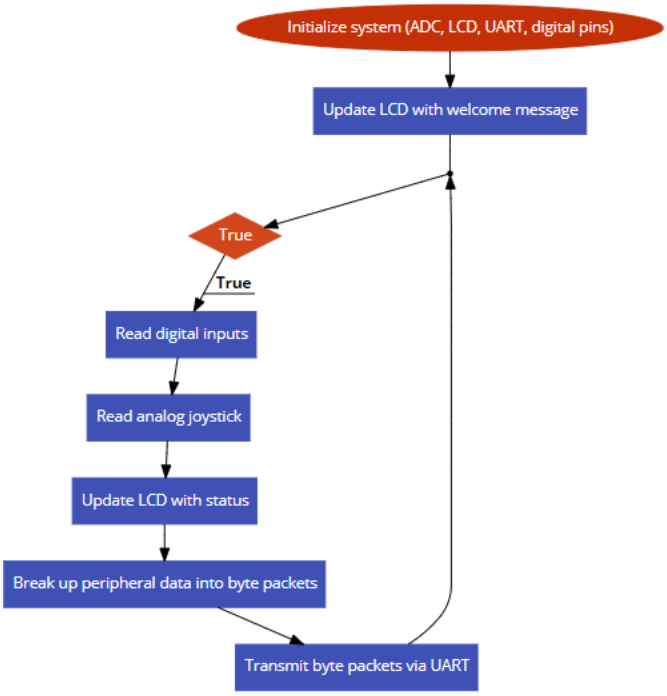

Figure 3.5 Controller Flowchart

direction to move the robot while the digital inputs are used to determine motion, sound, and mode. These commands are all sent via XBee to the R2-D2 robot. The transmission happens inside the while loop.

The second flowchart describes the functionality of the R2-D2 robot.

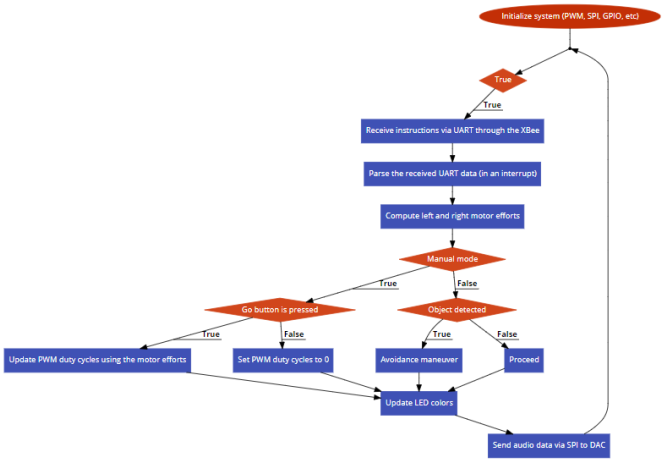

Figure 3.6 Robot (R2-D2) Flowchart

The software for the robot initializes all the necessary systems and consists of a while-true loop and several interrupt service routines. In the loop, the program receives the instructions from the controller via XBee and then reconstructs the

data. The UART data is received in an interrupt service routine, and the byte data is parsed in the ISR. We compute the left and right motor efforts from the joystick ADC values. If they exceed a certain threshold and a button is pressed, the PWM duty cycles are changed, allowing the robot to move. Meanwhile, the robot also plays sounds and changes LEDs in the background.

At the same time, if a switch is flipped on, the robot will switch to an autonomous mode, where it will move and attempt collision avoidance using the ultrasonic sensor to determine distance.

Chapter 4

Bill of Materials

Major Part	Vendor	Quantity	Unit Price	Part Price
MSP432	Digikey	2	$6.86	$13.72
16x2 Character LCD	Digikey	1	$14.69	$14.69
XBee Series 2	Digikey	2	$17.50	$35.00
Analog Joystick	Digikey	1	$3.95	$3.95
Push Buttons	Digikey	2	$0.74	$1.48
DIP Switches	Digikey	1	$0.76	$0.76
LM2940	Digikey	2	$1.56	$3.12
LM3940	Digikey	2	$1.82	$3.64
Capacitor (100nF)	Digikey	14	$0.11	$1.54
Capacitor (10uF)	Digikey	2	$0.11	$0.22
Resistor (0 Ohms)	Digikey	2	$0.11	$0.22
Resistor (47 kOhms)	Digikey	2	$0.11	$0.22
Resistor (91 kOhms)	Digikey	2	$0.11	$0.22
Capacitor (22pF)	Digikey	8	$0.11	$0.88
Capacitor (4.7uF)	Digikey	2	$0.11	$0.22
Capacitor (47uF)	Digikey	8	$0.11	$0.88
Inductor (4.7uH)	Digikey	2	$0.11	$0.22
Capacitor (100pF)	Digikey	2	$0.11	$0.22
12 MHz Oscillator	Digikey	2	$0.28	$0.56
1N4148 Diode	Digikey	2	$0.10	$0.20
Resistor (100 Ohms)	Digikey	3	$0.11	$0.33
Resistor (100 kOhms)	Digikey	4	$0.11	$0.44
Resistor (1 kOhms)	Digikey	3	$0.11	$0.33
Capacitor (470uF)	Digikey	1	$0.11	$0.11
2907	Digikey	1	$2.58	$2.58
2222	Digikey	2	$2.58	$5.16
Capacitor (100nF)	Digikey	5	$0.11	$0.55
DACLT1661	Digikey	1	$4.00	$4.00
uSD Card Reader	Digikey	1	$10.95	$10.95
8 Ohm Speaker	Digikey	1	$8.52	$8.52
HC-SR04 Sensor	Digikey	1	$3.95	$3.95
APA102 LED	Digikey	1	$79.95	$79.95
Resistor (10 kOhms)	Digikey	1	$0.11	$0.11
Resistor (4.7 kOhms)	Digikey	1	$0.11	$0.11
2N222	Digikey	4	$2.58	$10.32
DC Brush Motors/Wheels	Amazon	1 Set of 4	$15.00	$15.00
IRLZ44N	Digikey	4	$1.25	$5.00
SUP65P04	Digikey	4	$1.15	$4.60
Resistor (500 Ohm)	Digikey	1	$0.11	$0.11
Resistor (39 Ohm)	Digikey	1	$0.11	$0.11
Resistor (2.6 kOhm)	Digikey	1	$0.11	$0.11
Resistor (7.3 kOhm)	Digikey	1	$0.11	$0.11
LEDs	Digikey	2	$0.11	$0.22
LT1017	Digikey	1	$4.72	$4.72
Total				$239.35

Chapter 5

Work Responsibilities and Gantt Chart

5.1 Work Responsibilities

For the project, Steven Paek was assigned to be in charge of the R2-D2 robot, and Daniel Suen was assigned to be in charge of the controller. As such, the work responsibilities were divided as follows:

Steven Paek	**Daniel Suen**
R2-D2 PCB	Controller PCB
Motor PWM	Battery Recharge PCB
Audio Circuit with External DAC	Joystick and Button Control
LED Circuit	Controller LCD
Other I/O	XBee Communication

There was collaboration on both sides for robot motion for both parsing the received data from the controller and turning that into motor efforts for PWM duty cycles.

5.2 Gantt Chart

The Gantt chart outlines the proposed timeline for completing the project and meeting our own internal deadlines. There was a focus on completing the most of the PCB designs and analog circuitry towards the beginning of the semester.

