ECE 4437: Embedded Microcomputer Systems

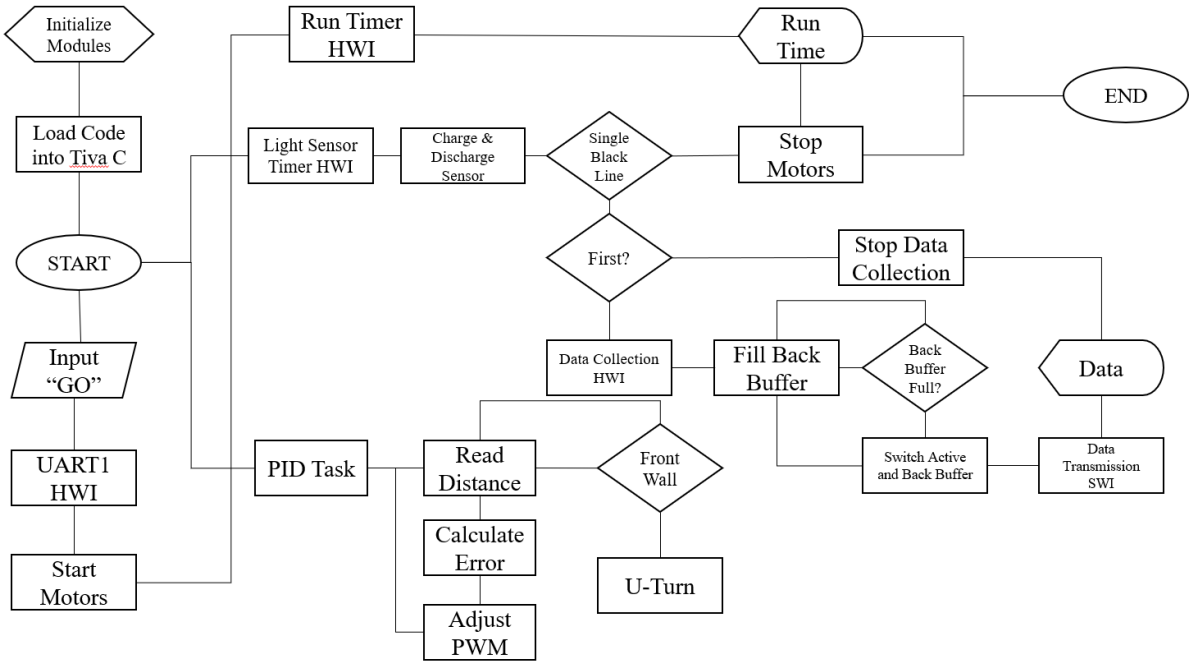
Maze Robot Documentation

David Rodriguez
Rogelio Lopez
Jamal Stewart

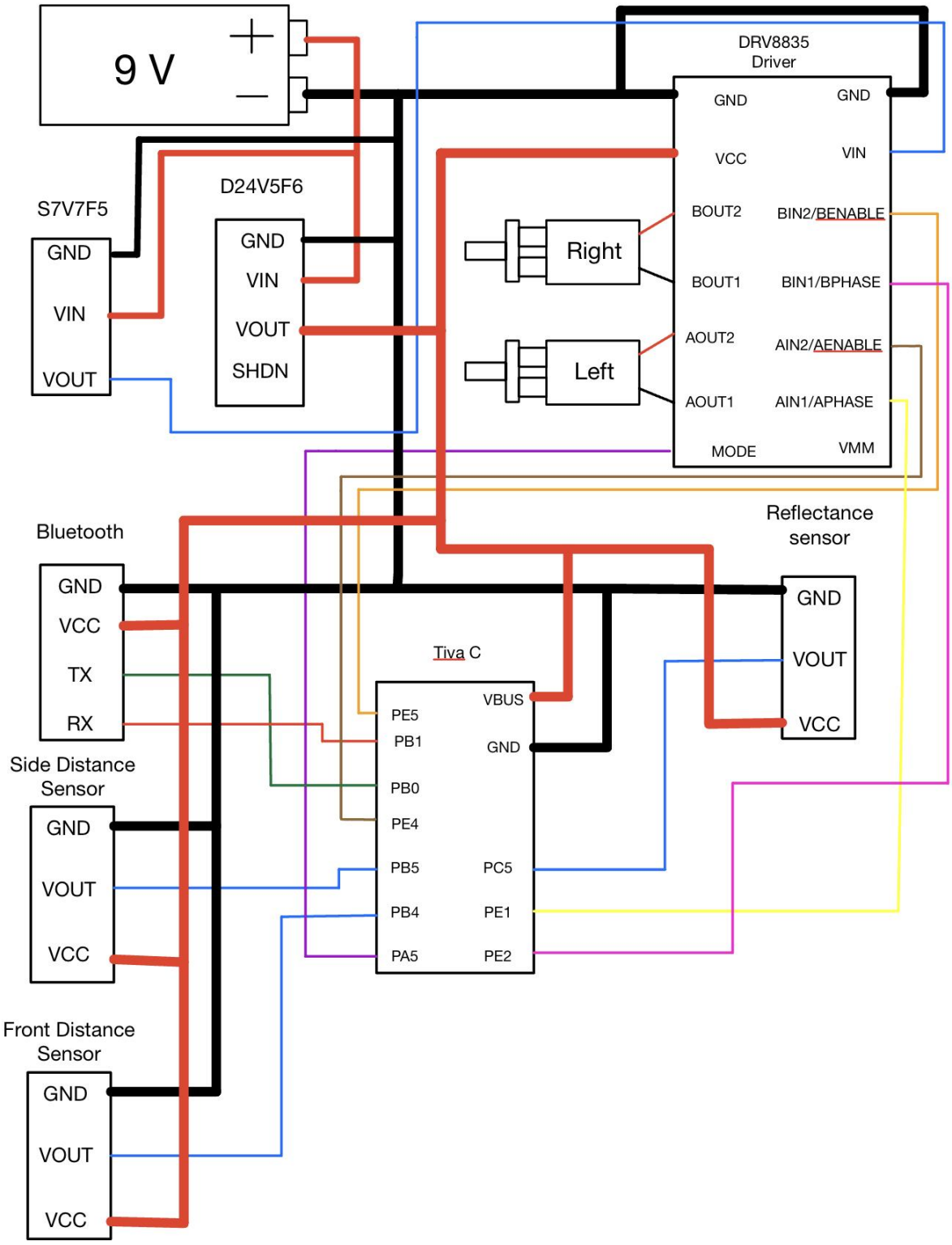
Intro

The goal of this project was to build and design the software for a robot that navigated through a maze using various sensors. For the hardware, this was to be done using the Tiva-C TM4C123GH6PM along with many other parts provided by the class. The software was to use TI-RTOS and be made in Code Composure Studio. The robot moves using two motors that are controlled and modified using real-time feedback of distance sensors and IR sensors. It also communicates data via bluetooth.

Block Diagrams



Wiring Details



Functions and Pseudo Code

Robot Navigation

PID() - This function is the task that is what steers our robot using PID control and allows our robot to do Uturns. The task is unlocked using a semaphore. It takes care or straight line steering, right turns and U-turns. It uses the right distance sensor ADC value to calculate an error value which then is used to calculate the the PID values which is used to calculate a steer value. It uses the front distance sensor ADC value to determine when it needs to U-turn.

```
void PID() {

        Initialize max steer value
        Initialize integral value

        While forever {

                Pend semaphore to unlock function
                Trigger ADC
                Set up voltage value array
                Get ADC values

                If front sensor is to close to wall{
                        While front sensor doesnt sensor open space in front{
                        Uturn robot counterclockwise
                        Update ADC values into voltage value array}
                }

                Move robot forward
                }

                If there is no wall to close to front of robot{

