Embedded Microcomputers Final Report

Andrew Pair, Timmy Phan, Alexander Tu

ECE 4437: Embedded Microcomputer Systems

Dr. Hung “Harry” Le

27 November 2023

Introduction

This report encapsulated our team's endeavor in constructing and coding a maze-traversing robot utilizing a TM4C123GH6PM Tiva C launchpad. The robot includes 5 main milestones: bluetooth functionality, infrared distance sensors for the front and right of the robot, motors that move the robot, a PID control algorithm, and a light sensor to detect black lines. First off, the addition of a bluetooth module allows our robot to be able to respond to commands wirelessly and allows our robot to run even when not connected to a pc. Secondly, the infrared sensors measure the distance from the front and the right side of the robot through ADC conversions of signals received from the sensors. The distances measured are in centimeters and will be used in PID calculations. Thirdly, the motors run by using the built in PWM of the Tiva C launchpad. The PWM will send pulses which will power the motor. Fourthly, the PID control algorithm ensures that the robot stays centered in the maze using the distances measured from the infrared distance sensors and is performed every 50 ms. The PID will modify the PWM pulses to correct the robot until the robot is centered in the maze track. Finally, the light sensor detects when a black line is crossed and whether the black line is thin or thick. Crossing the first thin black line starts data collection and crossing the second thin black line stops data collection. Crossing a thick black line outputs the run time, stops the robot, and has the red LED blink for 1 minute before completely shutting down the robot.