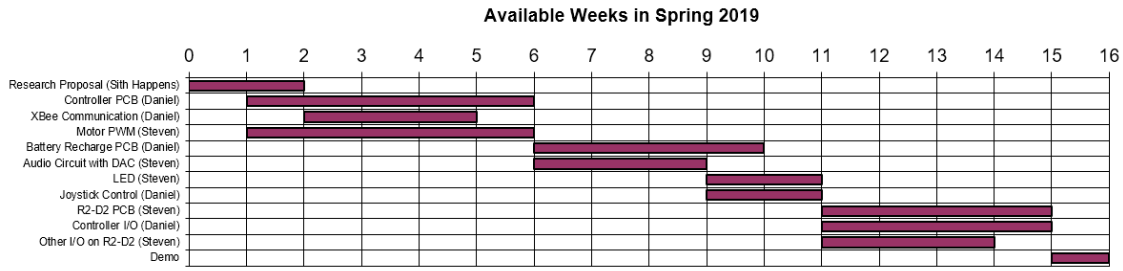

Figure 5.1 Gantt Chart

Appendix

Audio Circuit Schematic and PCB

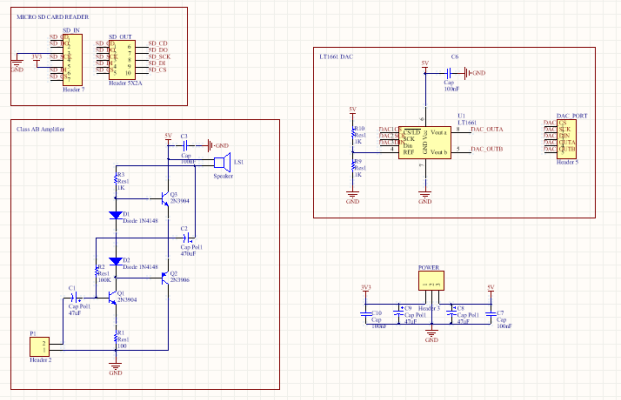

Figure 5.2 Audio Circuit Schematic

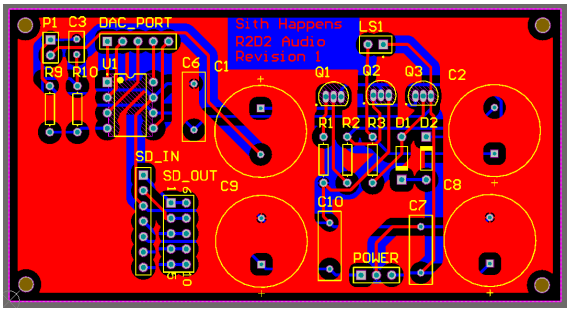

Figure 5.3 Top Layer of Audio Circuit PCB

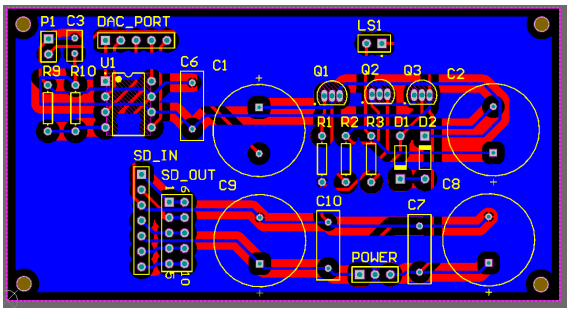

Figure 5.4 Bottom Layer of Audio Circuit PCB

Motor Circuit (H-Bridge) Schematic and PCB

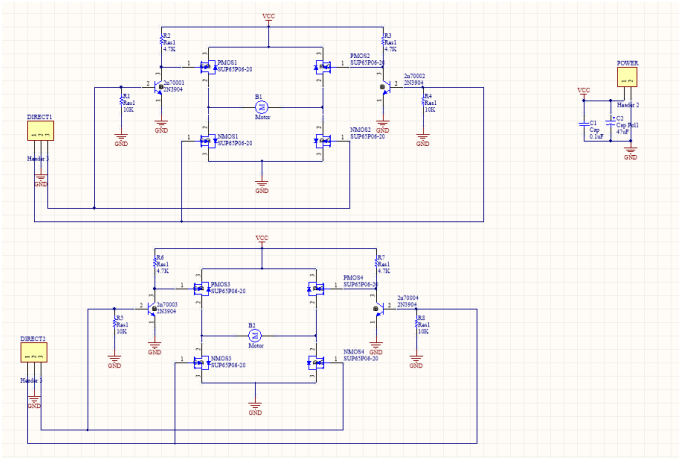

Figure 5.5 Motor Circuit Schematic

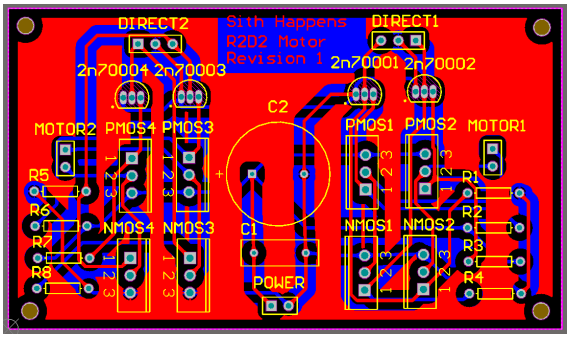

Figure 5.6 Top Layer of Motor Circuit PCB

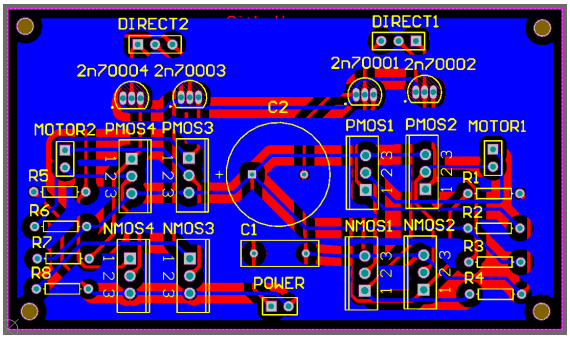

Figure 5.7 Bottom Layer of Motor Circuit PCB

Robot MSP432 Schematic and PCB

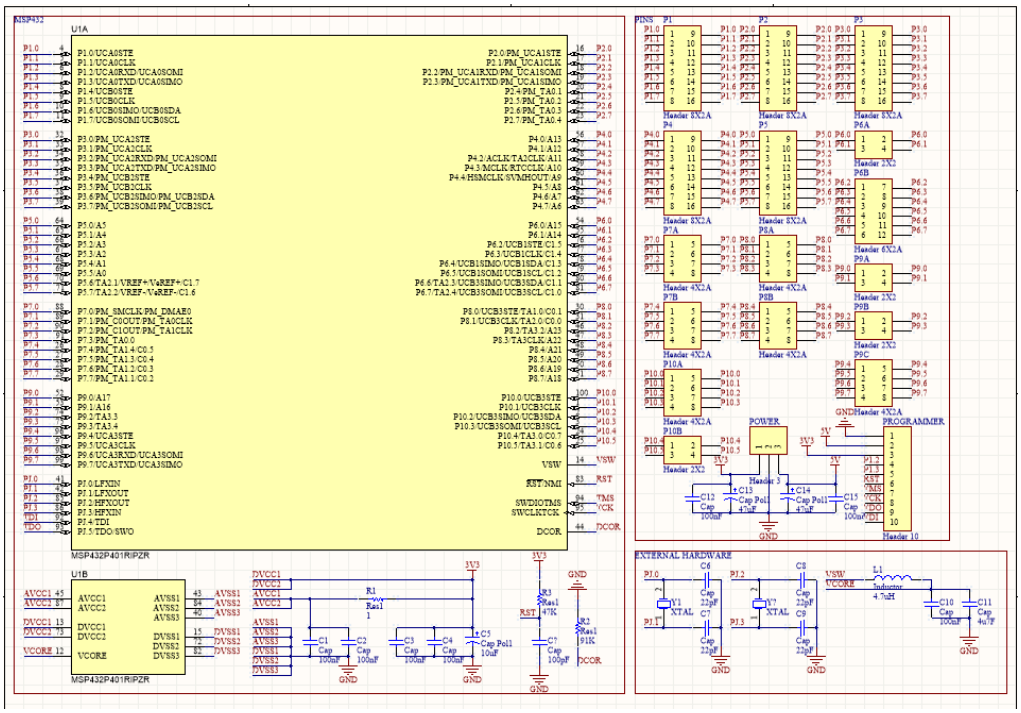

Figure 5.8 Robot MSP432 Circuit Schematic

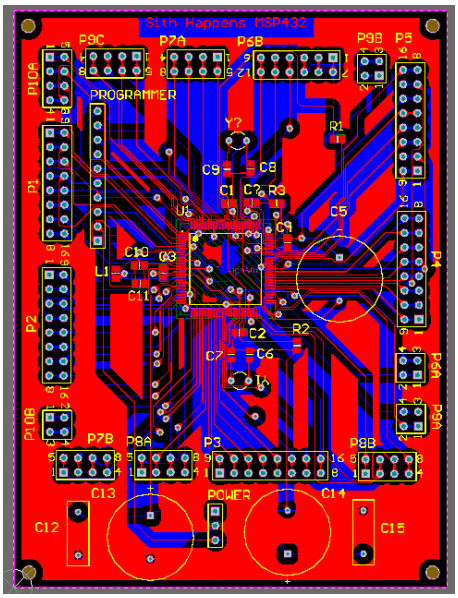

Figure 5.9 Top Layer of Robot MSP432 PCB

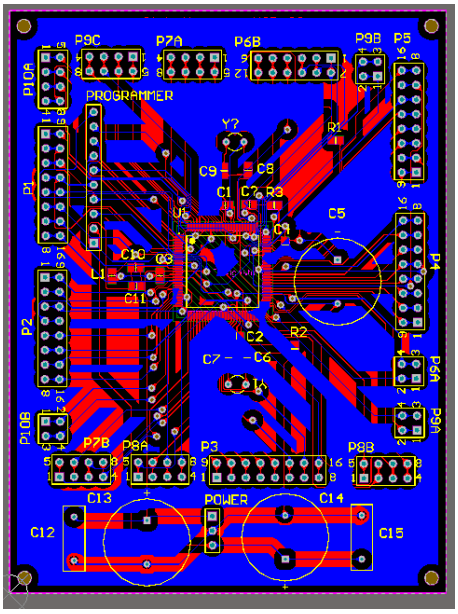

Figure 5.10 Bottom Layer of Robot MSP432 PCB

Sensor Schematic and PCB

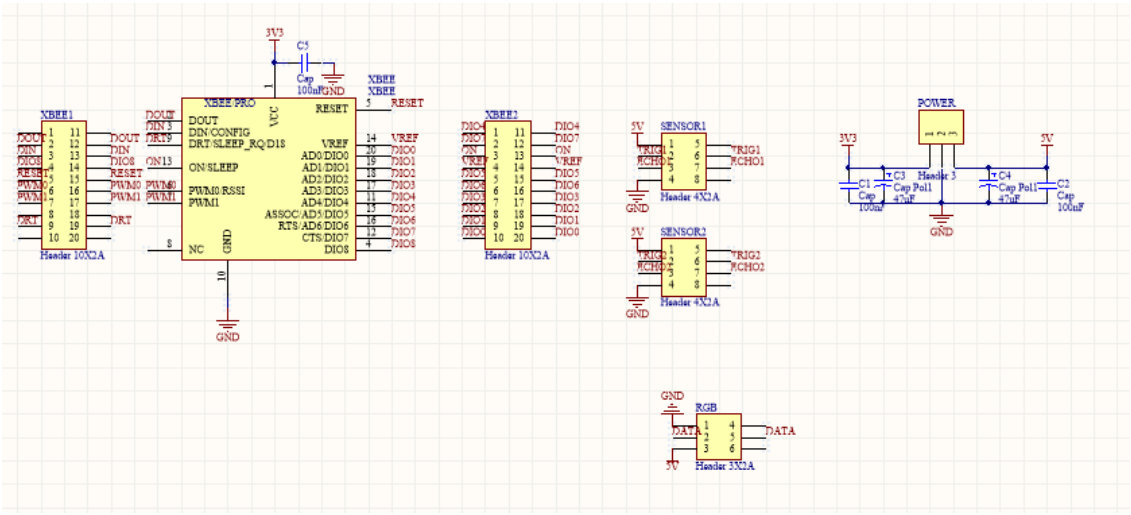

Figure 5.11 Sensor Circuit Schematic

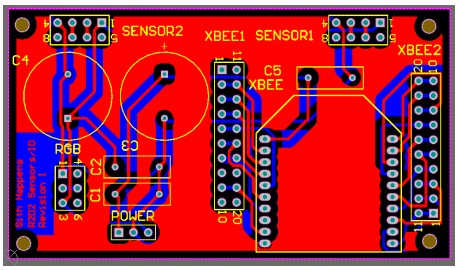

Figure 5.12 Top Layer of Sensor Circuit PCB

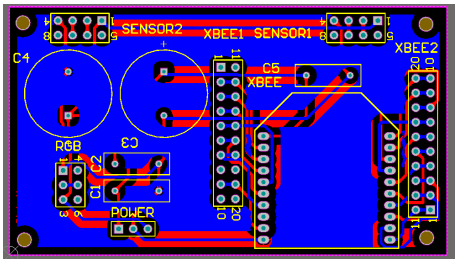

Figure 5.13 Bottom Layer of Sensor Circuit PCB

Controller Circuit Schematic and PCB

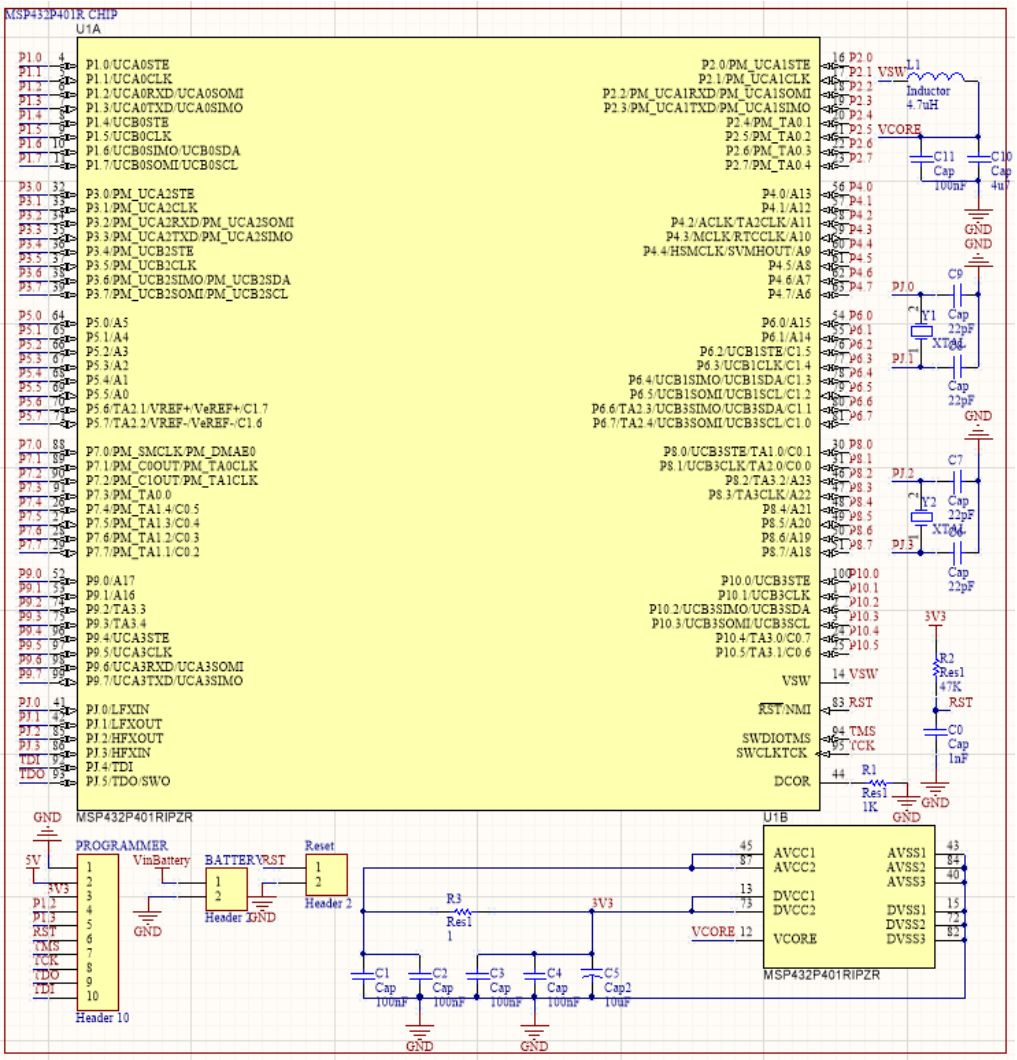

Figure 5.14 Controller Circuit Schematic Part 1

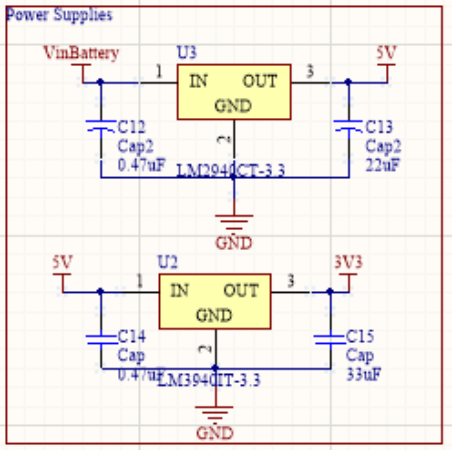

Figure 5.15 Controller Circuit Schematic Part 2

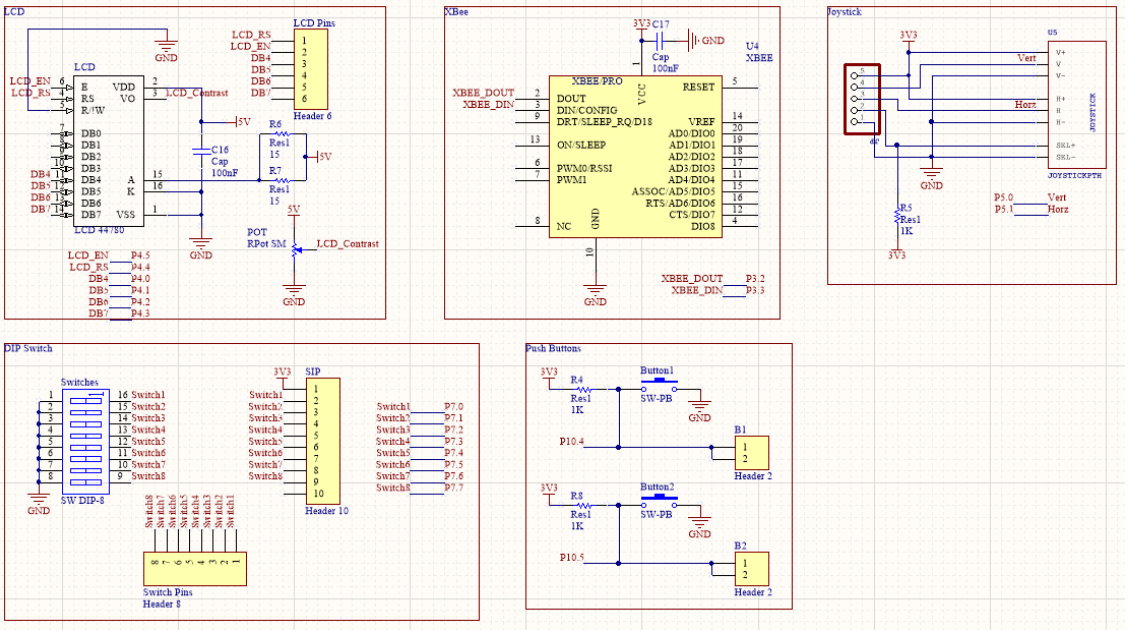

Figure 5.16 Controller Circuit Schematic Part 3

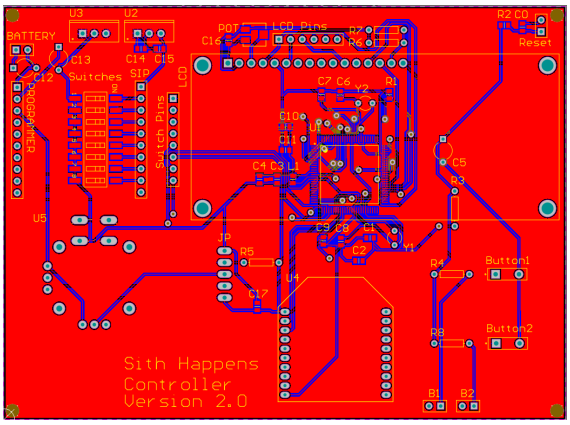

Figure 5.17 Top Layer of Controller Circuit PCB

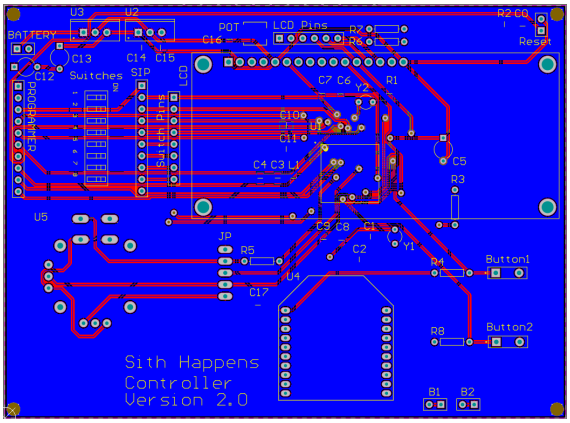

Figure 5.18 Bottom Layer of Controller Circuit PCB

Battery Charger Circuit Schematic and PCB

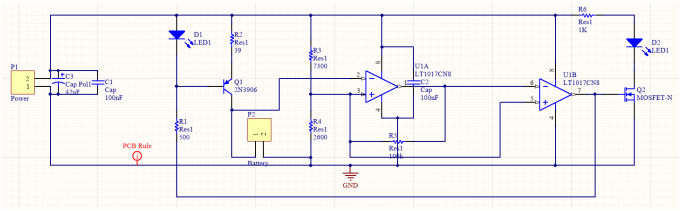

Figure 5.19 Battery Charger Circuit Schematic

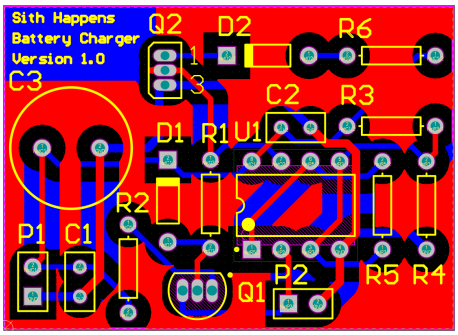

Figure 5.20 Top Layer of Battery Charger Circuit PCB

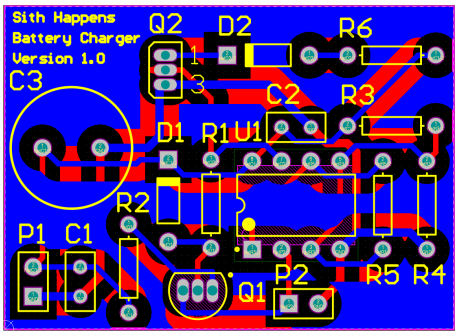

Figure 5.21 Bottom Layer of Battery Charger Circuit PCB

Controller Code – Main

```
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>

#include "UF_LCD.h"
#include "UART.h"
#include "ADC.h"

volatile uint16_t resultsBuffer[2];

volatile char button1 = 1;
volatile char button2 = 1;

void clk_init(void);
void GPIO_init(void);

int main(void)
{
    /* Halting the Watchdog */
    MAP_WDT_A_holdTimer();

    memset(resultsBuffer, 0x00, 2 * sizeof(uint16_t));

    lcd_init();
    ADC_init();
    UART_init();
    clk_init();
    GPIO_init();

    volatile int8_t analogX_upper = 0;
    volatile int8_t analogX_lower = 0;
    volatile int8_t analogY_upper = 0;
    volatile int8_t analogY_lower = 0;

    volatile int8_t analogX_data;
    volatile int8_t analogY_data;

    volatile uint8_t switches_upper = 0;
    volatile uint8_t switches_lower = 0;