                        Calculate error value

                        Initialize kp, kd, and kd coefficients
                        Calculate PID values

                        Calculate steerVal

                        If steer value is above max{
                                Set steer value to max}

                        If right sensor is to far from wall{
                                Steer right}

                        If right sensor is to close to wall{
                                Steer left}

                        Record current error before it updates
                        Update integral value with updated one
}}}
```

Command Interpreter Functions (MENU Functions)

Run() - This function enables the PWM signal and starts running the motor

```
void Run(){
        Call forward function
        Start motor function
        Enable 1 ms timer
}
```

EmergencyStop() - This function stops our robot using the Brake() function, disables our timer used to time the robot, and prints the Total time. It is called in our BlackLineDetection() function once a thick line is detected.

```
void EmergencyStop(){
        Print emergency stop to UART
        Call brake function
        Disable 1 ms timer
        Print total time in seconds and  milliseconds to UART
}
```

Data Collection and Black Line Detection

LightSensorInit(void) - This function Initializes the GPIO for the reflectance senso to Port C Pin 5. It also charges and discharges the reflectance sensor to be able to determine how long it takes the reflectance sensor to discharge.

```
void LightSensorInit(void) {
        Configure GPIO Pin C
        Configure Port C Pin 5
        write into pin C5. Turns on pin 5 to charge light sensor
        Wait so light sensor gets fully charged
        Turn pin 5 off pin 5
        Turns pin 5 into input
}
```