Block Diagram

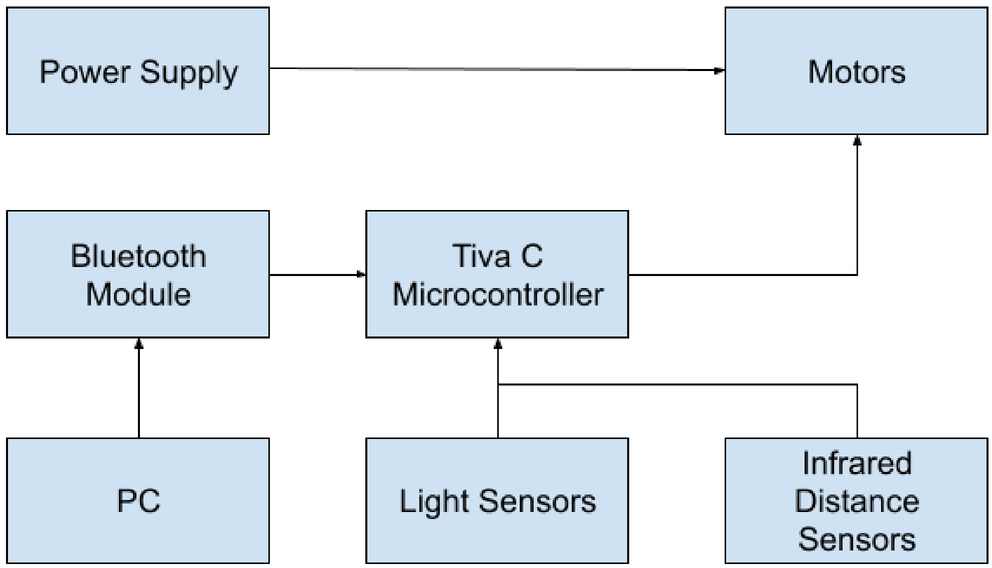

Figure 1. Block Diagram

Wiring Diagram

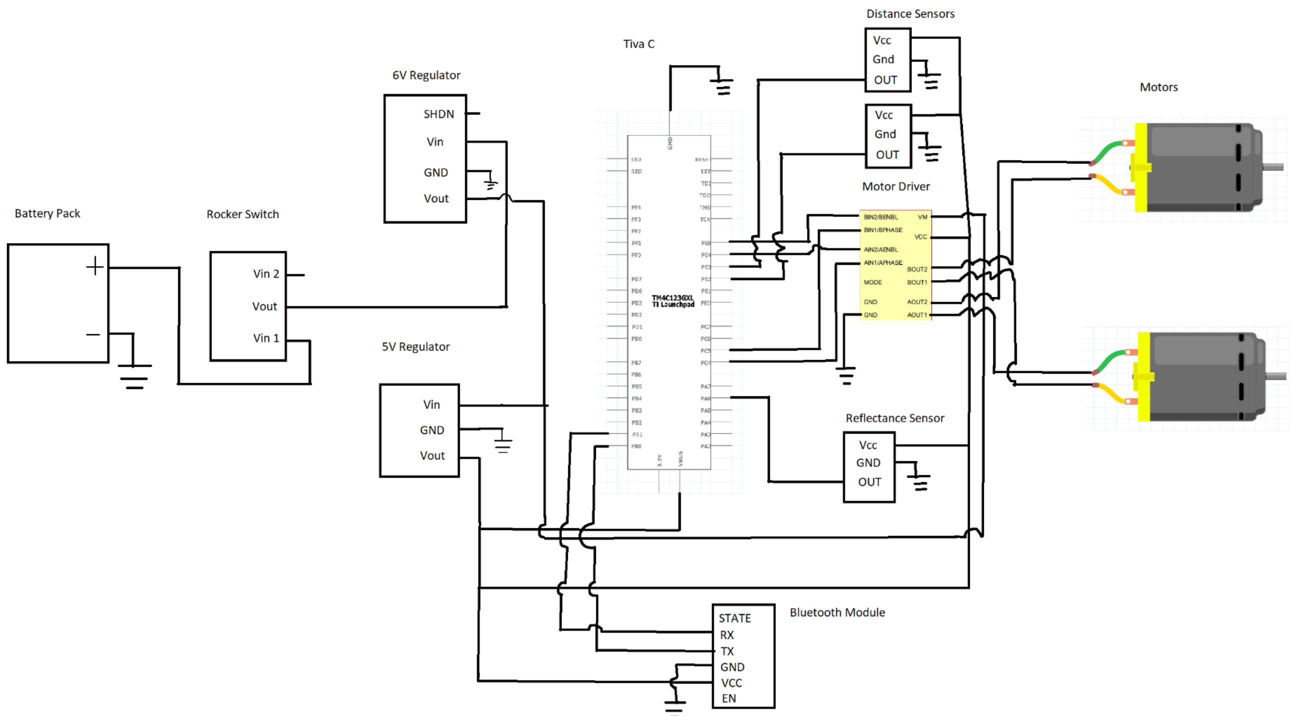

Figure 2. Wiring Diagram

Pseudo Code

```
# Constants
    TIMER_INTERVAL = 50  # in milliseconds
    MAX_BUFFER_SIZE = 20
    STACK_SIZE = 1024

# Global variables
    runtime = 0
    BUFFER1[MAX_BUFFER_SIZE]
    BUFFER2[MAX_BUFFER_SIZE]

# Command structure
    Command: command, function

# List of supported commands and their functions
    commandList: { "fr", cmd_RIGHTFORWARDFAST }, { "bw", cmd_BACK }, { "go",
        cmd_START } ...

# Task handles
    bluetoothTask, frontSensorTask, sideSensorTask, centralControlTask

# Semaphores
    frontSensorDataSemaphore
```

sideSensorDataSemaphore

Semaphore0

Semaphore1

Timer interrupt handlers

PIDInterruptHandler():

Signal semaphore for PID task

BlackLineInterruptHandler():

Signal semaphore for BlackLineTask

Initialization

ConfigureTimer2A()

ConfigureTimer1A()

btConfig()

ADCConfigure()

PWMConfigure()

Main loop

Set system clock with a division of 4 and a crystal clock of 16 mHz

Configure the PWM

Configure the bluetooth

Configure the ADC

```
    Configure the timers
    Read user input and execute the command entered
    Start the clock
    Start the RTOS scheduler

# Task functions
    BluetoothTask(arg0, arg1):
        Wait for UART characters
        Parse received command
        Execute corresponding function

    FrontSensorTask(arg0, arg1):
        Measure front sensor data
        Update global variables
        Signal semaphore for CentralControlTask

    SideSensorTask(arg0, arg1):
        Measure side sensor data
        Update global variables
        Signal semaphore for CentralControlTask

    CentralControlTask(arg0, arg1):
        Wait for semaphores from FrontSensorTask and SideSensorTask
```

```
        Implement central control logic
        Update PWM and motor control

# Other functions
    BufferCall(output):
        Add output to BUFFER1 or BUFFER2 based on bufferToggle
    SwitchBuffer():
        Toggle bufferToggle between BUFFER1 and BUFFER2
        LED turns yellow during ping buffer and blue during pong buffer

    PingPongSWIHandler():
        When buffer is full, switches buffers and prints contents of full buffer

    reflectance():
        Checks the number of cycles returned from the light sensor
        If cycles > 100, black line detected
        If cycles <= 100, no black line detected

    blackLineTask():
        If thin black line detected, increment variable “thinline”
        If thick black line detected, stop robot, output run time, and flash LED red for a
            minute before  completely shutting down robot
```