    volatile uint8_t buttons = 0;
    lcdString("Welcome to R2-D2");

    char analogX[5];
    char analogY[5];
    char P7IN_String[5];
    char button1_string[1];
    char button2_string[1];
    char scroll = 0;
    char pos = 0;

    int8_t leftMotorEffort = 0;
    int8_t rightMotorEffort = 0;

    MAP_Interrupt_enableMaster();

    while(1) {
        resultsBuffer[0] = MAP_ADC14_getResult(ADC_MEM0); // analogY
        resultsBuffer[1] = MAP_ADC14_getResult(ADC_MEM1); // analogX

        analogY_data = resultsBuffer[0]*200/16383 - 100;
        analogX_data = -resultsBuffer[1]*200/16383 + 100;

        analogX_upper = (int8_t) ((analogX_data >> 4) & 0x0F); // 0b000
        analogX_lower = (int8_t) (analogX_data & 0x0F);
        analogX_lower = (analogX_lower | 0x20); // 0b001

        analogY_upper = (int8_t) ((analogY_data >> 4) & 0x0F);
        analogY_upper = (analogY_upper | 0x40); // 0b010
        analogY_lower = (int8_t) (analogY_data & 0x0F);
        analogY_lower = (analogY_lower | 0x60); // 0b011

        switches_upper = (uint8_t) ((P7IN >> 4) & 0x0F);
        switches_upper = (switches_upper | 0x80); // 0b100
        switches_lower = (uint8_t) (P7IN & 0x0F);
        switches_lower = (switches_lower | 0xA0); // 0b101

        buttons = (uint8_t) ((P10IN >> 4) & 0x03);
        buttons = (buttons | 0xC0); // 0b110

        MAP_UART_transmitData(EUSCI_A2_BASE, analogX_upper);
        MAP_UART_transmitData(EUSCI_A2_BASE, analogX_lower);
        MAP_UART_transmitData(EUSCI_A2_BASE, analogY_upper);
        MAP_UART_transmitData(EUSCI_A2_BASE, analogY_lower);
        MAP_UART_transmitData(EUSCI_A2_BASE, switches_upper);
        MAP_UART_transmitData(EUSCI_A2_BASE, switches_lower);
        MAP_UART_transmitData(EUSCI_A2_BASE, buttons);

        //lcd_command(0x02);

        if (analogX_data <= 20 && analogX_data >= -20 && analogY_data >= 20 && ((buttons & 0x01) == 0x00)) {
            lcd_command(0x02);
            lcdString("Moving Forward ");
            scroll = 0;
        }
        else if (analogX_data <= 20 && analogX_data >= -20 && analogY_data <= -20 && ((buttons & 0x01) == 0x00)) {
            lcd_command(0x02);
            lcdString("Moving Backward ");
            scroll = 0;
        }
        else if (analogX_data > 20 && analogY_data >= 20 && ((buttons & 0x01) == 0x00)) {
            lcd_command(0x02);
```

```
            lcdString("Forward Right ");
            scroll = 0;
        }
        else if (analogX_data < -20 && analogY_data >= 20 && ((buttons & 0x01) == 0x00)) {
            lcd_command(0x02);
            lcdString("Forward Left  ");
            scroll = 0;
        }
        else if (analogX_data < -20 && analogY_data <= -20 && ((buttons & 0x01) == 0x00)) {
            lcd_command(0x02);
            lcdString("Backward Left ");
            scroll = 0;
        }