DataTimerHandler() - This function is in charge of filling the buffer with the error value from the PID task, switching buffers, and posting the SWI that is in charge of transmitting data and toggling green and blue LEDs.

```
void DataTimerHandler(){

        Clear Timer Interrupt Sources

        If buffer is not full{
                Then add error value to buffer
                Increment Error value count }
        Else if sample # is 20 {
                Reset error value count }

        If buffer is full and back buffer is buffer 1{
                Then Swap Buffers
                Post swi for data transfer}

        Else if buffer is full and back buffer is buffer 1{
                Then Swap Buffers
                Post swi for data transfer}
}
```

TransmitData() - This function is the SWI that is called in the DataTimerHandler HWI to transmit data. First, it turns off blue led and turn on green one to indicate data is being transmitted. Then the function splits a 16 bit integer into two 8 bit integers to send each 8 bit integer as a char through bluetooth. And it ends with turning off green led and turning on blue one to indicate data transmission has stopped.

```
void TransmitData(){
        Set port f and pin 3 & 2 for LED
        Turn on green led, turn off blue

        If we are in data collection mode (between thin line 1 and 2){

                For 1 to 20 error values in buffer {
                        Split 16 bit integer into two 8 bit integers

                        Print both 8 bit integers out}

                        Print “24 CR LF” to UART1
}
        Else if we are not in data collection{
                For every remaining sample in non-full buffer{
                        Split 16 bit integer into two 8 bit integers

                        Print both 8 bit integers out}}

Turn off green led, turn on blue
}
```

ReadLightSensor() - This function returns a value used to determine if the surface is black or white. It accomplishes this by using a counter that counts how long it takes the Input pin 5 to go from logic high to logic low (how long it takes the sensor to discharge. Essentially, the longer it takes, the darker the surface. For our robot, if the counter exceeds 5000 then it is a black surface.

```
int ReadLightSensor(){

        Call light sensor init function for configuration

        Reset light variable equal to 0

        While pin C5 reads logic high {
                Increment light count
        }

        Return light count
}
```

BlackLineDetection() - Called in the LightSensorTimerHandler every 5 [ms], this function is allows data collection to be start on the first thin line stop at the second thin line. It also stops the robot once the robot hits a thick line. It determines whether a line is thin or thick by counting how many consecutive “black” readings our ReadLightSensor() function. The more black readings the thicker the line. For our robot and its speed, we determined that 11 consecutive black readings or less means it is a thin line. We use a counter to determine whether it is the first thin line or thick line to enable and disable the DataCollection timer accordingly.

```
void BlackLineDetection() {

        Read light value

        If current light value indicates black surface{
                Increment count of consistent black surface readings
        }


        If current light value indicates white surface {
                If were are going from black reading to white reading{
                        If thin line{
                                If its the first thin line{

                                        Turn on blue led
                                        Start data collection by enabling data collection timer}

                                If its the 2nd black thin line{

                                        Turn off blue led
                                        Stop data collection by disabling data collection timer}

                        Increment thin line count
                        }

                        If thick line {
                                Stop robot}
                        Reset thickness counter
}}}
```

Motor Control Functions

Start_Motor() - This function turns on the motors.

void Start_Motor(){

Set PWM output 4 bit to true to turn on left motor

Set PWM output 5 bit to true tot turn on right motor

}

forward() - This function makes the robot go forward.

void forward(){

Enable Port E1 as 0x0 for forward movement of left wheel

Enable Port E2 as 0x0 for forward movement of right wheel

}

backward() - This function makes the robot go backwards.

void backward(){

Enable Port E1 as 0x2 for backward movement of wheel

Enable Port E2 as 0x4 for backward movement of wheel

}

Brake() -This function breaks the robot by setting the duty cycle of the PWM signal to 0.

void Brake(){

Set PWM output 4 bit to false to turn off left motor

Set PWM output 5 bit to false to turn off right motor

}

TurnRight(float value) - This function steers our robot right with calculated steer value

void TurnRight(float value){

Set PWM output 4 bit to 2000 to set left motor to full speed

*Set PWM output 5 bit to 2000 * (100 - Steervalue found in PID function) * 0.01 to steer right wheel*

}

TurnLeft(float value) - This function steers our robot left with calculated steer value

void TurnLeft(float value){

*Set PWM output 4 bit to 2000 * (100 - Steervalue found in PID function) * 0.01 to steer left wheel*