ComputePID():

Implement PID control logic based on sensor readings

Update motor PWM based on PID output every 50 ms

U-turn if dead end

If thinline == 1, start collecting data

If thin line == 2, stop collecting data

Real Code with comments

```
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <inttypes.h>
#include <string.h>

#include "inc/hw_ints.h"
#include "inc/hw_types.h"
#include "inc/hw_memmap.h"

#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/uart.h"
#include "driverlib/adc.h"
#include <driverlib/pwm.h>
#include <ti/drivers/GPIO.h>
#include <ti/drivers/PWM.h>
#include "driverlib/adc.h"
#include "utils/uartstdio.h"
#include "Board.h"
```

```
#include <math.h>

#include <ti/sysbios/knl/Swi.h>

#include <time.h>



#include <xdc/runtime/System.h>

#include <ti/sysbios/BIOS.h>

#include <ti/sysbios/knl/Task.h>

#include <ti/sysbios/knl/Semaphore.h>

#include "driverlib/timer.h"


#include "utils/uartstdio.c"


extern const Swi_Handle swi0;

extern const ti_sysbios_knl_Semaphore_Handle Semaphore0;

extern const ti_sysbios_knl_Semaphore_Handle Semaphore1;



//for timer task

double runtime = 0;


// Task stack sizes

#define STACKSIZE   1024
```

```
#define BUFFERSIZE 20

char *BUFFER1[BUFFERSIZE];
char *BUFFER2[BUFFERSIZE];

// Define the list of supported commands and their corresponding functions

void UARTPrintInt(uint32_t num);
void cmd_RIGHTCUSTOMSPEED(double PID);
void cmd_LEFTCUSTOMSPEED(double PID);
void cmd_FORWARD(const char* args);
void cmd_BACK(const char* args);
void cmd_STOP(const char* args);
void cmd_ERR(const char* args);
void cmd_ALLFORWARDFAST(const char* args);
void cmd_RIGHTFORWARDFAST(const char* args);
void cmd_RIGHTFORWARDSLOW(const char* args);
void cmd_LEFTFORWARDSLOW(const char* args);
void cmd_LEFTFORWARDFAST(const char* args);
void cmd_RIGHTSTOP(const char* args);
void cmd_LEFTSTOP(const char* args);
```

```
void cmd_START(const char* args);
void ComputePID();

// Define a structure to hold command information
typedef struct {
	const char* command;
	void (*function)(const char*);
} Command;

// Define the list of supported commands and their corresponding functions
Command commandList[] = {
	{ "fr", cmd_RIGHTFORWARDFAST },
	{ "bw", cmd_BACK },
	{ "st", cmd_STOP },
	{ "ER", cmd_ERR },
	{ "rs", cmd_RIGHTFORWARDSLOW},
	{ "ls", cmd_LEFTFORWARDSLOW},
	{ "lf", cmd_LEFTFORWARDFAST},
	{ "er", cmd_RIGHTSTOP},
	{ "el", cmd_LEFTSTOP},
	{ "go", cmd_START }
};
```

```
clock_t start, end;

double cpu_time_used;



void ConfigureTimer2A(){//timer for PID controller

	// Timer 2 setup code		50 MS

	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);			// enable Timer 2
periph clks

	TimerConfigure(TIMER2_BASE, TIMER_CFG_A_PERIODIC |
TIMER_CFG_SPLIT_PAIR);		// cfg Timer 2 mode - periodic


	uint32_t ui32Period = (SysCtlClockGet() /1000);			// period = 1/20th of a second
AKA 50MS

	TimerLoadSet(TIMER2_BASE, TIMER_A, ui32Period);			// set Timer 2 period

	TimerPrescaleSet(TIMER2_BASE, TIMER_A, 50-1);

	TimerIntEnable(TIMER2_BASE, TIMER_TIMA_TIMEOUT);	// enables Timer 2 to
interrupt CPU


	TimerEnable(TIMER2_BASE, TIMER_A);			// enable Timer 2
```

```
	//UARTprintf("Timer2A \n");
}

void ConfigureTimer1A(){//timer for reflectance sensor
	SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER1);
	TimerConfigure(TIMER1_BASE, TIMER_CFG_A_PERIODIC |
TIMER_CFG_SPLIT_PAIR);

	uint32_t ui32Period = (SysCtlClockGet() / 1000);		// period = 1/100th of a
second AKA 10MS
	TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period);		// set Timer 2 period
	TimerPrescaleSet(TIMER1_BASE, TIMER_A, 10 - 1);
	TimerIntEnable(TIMER1_BASE, TIMER_TIMA_TIMEOUT);	// enables Timer 1 to
interrupt CPU

	TimerEnable(TIMER1_BASE, TIMER_A);
	//UARTprintf("Timer1A \n");
}

int reflectance(){ //Checks value of light sensor
	//UARTPrintf("Reflectance");
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
	GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
```

```
                    GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);


        int cycles = 0;

        GPIOPinTypeGPIOOutput(GPIO_PORTA_BASE, GPIO_PIN_6);

        GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_6, GPIO_PIN_6);

        SysCtlDelay(SysCtlClockGet()/80000); //12.5 us


        GPIOPinTypeGPIOInput(GPIO_PORTA_BASE, GPIO_PIN_6);


        while(GPIOPinRead(GPIO_PORTA_BASE, GPIO_PIN_6) == GPIO_PIN_6){

        SysCtlDelay(SysCtlClockGet()/1000000); //1 us

        cycles++;

        }

        if(cycles > 100){//on black line

                return 1;

        }

        else if (cycles <= 100)//off black line

        {

        return 0;

        }


}

int greenFlag = 0; // flag for green LED
```

```
int counter=0;


void BlackLineInterruptHandler() // called every 10 ms
{


	TimerIntClear(TIMER1_BASE, TIMER_TIMA_TIMEOUT);
	TimerDisable(TIMER1_BASE, TIMER_A);// reset 10 ms timer
	counter++; //THIS WILL BE USED TO FIND TOTAL RUN TIME IN MAZE.
INCREMENTED EVERY 10MS THE ROBOT RUNS.
	Semaphore_post(Semaphore1); // this is gonna start blackline task
}
int bufferToggle;
int thinline = 0;
void blackLineTask(){//attach to timer
	int currentreflectance;
	while(1){
	int cycles = 0;
	Semaphore_pend(Semaphore1, BIOS_WAIT_FOREVER); //task unblocked every 10
[ms] by timer


	while (reflectance() == 1)	//WHILE SENSOR READING BLACK. allows one cycle
of white incase of error in reading. sometimes the sensor would read one white in the middle of a
```