        else if (analogX_data > 20 && analogY_data <= -20 && ((buttons & 0x01) == 0x00)) {
            lcd_command(0x02);
            lcdString("Backward Right ");
            scroll = 0;
        }
        else {
            if (scroll == 1 && pos <= 16) {
                lcd_command(0x1C);
                pos = pos + 1;
                volatile int delay = 0;
                while (delay < 500000)
                    delay++;
            }
            else {
                lcd_command(0x02);
                lcdString("               ");
                lcd_command(0x02);
                lcdString("R2-D2");
                scroll = 1;
                pos = 0;
                volatile int delay = 0;
                while (delay < 500000)
                    delay++;
            }
        }
    }

}

void clk_init(void)
{
    MAP_PCM_setCoreVoltageLevel(PCM_VCORE1);
    MAP_FlashCtl_setWaitState(FLASH_BANK0, 2);
    MAP_FlashCtl_setWaitState(FLASH_BANK1, 2);

    MAP_FPU_enableModule();

    MAP_CS_setDCOFrequency(48000000);

    MAP_CS_initClockSignal(CS_MCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
    MAP_CS_initClockSignal(CS_HSMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_2);
}

void GPIO_init(void)
{
    // GPIO inputs for buttons
    MAP_GPIO_setAsInputPin(GPIO_PORT_P10, GPIO_PIN4);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P10, GPIO_PIN5);

    // GPIO inputs for switches
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN7);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN6);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN5);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN4);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN3);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN2);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN1);
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN0);
}
```

Controller Code – ADC

```
#ifndef __ADC
#define __ADC
void ADC_init(void);
#endif
```

```
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <ti/devices/msp432p4xx/inc/msp432p401r_classic.h>
#include <ADC.h>

void ADC_init(void)
{
    MAP_ADC14_enableModule();
    MAP_ADC14_initModule(ADC_CLOCKSOURCE_MCLK, ADC_PREDIVIDER_1, ADC_DIVIDER_4, 0);

    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P5,
        GPIO_PIN0 | GPIO_PIN1, GPIO_TERTIARY_MODULE_FUNCTION);

    MAP_ADC14_configureMultiSequenceMode(ADC_MEM0, ADC_MEM1, true);
    MAP_ADC14_configureConversionMemory(ADC_MEM0,
        ADC_VREFPOS_AVCC_VREFNEG_VSS,
        ADC_INPUT_A5, false); // Pin 5.0 Vert
    MAP_ADC14_configureConversionMemory(ADC_MEM1,
        ADC_VREFPOS_AVCC_VREFNEG_VSS,
        ADC_INPUT_A4, false); // Pin 5.1 Horz
    MAP_ADC14_enableSampleTimer(ADC_AUTOMATIC_ITERATION);
    MAP_ADC14_enableConversion();
```

```
    MAP_ADC14_toggleConversionTrigger();

}
```

