FAMU- FSU College of Engineering

Department of Electrical and Computer Engineering

Fall 2024

EEL-4742L Advanced Microprocessor-Based System Design Lab Report

Section No: 1

Lab Instructor: Apuroop Sampath Ponna

Lab No: 10

Lab Title: Robot Navigation

Name: Sebastian Cox

Partner's Name: Keegan Mcgilvray

Date Performed: November 14, 2024

Date Delivered: November 21, 2024

Contents

1. Introduction

Lab #10 is the last challenge students face during this course. All previous labs have aspects that apply here. The purpose of Lab #10 is to assign students with the difficult task of implementing a program in the TI-RSLK MAX that follows a course designed by the instructor. The course in question contains multiple turns in both directions, along with an area that has no line that the QTRK line sensor can rely on, forcing students to create a code that can navigate on and off of the track. At the end of this experiment, the algorithm designed should allow the TI-RSLK MAX to successfully follow a pre-designed maze using the QTRK line sensor. This can be done with a simple program, but the largest obstacle students will face is time, as the robot must navigate the entire course in under two minutes, with large incentives given for students whose times are short. This brings a sense of competition to the lab giving freedom to students to implement the design however desired, as long as the TI-RSLK MAX steers through the course by itself.

2. Design Requirements

This lab's code is similar to that of Lab #8 code, which revolved around the QTRK line sensor and its ability to navigate a short course containing two turns. Upon startup, the MSP432 and TI-RSLK MAX will toggle every LED once, to indicate to the user that the device is functioning properly. This solved the issue where one could not determine if it was a hardware issue such as a lack of battery charge, or a bug in the code. After that, the TI-RSLK MAX's QTRK line sensor will begin reading the inputs presented. An improved version of the algorithm used from the previous experiments involving the line sensor was created. The difference in this lab is multiple changes. The first change involved slowing

the wheels down, to allow the device to have enough time to read. Along with that, there are more scenarios added, meaning when the robot needs to turn, there will be different speeds at which it performs the turn based on its current inputs and position. This allows for smooth and effortless turns which decreases the overall time needed to complete the maze. All these features successfully allowed the TI-RSLK MAX to complete the course in an adequate time.

3. Theoretical Design

Lab #10 used a different algorithm in week 2, adding more scenarios for the QTRK line sensor to allow for smoother turns. This works because it saves its last mode, such as hard left or hard right, and takes it into account to determine the next direction. The pseudocode being described as:

DEFINE PERIOD as 100
DEFINE DUTY as 25
DEFINE CLOCKDIVIDER as 48
DEFINE LEFTCHANNEL TIMER_A_CAPTURECOMPARE_REGISTER_4
DEFINE RIGHTCHANNEL TIMER_A_CAPTURECOMPARE_REGISTER_3
DECLARE config432IO
DECLARE configRobotIO
DECLARE configPWMTimer
DECLARE powerUp
DECLARE standBy
DECLARE tableII
DECLARE rotate
DECLARE bumperSwitchesHandler
DECLARE readSensor
DECLARE Input_Output

DECLARE Tracking

DECLARE Lost

DECLARE RGBDriver

DECLARE sensorReader

DECLARE enum motorState

DECLARE enum buttonStates

DECLARE enum whiteturning

DECLARE enum greenturning

DECLARE enum allturning

DECLARE enum speed

DECLARE buttonStates b0, bn

DECLARE motorState leftMotorState, rightMotorState

DECLARE whiteturning wturn

DECLARE greenturning gturn

DECLARE allturning aturn

DECLARE Condition Con

DECLARE led2 LED2State

DECLARE speed izer, stabl, stabr

DECLARE LEFT, RIGHT, NumSensor

main:

SET b0 to buttonOFF

STOP WDT Timer

CALL config432IO

CALL configRobotIO

CALL powerUp

CALL configPWMTimer with PERIOD, CLOCKDIVIDER, DUTY, LEFTCHANNEL

CALL configPWMTimer with PERIOD, CLOCKDIVIDER, DUTY, RIGHTCHANNEL

START Timer A

SET leftMotorState, rightMotorState to motorOFF

Enable Interrupts

SET CNTL EVEN and CNTL ODD to LOW

CALL bumperSwitchesHandler

DELAY 1.5 s

Repeat

CALL Input_Output

IF b0 is buttonON THEN

CALL readSensor

CALL sensorReader

START Timer A

IF izer = stable THEN

DELAY 15 ms

ELSE

DELAY 3.05 ms

End IF

ELSE

CALL standBy

End IF

End of Repeat

End of main.