black line and mess everything up. so this disregards one white reading in the middle of a black line.

```
    {
    cycles++;
    }

    if(cycles > 70){//number of cycles for thick line
    UARTprintf("%d", counter/90); //Prints the run time
    //Turns off motors
    PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, false);
    PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, false);
    PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, false);
    PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, false);

    unsigned long timer = 750000000/7;// 60 sec
    unsigned long flash = 5000000; //1 sec
    int led = 2;
        while(timer > 0){ //1 minute timer
        if(timer%(flash/4) == 0){
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3, led);//red on
        if(led == 2){
            led = 0;
```

```
            }
            else if(led == 0){
                led = 2;
            }
            }
            timer--;
            }
            GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 |
GPIO_PIN_3, 0); // Turns off LED


        BIOS_exit(0); //exit bios after 1 minute


        }
        else if(cycles > 5){//thin line
        thinline++;
        greenFlag = 0;


        }
        uint32_t ui32Period = (SysCtlClockGet() /1000);
        TimerEnable(TIMER1_BASE, TIMER_A);
        TimerLoadSet(TIMER1_BASE, TIMER_A, ui32Period);
        }
```

```
}

int leftDutyCycle;
int rightDutyCycle;
int pwmMAX = 6250; //(1/100 Hz)/(1/(40 MHz/64))

// Function prototypes for command handlers
void cmd_FORWARD(const char* args);
void cmd_BACK(const char* args);
void cmd_STOP(const char* args);
void cmd_ERR(const char* args);

//Bluetooth Commands
//Starts robot
void cmd_START(const char* args){

}
//sets custom right motor speed to go forwards
void cmd_RIGHTCUSTOMSPEED(double PID){
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, (PID)*pwmMAX/100);
}
```

```
//sets custom left motor speed to go forwards
void cmd_LEFTCUSTOMSPEED( double PID){
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, (PID)*pwmMAX/100);
}
//sets custom right motor speed to go backwards
void cmd_RIGHTCUSTOMSPEEDBACK(double PID){
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4, (PID)*pwmMAX/100);
}
//sets custom left motor speed to go backwards
void cmd_LEFTCUSTOMSPEEDBACK( double PID){
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5, (PID)*pwmMAX/100);
}

//both motors go forward fast
void cmd_ALLFORWARDFAST(const char* args) {
	int LED = 2;
	//UARTprintf("\n");
	//UARTprintf("Right Forward Fast");
	leftDutyCycle = 50;
	rightDutyCycle = 50;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, leftDutyCycle*pwmMAX/100);
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, leftDutyCycle*pwmMAX/100);
```

```
	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}

//Right motor moves fast
void cmd_RIGHTFORWARDFAST(const char* args) {
	int LED = 2;
	//UARTprintf("\n");
	//UARTprintf("Right Forward Fast");
	leftDutyCycle = 50;
	rightDutyCycle = 50;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, leftDutyCycle*pwmMAX/100);

	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}

//Left motor moves fast
void cmd_LEFTFORWARDFAST(const char* args) {
	int LED = 2;
	//UARTprintf("\n");
```

```
	//UARTprintf("Left Forward Fast");
	leftDutyCycle = 50;
	rightDutyCycle = 50;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, leftDutyCycle*pwmMAX/100);

	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}

//Right motor moves slow
void cmd_RIGHTFORWARDSLOW(const char* args) {
	int LED = 2;
	//UARTprintf("\n");
	//UARTprintf("Right Forward Slow");
	leftDutyCycle = 15;
	rightDutyCycle = 15;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, leftDutyCycle*pwmMAX/100);

	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}
```

```
//Left motor moves slow
void cmd_LEFTFORWARDSLOW(const char* args) {
	int LED = 2;
	//UARTprintf("\n");
	//UARTprintf("Left Forward Slow");
	leftDutyCycle = 15;
	rightDutyCycle = 15;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, leftDutyCycle*pwmMAX/100);

	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}