Controller Code – UART

```
#ifndef __UART
#define __UART
void UART_init(void);
#endif
```

```
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <ti/devices/msp432p4xx/inc/msp432p401r_classic.h>
#include <UART.h>

// assume 48 MHz
const eUSCI_UART_Config uartConfig =
{
    EUSCI_A_UART_CLOCKSOURCE_SMCLK,      // SMCLK Clock Source
    13,//3,                                 // BRDIV = 78
    0,//4,                                  // UCxBRF = 2
    0,                                   // UCxBRS = 0
    EUSCI_A_UART_NO_PARITY,              // No Parity
    EUSCI_A_UART_LSB_FIRST,              // MSB First
    EUSCI_A_UART_ONE_STOP_BIT,           // One stop bit
    EUSCI_A_UART_MODE,                   // UART mode
    EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION // Oversampling
};

void UART_init(void)
{
    /* Initialize TX and RX for UART */
    MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3,
        GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
    MAP_UART_initModule(EUSCI_A2_BASE, &uartConfig);
    MAP_UART_enableModule(EUSCI_A2_BASE);

}
```

Controller Code – LCD

```
#ifndef __UF_LCD
#define __UF_LCD

void delay(void);
void lcd_command(char);
void lcd_init(void);
void lcd_char(char);
void lcdString(char *string);

#endif
```

```
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <ti/devices/msp432p4xx/inc/msp432p401r_classic.h>
#include <UF_LCD.h>

char uf_lcd_temp;
char uf_lcd_temp2;
char uf_lcd_x;

void delay() {
    volatile int count = 0;
    while (count < 1001) {
        count++;
    }
}

void lcd_command(char uf_lcd_x){
    P4DIR = 0xFF;
   uf_lcd_temp = uf_lcd_x;
   P4OUT = 0x00;
   delay();
   uf_lcd_x = uf_lcd_x >> 4;
   uf_lcd_x = uf_lcd_x & 0x0F;
   uf_lcd_x = uf_lcd_x | 0x20;
   P4OUT = uf_lcd_x;
   delay();
   uf_lcd_x = uf_lcd_x & 0x0F;
   P4OUT = uf_lcd_x;
   delay();
   P4OUT = 0x00;
   delay();
   uf_lcd_x = uf_lcd_temp;
   uf_lcd_x = uf_lcd_x & 0x0F;
   uf_lcd_x = uf_lcd_x | 0x20;
   P4OUT = uf_lcd_x;
   delay();
   uf_lcd_x = uf_lcd_x & 0x0F;
   P4OUT = uf_lcd_x;
   delay();
}

void lcd_init(void){
   lcd_command(0x33);
   lcd_command(0x32);
```

```
   lcd_command(0x2C);
   lcd_command(0x0C);
   lcd_command(0x01);
}

void lcd_char(char uf_lcd_x){
   P4DIR = 0xFF;
   uf_lcd_temp = uf_lcd_x;
   P4OUT = 0x10;
   delay();
   uf_lcd_x = uf_lcd_x >> 4;
   uf_lcd_x = uf_lcd_x & 0x0F;
   uf_lcd_x = uf_lcd_x | 0x30;
   P4OUT = uf_lcd_x;
   delay();
   uf_lcd_x = uf_lcd_x & 0x1F;
   P4OUT = uf_lcd_x;
   delay();
   P4OUT = 0x10;
   delay();
   uf_lcd_x = uf_lcd_temp;
   uf_lcd_x = uf_lcd_x & 0x0F;
   uf_lcd_x = uf_lcd_x | 0x30;
   P4OUT = uf_lcd_x;
   delay();
   uf_lcd_x = uf_lcd_x & 0x1F;
   P4OUT = uf_lcd_x;
   delay();
}

void lcdString(char *string)
{
    while (*string != 0) {
        lcd_char(*string);
        *string++;
    }
}
```