Set PWM output 5 bit to 2000 to set right motor to full speed

}

void uturnCCL() - This function is used to u-turn in our PID control function once the front sensor reading indicated there is a wall in front of our robot.

void uturnCCL(){

Enable Port E1 as 0x2 to turn wheel backwards

Enable Port E2 as 0x0 to make wheel go forward

Set PWM output 4 bit to 2000 to set left motor to full speed

Set PWM output 5 bit to 2000 to set right motor to full speed

}

Timer set up

void MazeTimer() - This function configures a timer to time how long it takes the robot to run complete the maze

void MazeTimer() {

Set CPU Clock to 40MHz

Timer 3 Configuration

Configure Timer 3B Periodic Mode

Period = 0.01 sec

}

DataTimer() - This function is allows data collection to be start on the first thin line stop at the second thin line. It also stops the robot once the robot hits a thick line.

void DataTimer() {

Set CPU Clock to 40Mhz

Setup Timer 2

Set period to 100 [ms]

Enable timer}

LightSensorTimerInit() - This function initializes timer using WTIMER0 to trigger the reflectance sensor to read every 5ms.

void LightSensorTimerInit() {

Set Clock Source

Configure WTimer0 Periodic Mode

Load WTimer0 with a period of 50 ms

Enable WTimer0 to trigger an interrupt when period is reached

Enable WTimer0

}

ADCTimer_Init(void) - This function initializes timer using WTIMER2 to trigger the ADCTimer_Handler() to post the semaphore so the PID control can be processed every 50ms.

void ADCTimer_Init(void) {

Set Clock Source

Configure WTimer2 Periodic Mode

Load WTimer2 with a period of 50 ms

Enable WTimer2 to trigger an interrupt when period is reached

Enable WTimer2

}

Timer handlers

LightSensorTimerHandler() - This function timer handler calls our BlackLineDetection() function every 5 ms.

void LightSensorTimerHandler() {

Clear Timer Interrupt Sources

Call BlackLineDetection function

}

ADCTimer_Handler() - This function is our ADC handler that post semaphore so that the PID function can be unlocked every 50 ms

void ADCTimer_Handler() {

Clear Timer Interrupt Sources

Post semaphore to unlock PID function

}

Peripheral Initialization Functions

UART0_init()

void UART0_init(){

System clockset 16MHz
Enable UART0
Enable Port A
Configure PA0 for receiving
Configure PA1 for transmitting
Configure pins PA1 and PA0 for use by UART peripheral
Enable Port F for LED
Enable port f2 for LED
Set clock config for 115200 bandwidth for UART
Initalize UART0 for console I/O

}

UART1_init()

void UART1_init(){

System clockset 16MHz
Enable UART1
Enable Port B
Configure PB0 for receiving
Configure PB1 for transmitting
Configure pins PB1 and PB0 for use by UART peripheral
Enable Port F for LED
Enable port f 1-3 for led
Set clock config for 115200 bandwidth
Enabled FIFO for uart 1 for both transmit and receiving
Enabled processor interrupts
Enable the UART interrupt
Enabled RX and TX interrupts
Initalize UART for console I/O

}

UART1IntHandler(void)

void UART1IntHandler(void){

Clear the asserted interrupts

While loop while characters are available{

Set variable one to the character we got from UART1

Set variable two to the character we got from UART 1

Creare a command array with variable one and two

For loop iterate from 0 to 14{

If statement that compares our command to our list of commands and if a match is found{

We blink LED

Delay clock by 1 ms

Turn off led

Run the function for the command chosen.

}

LEDConfig() - Configures the LEDs for use

void LEDConfig(){

Add tiva-c GPIO setup

Enables port

Sets pins 1-3 (RGB pins for output)

}

ADC initConfig() - This functions configures ADC

void ADCinit(void) {

Configure the ADC0 for use

Configure GPIO Pin B for use

Set ADC clock

Set pin to ADC

Set the ADC reading trigger

set channels to read in samples

enable adc

}

PWMinit() - This function configures the PWM signals

void PWMinit(void) {

Set clock

Configure the PWM0 for use

Set Clock for PWM

Configure PWM Module 0 Generator 2

*Set Period to 20kHz (N = (1 / f) * PWMClk.)*

Set the pulse width of PWM0

Configure GPIO Pin E for use

Configure GPIO Pins to send PWM to motors through Motor

Enable generator 2 in module 0

Configure GPIO Pins 1 and 2 for output

Configure GPIO Port A for use

Configure GPIO Port A Pin 5 for mode

Configure Mode = 1

}