//Stops right motor of robot
void cmd_RIGHTSTOP(const char* args) {
	int LED = 2;
	//UARTprintf("\n");
	//UARTprintf("Right Stop");
	leftDutyCycle = 0;
	rightDutyCycle = 0;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, leftDutyCycle*pwmMAX/100);
```

```
    //UARTprintf("\n");
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}

//Stops left motor of robot
void cmd_LEFTSTOP(const char* args) {
    int LED = 2;
    //UARTprintf("\n");
    //UARTprintf("Left Stop");
    leftDutyCycle = 0;
    rightDutyCycle = 0;
    PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, leftDutyCycle*pwmMAX/100);

    //UARTprintf("\n");
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);//14=white 8= green 4=blue 2=red
}


//Robot moves backwards
void cmd_BACK(const char* args) {
```

```
	int LED = 4;
	//UARTprintf("\n");
	//UARTprintf("Backwards");

	leftDutyCycle = -50;
	rightDutyCycle = -50;
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_0, leftDutyCycle*pwmMAX/100);
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1, rightDutyCycle*pwmMAX/100);

	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_2, leftDutyCycle*pwmMAX/100);
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_3, rightDutyCycle*pwmMAX/100);
	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);
}

//Stops robot
void cmd_STOP(const char* args) {
	int LED = 8;
	//UARTprintf("\n");
	//UARTprintf("Stop All");
	leftDutyCycle = 0;
	rightDutyCycle = 0;
```

```
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6, leftDutyCycle*pwmMAX/100);
	PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7, rightDutyCycle*pwmMAX/100);



	//UARTprintf("\n");
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3,
LED);
}

// Command not found
void cmd_ERR(const char* args) {
	//UARTprintf("Error");
}

//Find the distance from the front of the robot to the wall
int findDistanceFront() {  //PE3
	uint32_t adc0Val[1];
	ADCProcessorTrigger(ADC0_BASE, 3);
	while (!ADCIntStatus(ADC0_BASE, 3, false)) {}
	ADCIntClear(ADC0_BASE, 3);
	ADCSequenceDataGet(ADC0_BASE, 3, adc0Val);
  //this eq converts ADC value to cm
	double adcVal = (adc0Val[0] + adc0Val[1])/2;
```

```
    // calculate distance from the front of the robot to the wall
    double distancef = -1.857*pow(10,-9)*pow(adcVal, 3) + 1.321*pow(10, -5)*pow(adcVal,
2) - 0.03355*adcVal + 35.537;
    return distancef; //Returns the distance from the front of the robot to the wall
}
uint32_t adc0Val2;
//Find the distance from the side of the robot to the wall
int findDistanceSide() {   //PE2
    uint32_t adc0Val[1];
    ADCProcessorTrigger(ADC0_BASE, 2);
    while (!ADCIntStatus(ADC0_BASE, 2, false)) {}
    ADCIntClear(ADC0_BASE, 2);
    ADCSequenceDataGet(ADC0_BASE, 2, adc0Val);
    double adcVal = (adc0Val[0] + adc0Val[1])/2;
    // calculate distance from the side of the robot to the wall
    double distances = -1.857*pow(10,-9)*pow(adcVal, 3) + 1.321*pow(10, -5)*pow(adcVal,
2) - 0.03355*adcVal + 35.537;
    adc0Val2 = adc0Val[0];
    return distances; //Returns the distance from the side of the robot to the wall
}



void btConfig() //Configure the bluetooth
```

```
{

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE,
                GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3);

        // Enable the GPIO Peripheral used by the UART.
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);



        // Enable UART1
        SysCtlPeripheralEnable(SYSCTL_PERIPH_UART1);

        UARTConfigSetExpClk(
        UART1_BASE, SysCtlClockGet(), 9600,
                (UART_CONFIG_WLEN_8 | UART_CONFIG_PAR_NONE |
UART_CONFIG_STOP_ONE));

        // Configure GPIO Pins for UART mode.
        GPIOPinConfigure(GPIO_PB0_U1RX);                    //PD6RX PD7TX
        GPIOPinConfigure(GPIO_PB1_U1TX);
        GPIOPinTypeUART(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
```

```
        // Use the internal 16MHz oscillator as the UART clock source.
        //UARTClockSourceSet(UART2_BASE, UART_CLOCK_PIOSC);
        UARTEnable(UART1_BASE);

        // Initialize the UART for console I/O.
        UARTStdioConfig(1, 9600, SysCtlClockGet());