Robot Code

```
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

/* Standard Includes */
#include <stdint.h>
#include <stdbool.h>
#include <file.h>
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

#include <third_party/fatfs/ffcio.h>

#include <ti/display/Display.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/SDFatFS.h>
/*********************************************************************************************************
 ************************************** LIGHTS ***********************************************************
 *********************************************************************************************************/
const eUSCI_SPI_MasterConfig spiMasterConfigLED=
{
 EUSCI_B_SPI_CLOCKSOURCE_SMCLK,
 12000000,
 5333333,
 EUSCI_B_SPI_MSB_FIRST,
 EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT,
 EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH,
 EUSCI_B_SPI_3PIN
};
void initStrip(){//SPI for DAC
{
   GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P10, GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);
  SPI_initMaster(EUSCI_B3_BASE, &spiMasterConfigLED);
  SPI_enableModule(EUSCI_B3_BASE);
}
void goBlue(){
   SPI_disableInterrupt(EUSCI_B3_BASE, EUSCI_B_SPI_TRANSMIT_INTERRUPT);  // disable interrupts
   // Send 0x00 for G
   uint8_t lights;
   for(lights = 0; lights<4;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
   }
   for(lights=0;lights<11;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xEF;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xFF;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
   }
   for(lights = 0; lights<4;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xFF;
   }
   SPI_enableInterrupt(EUSCI_B3_BASE, EUSCI_B_SPI_TRANSMIT_INTERRUPT);  // enable interrupts
}
void goRed(){
   SPI_disableInterrupt(EUSCI_B3_BASE, EUSCI_B_SPI_TRANSMIT_INTERRUPT);  // disable interrupts
   // Send 0x00 for G
   uint8_t lights;
   for(lights = 0; lights<4;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
   }
   for(lights=0;lights<11;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xEF;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xFF;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
   }
   for(lights = 0; lights<4;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xFF;
   }
   SPI_enableInterrupt(EUSCI_B3_BASE, EUSCI_B_SPI_TRANSMIT_INTERRUPT);  // enable interrupts
}
void goGreen(){
   SPI_disableInterrupt(EUSCI_B3_BASE, EUSCI_B_SPI_TRANSMIT_INTERRUPT);  // disable interrupts
   // Send 0x00 for G
   uint8_t lights;
   for(lights = 0; lights<4;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
   }
   for(lights=0;lights<11;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xEF;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xFF;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0x00;
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
   }
   for(lights = 0; lights<4;lights++)
   {
      while (!(EUSCI_B3->IFG & EUSCI_B_IFG_TXIFG));
      EUSCI_B3->TXBUF = 0xFF;
   }
   SPI_enableInterrupt(EUSCI_B3_BASE, EUSCI_B_SPI_TRANSMIT_INTERRUPT);  // enable interrupts
}
/*********************************************************************************************************
 ************************************** XBEE ***********************************************************
 *********************************************************************************************************/
const eUSCI_UART_Config uartConfig =
{
   EUSCI_A_UART_CLOCKSOURCE_SMCLK,        // SMCLK Clock Source
   3,                                     // BRDIV = 78
   4,                                     // UCxBRF = 2
   0,                                     // UCxBRS = 0
   EUSCI_A_UART_NO_PARITY,                // No Parity
   EUSCI_A_UART_LSB_FIRST,                // MSB First
   EUSCI_A_UART_ONE_STOP_BIT,             // One stop bit
   EUSCI_A_UART_MODE,                     // UART mode
   EUSCI_A_UART_OVERSAMPLING_BAUDRATE_GENERATION  // Oversampling
};

volatile int8_t analogX_data = 0;
volatile int8_t analogY_data = 0;

volatile int8_t analogX_upper = 0;
volatile int8_t analogX_lower = 0;
volatile int8_t analogY_upper = 0;
volatile int8_t analogY_lower = 0;

volatile int8_t analogX_data;
volatile int8_t analogY_data;

volatile uint8_t switches_upper = 0;
volatile uint8_t switches_lower = 0;
volatile uint8_t switches_data = 0;

volatile uint8_t buttons = 0;

volatile uint8_t byte = 0;

int16_t leftMotorEffort = 0;
int16_t rightMotorEffort = 0;
uint16_t soundCounter = 0;
char flag = 0;

void init_clk(void);
volatile uint16_t choice = 0;
volatile uint16_t steve = 0;
volatile int rising_edge;
volatile int meas1 = 0;
volatile int meas2 = 0;
volatile int prev_rising = 0;
volatile int wait = 0;
char obstacle_detected = 0;


/*********************************************************************************************************************************************
 ************************************************** US Sensor Controls ***********************************************************************
 *********************************************************************************************************************************************/
volatile uint16_t timer;
/* Timer_A Continuous Mode Configuration Parameter */
const Timer_A_ContinuousModeConfig continuousModeConfig =
{
      TIMER_A_CLOCKSOURCE_SMCLK,          // SMCLK Clock Source
      TIMER_A_CLOCKSOURCE_DIVIDER_64,     // SMCLK/1 = 3MHz
      TIMER_A_TAIE_INTERRUPT_DISABLE,     // Disable Timer ISR
      TIMER_A_SKIP_CLEAR                  // Skup Clear Counter
};

/* Timer_A Capture Mode Configuration Parameter */
const Timer_A_CaptureModeConfig captureModeConfig =
{
      TIMER_A_CAPTURECOMPARE_REGISTER_1,      // CC Register 1
      TIMER_A_CAPTUREMODE_RISING_AND_FALLING_EDGE,        // Rising Edge and falling
      TIMER_A_CAPTURE_INPUTSELECT_CCIxA,      // CCIxA Input Select
      TIMER_A_CAPTURE_SYNCHRONOUS,            // Synchronized Capture
      TIMER_A_CAPTURECOMPARE_INTERRUPT_ENABLE,  // Enable interrupt
      TIMER_A_OUTPUTMODE_OUTBITVALUE          // Output bit value
};


/*********************************************************************************
 ****************************** PWM Configurations *******************************
 *********************************************************************************/
// Pin 7.7
extern Timer_A_PWMConfig pwmConfig_LeftBack =
{
 TIMER_A_CLOCKSOURCE_SMCLK,
 TIMER_A_CLOCKSOURCE_DIVIDER_1,
 24000,
 TIMER_A_CAPTURECOMPARE_REGISTER_1,
 TIMER_A_OUTPUTMODE_RESET_SET,
 6000
};
// Pin 7.6
extern Timer_A_PWMConfig pwmConfig_RightBack =
```

```
{
TIMER_A_CLOCKSOURCE_SMCLK,
TIMER_A_CLOCKSOURCE_DIVIDER_1,
24000,
TIMER_A_CAPTURECOMPARE_REGISTER_2,
TIMER_A_OUTPUTMODE_RESET_SET,
6000
};
// Pin 7.5
extern Timer_A_PWMConfig pwmConfig_LeftFront =
{
TIMER_A_CLOCKSOURCE_SMCLK,
TIMER_A_CLOCKSOURCE_DIVIDER_1,
24000,
TIMER_A_CAPTURECOMPARE_REGISTER_3,
TIMER_A_OUTPUTMODE_RESET_SET,
12000
};
// Pin 7.4
extern Timer_A_PWMConfig pwmConfig_RightFront =
{
TIMER_A_CLOCKSOURCE_SMCLK,
TIMER_A_CLOCKSOURCE_DIVIDER_1,
24000,
TIMER_A_CAPTURECOMPARE_REGISTER_4,
TIMER_A_OUTPUTMODE_RESET_SET,
12000
};

Timer_A_PWMConfig* pointer_LeftBack = &pwmConfig_LeftBack;
Timer_A_PWMConfig* pointer_RightBack = &pwmConfig_RightBack;
Timer_A_PWMConfig* pointer_LeftFront = &pwmConfig_LeftFront;
Timer_A_PWMConfig* pointer_RightFront = &pwmConfig_RightFront;

void randomMovement(uint16_t choice, int gogo0, int gogo1)
{
  if(choice%16 == 0) //right
  {
     pointer_LeftBack->dutyCycle = 0;
     pointer_LeftFront->dutyCycle = 12000;
     pointer_RightBack->dutyCycle = 0;
     pointer_RightFront->dutyCycle = 12000;
     Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
     Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
     Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
     Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
  }
  else if(choice%16 == 1) //right
     {
        pointer_LeftBack->dutyCycle = 0;
        pointer_LeftFront->dutyCycle = 12000;
        pointer_RightBack->dutyCycle = 0;
        pointer_RightFront->dutyCycle = 12000;
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
     }
  else if(choice%16 == 2) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 0;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 18000;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 3) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 0;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 0;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 4) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 18000;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 0;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 5) //right
     {
        pointer_LeftBack->dutyCycle = 0;
        pointer_LeftFront->dutyCycle = 18000;
        pointer_RightBack->dutyCycle = 0;
        pointer_RightFront->dutyCycle = 0;
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
     }
  else if(choice%16 == 6) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 18000;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 18000;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 7) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 18000;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 18000;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 8) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 0;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 0;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 9) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 18000;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 0;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
  else if(choice%16 == 10) //right
     {
        pointer_LeftBack->dutyCycle = 0;
        pointer_LeftFront->dutyCycle = 12000;
        pointer_RightBack->dutyCycle = 0;
        pointer_RightFront->dutyCycle = 12000;
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
        Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
     }
  else if(choice%16 == 11) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 12000;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 12000;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
     else if(choice%16 == 12) //right
           {
              pointer_LeftBack->dutyCycle = 0;
              pointer_LeftFront->dutyCycle = 0;
              pointer_RightBack->dutyCycle = 0;
              pointer_RightFront->dutyCycle = 18000;
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
           }
     else if(choice%16 == 13) //right
           {
              pointer_LeftBack->dutyCycle = 0;
              pointer_LeftFront->dutyCycle = 12000;
              pointer_RightBack->dutyCycle = 0;
              pointer_RightFront->dutyCycle = 0;
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
           }
     else if(choice%16 == 14) //right
           {
              pointer_LeftBack->dutyCycle = 0;
              pointer_LeftFront->dutyCycle = 0;
              pointer_RightBack->dutyCycle = 0;
              pointer_RightFront->dutyCycle = 0;
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
              Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
           }
     else if(choice%16 == 15) //right
        {
           pointer_LeftBack->dutyCycle = 0;
           pointer_LeftFront->dutyCycle = 12000;
           pointer_RightBack->dutyCycle = 0;
           pointer_RightFront->dutyCycle = 0;
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
           Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
        }
else //left
{
```

```
    pointer_LeftBack->dutyCycle = 0;
    pointer_LeftFront->dutyCycle = 0;
    pointer_RightBack->dutyCycle = 0;
    pointer_RightFront->dutyCycle = 0;
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
  }
}

/*******************************************************************************
 ********************** Peripheral Initializations ****************************
 *******************************************************************************/

void initTimer32()
{
  Timer32_initModule(TIMER32_0_BASE, TIMER32_PRESCALER_1, TIMER32_32BIT, TIMER32_PERIODIC_MODE);
  Timer32_setCount(TIMER32_0_BASE, 3000000);
  Interrupt_enableInterrupt(TIMER32_0_INTERRUPT);
  Timer32_enableInterrupt(TIMER32_0_BASE);
}

void initTimer32_R2D2()
{
  Timer32_initModule(TIMER32_1_BASE, TIMER32_PRESCALER_1, TIMER32_32BIT, TIMER32_PERIODIC_MODE);
  Timer32_setCount(TIMER32_1_BASE, 1200);
  Interrupt_enableInterrupt(TIMER32_1_INTERRUPT);
 // Timer32_enableInterrupt(TIMER32_1_BASE);
}
const eUSCI_SPI_MasterConfig spiMasterConfig=
{
EUSCI_B_SPI_CLOCKSOURCE_SMCLK,
3000000,
1000000,
EUSCI_B_SPI_MSB_FIRST,
EUSCI_B_SPI_PHASE_DATA_CHANGED_ONFIRST_CAPTURED_ON_NEXT,
EUSCI_B_SPI_CLOCKPOLARITY_INACTIVITY_HIGH,
EUSCI_B_SPI_3PIN
};
void initSPI()//SPI for DAC
{
  GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P6, GPIO_PIN3 | GPIO_PIN4 | GPIO_PIN5, GPIO_PRIMARY_MODULE_FUNCTION);
  GPIO_setAsOutputPin(GPIO_PORT_P6, GPIO_PIN2); //For CS
  SPI_initMaster(EUSCI_B1_BASE, &spiMasterConfig);
  SPI_enableModule(EUSCI_B1_BASE);
}
void writeDAC(uint16_t data)
{
  GPIO_setOutputLowOnPin(GPIO_PORT_P6, GPIO_PIN2);
  uint8_t high = (0x90 | ((data>>6) & 0xff));
  SPI_transmitData(EUSCI_B1_BASE, high);
  uint8_t low = ((data<<2) & 0xff);
  SPI_transmitData(EUSCI_B1_BASE, low);
  GPIO_setOutputHighOnPin(GPIO_PORT_P6, GPIO_PIN2);
}
volatile uint16_t position = 0;
//![Simple GPIO Config]
int main(void)
{
  /* Halting the Watchdog */
  MAP_WDT_A_holdTimer();
  /* Setting DCO to 12MHz */
  MAP_CS_setDCOCenteredFrequency(CS_DCO_FREQUENCY_12);
  MAP_CS_initClockSignal(CS_SMCLK, CS_DCOCLK_SELECT, CS_CLOCK_DIVIDER_1);
  /* Initialize TX and RX for UART */
  initStrip();
  goBlue();
  /* Configuring P2.4 as peripheral input for capture */
  GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P2, GPIO_PIN4, GPIO_PRIMARY_MODULE_FUNCTION);
  /* P2.5 Trigger Pin for Ultra Sonic */
  GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN5);
  GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN5);
  /* Test signal for US */
  GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN5);
  GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5);
  MAP_GPIO_setAsPeripheralModuleFunctionInputPin(GPIO_PORT_P3,
          GPIO_PIN2 | GPIO_PIN3, GPIO_PRIMARY_MODULE_FUNCTION);

  MAP_UART_initModule(EUSCI_A2_BASE, &uartConfig);

  MAP_UART_enableModule(EUSCI_A2_BASE);

  /* Enabling interrupts */
  MAP_UART_enableInterrupt(EUSCI_A2_BASE, EUSCI_A_UART_RECEIVE_INTERRUPT);
  MAP_Interrupt_enableInterrupt(INT_EUSCIA2);

  // PWM for Motors
  MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P7, GPIO_PIN4, GPIO_PRIMARY_MODULE_FUNCTION);
  MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P7, GPIO_PIN5, GPIO_PRIMARY_MODULE_FUNCTION);
  MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P7, GPIO_PIN6, GPIO_PRIMARY_MODULE_FUNCTION);
  MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P7, GPIO_PIN7, GPIO_PRIMARY_MODULE_FUNCTION);

  /* Configuring Continuous Mode */
  MAP_Timer_A_configureContinuousMode(TIMER_A1_BASE, &continuousModeConfig);

  MAP_Interrupt_enableInterrupt(INT_TA1_N);
  initSPI();
  initTimer32();
  initTimer32_R2D2();

  /* Configuring Capture Mode */
  Timer_A_initCapture(TIMER_A0_BASE, &captureModeConfig);
  /* Configuring Continuous Mode */
  Timer_A_configureContinuousMode(TIMER_A0_BASE, &continuousModeConfig);
  Interrupt_enableInterrupt(INT_TA0_N);

  MAP_Interrupt_enableMaster();

  /* Starting the Timer_A0 in continuous mode */
  Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_CONTINUOUS_MODE);
  MAP_Timer_A_startCounter(TIMER_A1_BASE, TIMER_A_CONTINUOUS_MODE);
  MAP_Timer32_startTimer(TIMER32_0_BASE, false);
  MAP_Timer32_startTimer(TIMER32_1_BASE, false);

  while (1)
  {
    goBlue();
    if(soundCounter == 5000)
    {
      Timer32_enableInterrupt(TIMER32_1_BASE);
      soundCounter = 0;
    }
    soundCounter++;
    switches_data = (((switches_upper & 0x0F) << 4) + (switches_lower & 0x0F));
    analogY_data = (((analogY_upper & 0x0F) << 4) + (analogY_lower & 0x0F));
    analogX_data = (((analogX_upper & 0x0F) << 4) + (analogX_lower & 0x0F));

/*******************************************************************************************************************************************************
 ********************************************************** MANUAL MODE ********************************************************************************
 *******************************************************************************************************************************************************/
    if ((switches_data & 0x07) == 0x01)
    {
    leftMotorEffort = (int16_t) analogY_data + (int16_t) analogX_data;
    rightMotorEffort = (int16_t) analogY_data - (int16_t) analogX_data;
    if ((abs(analogY_data) >= 20 | | abs(analogX_data) >= 20) && ((buttons & 0x01) == 0x00))
    {
      if (leftMotorEffort > 5)
      {
        pointer_LeftFront->dutyCycle = 120*leftMotorEffort;
        pointer_LeftBack->dutyCycle = 0;
      }
      else if (leftMotorEffort < -5)
      {
        pointer_LeftBack->dutyCycle = 120*abs(leftMotorEffort);
        pointer_LeftFront->dutyCycle = 0;
      }
      else
      {
        pointer_LeftFront->dutyCycle = 0;
        pointer_LeftBack->dutyCycle = 0;
      }

      if (rightMotorEffort > 5)
      {
        pointer_RightFront->dutyCycle = 120*rightMotorEffort;
        pointer_RightBack->dutyCycle = 0;
      }
      else if (rightMotorEffort < -5)
      {
        pointer_RightBack->dutyCycle = 120*abs(rightMotorEffort);
        pointer_RightFront->dutyCycle = 0;
      }
      else
      {
        pointer_RightFront->dutyCycle = 0;
        pointer_RightBack->dutyCycle = 0;
      }
    }
    else
    {
      pointer_LeftBack->dutyCycle = 0;
      pointer_LeftFront->dutyCycle = 0;
      pointer_RightBack->dutyCycle = 0;
      pointer_RightFront->dutyCycle = 0;
    }
    //goBlue();
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
    Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
    volatile int count = 0;
    while (count < 2501)
        count++;
    }//end of manual mode
/*******************************************************************************************************************************************************
 ********************************************************** AUTONOMOUS MODE ****************************************************************************
 *******************************************************************************************************************************************************/
    else if ((switches_data & 0x07) == 0x02)
    {
      Timer32_disableInterrupt(TIMER32_1_BASE);
      goGreen();
      //GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN5);
      volatile int count = 0;
      //while (count < 100000)
      //   count++;
      GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN5);
      //count = 0;
      while (count < 10)
          count++;
      GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN5);
      volatile int waiting = 0;
      srand(time(0));

      steve++;

      volatile int gogo0 = (rand()%12000)+12000;
      volatile int gogo1 = (rand()%12000)+12000;
      if (obstacle_detected == 1) // Obstacle
      {

          GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN5);
          //move back
          pointer_LeftBack->dutyCycle = 12000;
```

```
            pointer_LeftFront->dutyCycle = 0;
            pointer_RightBack->dutyCycle = 12000;
            pointer_RightFront->dutyCycle = 0;
            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);

            for(waiting=0;waiting<300000;waiting++);
            obstacle_detected = 0;
        }
        else // No obstacle
        {
            GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5);
            choice++;
            randomMovement((uint16_t)choice, gogo0, gogo1);

            for(waiting=0;waiting<300000;waiting++)
            {
              if(obstacle_detected==1)
              {
                 //randomMovement(choice, gogo0, gogo1);
                 pointer_LeftBack->dutyCycle = 12000;
                            pointer_LeftFront->dutyCycle = 0;
                            pointer_RightBack->dutyCycle = 12000;
                            pointer_RightFront->dutyCycle = 0;
                            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftFront);
                            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightBack);
                            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_LeftBack);
                            Timer_A_generatePWM(TIMER_A1_BASE, &pwmConfig_RightFront);
                 break;
              }
            }
            while(waiting<100000)
              waiting++;
        }
    }
/*****************************************************************************************************************************************************
 ********************************************************** PARTY MODE ********************************************************************************
 *****************************************************************************************************************************************************/
    else if ((switches_data & 0x07) == 0x04)
    {
        goRed();
    }
    else
    {

    }
  }//end of while
}//end of main

void EUSCIA2_IRQHandler(void)
{

    uint32_t status = MAP_UART_getEnabledInterruptStatus(EUSCI_A2_BASE);
    uint8_t byte = UCA2RXBUF;

    if(status & EUSCI_A_UART_RECEIVE_INTERRUPT_FLAG)
    {
      if ((byte & 0xE0) == 0x00) {
        analogX_upper = (byte & 0x0F);
      }
      else if ((byte & 0xE0) == 0x20) {
        analogX_lower = (byte & 0x0F);
      }
      else if ((byte & 0xE0) == 0x40) {
        analogY_upper = (byte & 0x0F);
      }

      else if ((byte & 0xE0) == 0x60) {
        analogY_lower = (byte & 0x0F);
      }
      else if ((byte & 0xE0) == 0x80) {
        switches_upper = (byte & 0x0F);
      }
      else if ((byte & 0xE0) == 0xA0) {
        switches_lower = (byte & 0x0F);
      }
      else if ((byte & 0xE0) == 0xC0) {
        buttons = (byte & 0x03);
      }

      else {
      }
    }
    EUSCI_A2->IFG &= 0xFFFE;

}

void T32_INT1_IRQHandler(void)
{
    Timer32_clearInterruptFlag(TIMER32_0_BASE);
}
void T32_INT2_IRQHandler(void)
{
    if(position == 13125)
    {
      position = 0;
      Timer32_disableInterrupt(TIMER32_1_BASE);
    }
    writeDAC(r2d2[position++]);
    Timer32_clearInterruptFlag(TIMER32_1_BASE);
}
void TA0_N_IRQHandler(void)
 {
    if ((P2IN & 0x10) != 0)
      rising_edge = 1;
    else
      rising_edge = 0;
    if (rising_edge == 1) // Start
    {
      meas1 = Timer_A_getCaptureCompareCount(TIMER_A0_BASE, TIMER_A_CAPTURECOMPARE_REGISTER_1);
    }
    else
    {
      meas2 = Timer_A_getCaptureCompareCount(TIMER_A0_BASE, TIMER_A_CAPTURECOMPARE_REGISTER_1);
    }
    if (rising_edge == 0) {
      if (meas2 >= meas1) {
        timer = meas2 - meas1;
      }
      else {
        timer = meas2 + (0xFFFF - meas1);
      }
    }
    if (timer > 400)
      obstacle_detected = 0;
    else
    {
      obstacle_detected = 1;
    }
    Timer_A_clearCaptureCompareInterrupt(TIMER_A0_BASE, TIMER_A_CAPTURECOMPARE_REGISTER_1);
}
```