}

void ADCConfigure(){//configures ADC for distance sensors
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
        GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_3);
        GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2);

        ADCSequenceDisable(ADC0_BASE,3);
        ADCSequenceDisable(ADC0_BASE,2);

        ADCSequenceConfigure(ADC0_BASE, 3, ADC_TRIGGER_PROCESSOR, 0);
        ADCSequenceStepConfigure(ADC0_BASE, 3, 0, ADC_CTL_CH0 | ADC_CTL_IE |
ADC_CTL_END);  //PE3
        ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);
```

```
    ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_CH1 | ADC_CTL_IE |
ADC_CTL_END);  //PE2
    ADCSequenceEnable(ADC0_BASE, 3);
    ADCSequenceEnable(ADC0_BASE, 2);
    ADCIntClear(ADC0_BASE, 3);
    ADCIntClear(ADC0_BASE, 2);

}

void PWMConfigure(){//Configure PWM for both motors, sets up PWM for both forwards and
Backwards

    pwmMAX = 6250; //(1/100 Hz)/(1/(40 MHz/64))

    SysCtlPWMClockSet(SYSCTL_PWMDIV_64);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);

    while(!SysCtlPeripheralReady(SYSCTL_PERIPH_PWM0)){};//wait for PWM to be
ready

    GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_4);
```

```
GPIOPinTypePWM(GPIO_PORTC_BASE, GPIO_PIN_5);
GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_4);
GPIOPinTypePWM(GPIO_PORTE_BASE, GPIO_PIN_5);

GPIOPinConfigure(GPIO_PC4_M0PWM6);
GPIOPinConfigure(GPIO_PC5_M0PWM7);
GPIOPinConfigure(GPIO_PE4_M0PWM4);
GPIOPinConfigure(GPIO_PE5_M0PWM5);

PWMGenConfigure(PWM0_BASE, PWM_GEN_2, PWM_GEN_MODE_UP_DOWN);
PWMGenConfigure(PWM0_BASE, PWM_GEN_3, PWM_GEN_MODE_UP_DOWN);
PWMGenPeriodSet(PWM0_BASE,PWM_GEN_2,pwmMAX);
PWMGenPeriodSet(PWM0_BASE,PWM_GEN_3,pwmMAX);

PWMPulseWidthSet(PWM0_BASE, PWM_OUT_6,0*pwmMAX/100);
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_7,0*pwmMAX/100);
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_4,0*pwmMAX/100);
PWMPulseWidthSet(PWM0_BASE, PWM_OUT_5,0*pwmMAX/100);

PWMOutputState(PWM0_BASE, PWM_OUT_6_BIT, true);
PWMOutputState(PWM0_BASE, PWM_OUT_7_BIT, true);
PWMOutputState(PWM0_BASE, PWM_OUT_4_BIT, true);
PWMOutputState(PWM0_BASE, PWM_OUT_5_BIT, true);
```

```
    PWMGenEnable(PWM0_BASE, PWM_GEN_3);
    PWMGenEnable(PWM0_BASE, PWM_GEN_2);

}


// Define PID parameters
#define Kp 20
#define Ki 1
#define Kd 0.1

void PIDInterruptHandler()// called every 50 ms
{
    TimerIntClear(TIMER2_BASE, TIMER_TIMA_TIMEOUT);
    Semaphore_post(Semaphore0); // this is gonna start PID task

}

int pidCycles=0;
uint16_t ping[20]; //Initialize ping buffer
uint16_t pong[20]; //Initialize pong buffer
uint16_t*  currentbuffer=ping;
```

```
void switchBuffer()//toggle ping/pong buffer
{
	if (currentbuffer==ping){
	currentbuffer=pong;
	//Turn off LED
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
	//Turn on yellow LED
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 10);
	}
	else {
	currentbuffer=ping;
	//Turn off LED
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
	//Turn on blue LED
	GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 4);
	}
}

uint8_t upperBits;
uint8_t lowerBits;

void pingPongSWIHandler()//CALLED WHEN BUFFER IS FULL
{
```

```
        UARTprintf(":6");
        int i;
        for( i=0;i<20;i++){
        upperBits = currentbuffer[i] >> 8;
        lowerBits = currentbuffer[i] & 0xFF;

        UARTprintf("%c",upperBits);//print buffer
        UARTprintf("%c",lowerBits);//print buffer
        //UARTprintf("%i\n",currentbuffer[i]);
        }
        UARTprintf("\r\n");
        switchBuffer();  //switch from ping to pong buffer or pong to ping

}


int pidCounter = 0;
int lastError = 0;

int flag = 0;
int bufferIndex = 0;

void ComputePID() { //Update PWM based on front and right sensor values
```

```
int frontTime = 0;
while(1){

Semaphore_pend(Semaphore0, BIOS_WAIT_FOREVER);  //task unblocked every 50
[ms] by timer



int desired = 14;

int frontDistance = findDistanceFront();
int adcValSide = findDistanceSide();

int error = adcValSide - desired; //error in cm
uint16_t error2 = (uint16_t)(abs(adc0Val2 - 1850));
double P = Kp*fabs(error);
double I = Ki*(error+lastError);
double D = Kd*(error-lastError);

double output = (P + I + D); //PID value to be added to PWM
lastError = error;

//Checks whether the robot is too close or too far from the right wall
if(error > 0){// Too far from right wall
```

```
        cmd_RIGHTCUSTOMSPEED(99-output);
        cmd_LEFTCUSTOMSPEED(99);
}
else if(error < 0){//Too close to right wall
        cmd_RIGHTCUSTOMSPEED(99);
        cmd_LEFTCUSTOMSPEED(99-output);
}
//Dead End, Performs U-turn
if(frontDistance < 10){
TimerDisable(TIMER1_BASE, TIMER_A);
TimerDisable(TIMER2_BASE,TIMER_A);
//GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 2);
// Stops motors
cmd_RIGHTCUSTOMSPEED(0);
cmd_LEFTCUSTOMSPEED(0);
//SysCtlDelay(1000);
//Turns robot
cmd_LEFTCUSTOMSPEEDBACK(99);
cmd_RIGHTCUSTOMSPEED(99);
while(findDistanceFront() <  22);
cmd_LEFTCUSTOMSPEED(99);
cmd_LEFTCUSTOMSPEEDBACK(0);
```

```
TimerEnable(TIMER1_BASE,TIMER_A);
TimerEnable(TIMER2_BASE,TIMER_A);



}

if(thinline==2)//after the second thin line stops collecting and printing data also set numthinlines to 999 so it won't keep re-enter this if statement. print remaining data in buffers
{
int i;
UARTprintf(":6");
for( i=0;i<bufferIndex;i++){
upperBits = currentbuffer[i] >> 8;
lowerBits = currentbuffer[i] & 0xFF;
UARTprintf("%c", upperBits);
UARTprintf("%c", lowerBits);
//UARTprintf("%i\n", currentbuffer[i]);
}
UARTprintf("\r\n");
//turn off LED
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 0);
//Turn red LED on
GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_3, 2);
```

```
        thinline=999;//do this so it doesn't repeat itself every 10ms
    }

    if((pidCycles%2==0)&&(thinline==1)) //every 100[ms] (every other PID cycle) && after
first thin line and before second thin line
    {
    currentbuffer[bufferIndex]= (error2); //Fill in current buffer
    if((bufferIndex == 19)) //every 40 PID cycles (2[s]) we call SWI to print buffer.
    {
        Swi_post(swi0);//SWI
    }
    bufferIndex = (bufferIndex + 1)%20; //Update buffer index

    }

    if(thinline==1){ // First thin black line crossed
    //UARTprintf("PID CYCLES: %d",pidCycles);
    pidCycles++;
    if(greenFlag == 0){//turn LED Green after 1st thin line
```

```
//Reset LEDs
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 0);
//LED set to green
        GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_1 | GPIO_PIN_2 | GPIO_PIN_3, 8);
        greenFlag = 1;
        }
        }
        }
}

int frontdist, sidedist; // Global variables for sensor distance

int main(void) {
        SysCtlClockSet(SYSCTL_SYSDIV_4 | SYSCTL_USE_PLL | SYSCTL_XTAL_16MHZ
                | SYSCTL_OSC_MAIN); // Set system clock division to 4 and the crystal clock to
                                16 mHz

        PWMConfigure();//start PWM
        btConfig();//start bluetooth
        ADCConfigure();//start ADC
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

        //UARTprintf("Hello World\n");
```

```
ConfigureTimer2A();//start timers
ConfigureTimer1A();
lastError = findDistanceSide() - 10;
char commandString[4]; //Variable used to find command from command list
char uartstring[2]; //User input stored
while(uartstring[0] != 'g' && uartstring[1] != 'o'){ //Enter loop when command is “go”

while (!UARTCharsAvail(UART1_BASE));//wait for bluetooth start command

        uartstring[0] = UARTCharGet(UART1_BASE); //read input
        uartstring[1] = UARTCharGet(UART1_BASE);



        char commandString[4];
        strncpy(commandString, uartstring, 2);
        commandString[2] = '\0';

        char* receivedArgs = uartstring;

        int i;
        int numCommands = sizeof(commandList) / sizeof(commandList[0]);
        for(i = 0; i < numCommands; i++){ //execute command based on input
        if(strcmp(commandString, commandList[i].command) == 0){
```

```
                commandList[i].function(receivedArgs);
                break;
                }
                }

                if(i == numCommands){
                cmd_ERR(receivedArgs);
                }
        }
        // Initialize TI-RTOS
        start = clock();
        BIOS_start(); // Start the RTOS scheduler
}
```

Problems, Suggestions for Improvements

There were some issues that arose over the course of the semester. One such issue was that the battery pack would sometimes rub against the wheels and cause friction, impeding the movement of the robot. The wheels would also occasionally pick up hairs that were accumulated in the maze with dust, which may have contributed to a small amount of friction and drift on the movement of the robot. Perhaps moving forward, there can be routine cleaning of the maze to reduce as many variables as possible to make the robots perform more consistently, with fewer external variables. With further regards to the battery pack, we used a battery pack that utilized 6 AA (1.5 volt) batteries, which increased the overall weight of the robot. A change that may be beneficial would be using a 9 volt D battery to make the robot lighter, and therefore, perhaps faster. Another issue that arose was having a thin black line too close to a turn. When turning, the robot is not completely centered in the maze so it takes time for it to become centered. The problem is that sometimes the robot will cross the thin black line at an angle to where it reads the thin black line as thicker than it actually is. A possible solution is to have the robot turn at a 90 degree angle rather than having it do a curved turn.

Conclusion

This semester wide project was an extremely challenging, yet gratifying endeavor. As a team, we embarked on a journey to design, construct, and program a maze-traversing robot, learning and applying a multitude of microcomputer principles. The utilization of threads, multitasking, hardware and software interrupts, and semaphores provided a multitude of functions to the final rendition of our robot. The robot stands not only as a testament to our technical skills developed throughout the semester, but also as a symbol of perseverance and many late nights in the electronic lab. This journey has equipped us with invaluable insights, preparing us for future endeavors in the ever-evolving field of embedded microcomputers.