Function 1: **configPWMTimer**

SET PWM Timer Configuration with clockPeriod, clockDivider, duty, and channel

GENERATE PWM

STOP Timer

End of Function 1

Function 2: **config432IO**

SET GPIO_PORT_P1.0, GPIO_PORT_P2.0-2 as Output

SET All GPIO_PORT_P1 and GPIO_PORT_P2 Outputs to Low

End of Function 2

Function 3: **configRobotIO**

SET LEDs for Front Left Yellow, Front Right Yellow, Back Right Red, and Back Left Red as Output

SET All Robot LEDs to Low

SET GPIO_PORT_P4 Pins as Input with Pull-Up Resistors

ENABLE Interrupts for GPIO_PORT_P4 Pins

CONFIGURE Interrupt Edge as High to Low for GPIO_PORT_P4 Pins

CLEAR Interrupt Flags for GPIO_PORT_P4 Pins

CONFIGURE Motor Pins as Output and PWM

ENABLE Left and Right Wheel Outputs

End of Function 3

Function 4: **powerUp**

SET All LEDs to ON

TOGGLE LED2 White

DELAY 3 s

SET All LEDs to OFF

SET Wheels to OFF

End of Function 4

Function 5: **standBy**

TOGGLE LED1 at 1 Hz

SET LED2 to Green

Disable Motors

End of Function 5

Function 6: **tableII**

IF LEFT = 0 AND RIGHT = 0 THEN

SET LED2 to Cyan

ELSE IF LEFT > 0 AND RIGHT = 0 THEN

SET LED2 to Red

ELSE IF RIGHT > 0 AND LEFT = 0 THEN

SET LED2 to Blue

ELSE IF LEFT > RIGHT THEN

SET LED2 to Yellow

ELSE IF LEFT = RIGHT THEN

SET LED2 to Green

ELSE

SET LED2 to Cyan

End IF

End of Function 6

Function 7: **rotate**

IF Cond = ALL_LEFT THEN

CONFIGURE Motors for Hard Left Turn

SET Stability States to Reset or Stable

UPDATE Turning States

ELSE IF Cond = MORE_LEFT THEN

CONFIGURE Motors for Slight Left Turn

UPDATE Turning and Stability States

ELSE IF Cond = MIDDLE THEN

CONFIGURE Motors for Straight Movement

UPDATE Stability and Turning States

ELSE IF Cond = MORE_RIGHT THEN

CONFIGURE Motors for Slight Right Turn

UPDATE Turning and Stability States

ELSE IF Cond = ALL_RIGHT THEN

CONFIGURE Motors for Hard Right Turn

SET Stability States to Reset or Stable

UPDATE Turning States

ELSE

CONFIGURE Motors for Stopping

UPDATE Turning and Stability States

End IF

End of Function 7

Function 8: **bumperSwitchesHandler**

DECLARE status

DELAY 300 ms

SET status to GPIO Interrupt Status

SWITCH status:

CASE GPIO_PIN0:

TOGGLE b0

End SWITCH

End of Function 8

Function 9: **Input_Output**

SET CNTL EVEN and CNTL ODD to HIGH

CONFIGURE P7 Pins as Output

SET P7 Pins to HIGH

DELAY 10 us

CONFIGURE P7 Pins as Input

DELAY 10 ms

SET NumSensor Bits Based on P7 Inputs

SET CNTL EVEN and CNTL ODD to LOW

End of Function 9

Function 10: **readSensor**

SET LEFT and RIGHT to 0

IF NumSensor Bits 0-3 are True THEN

Increment RIGHT

IF NumSensor Bits 4-7 are True THEN

Increment LEFT

SET Condition Based on LEFT and RIGHT Values

End of Function 10

Function 11: **Tracking**

SET LED1 to ON

CALL tableII

End of Function 11

Function 12: **Lost**

SET LED1 to OFF

SET LED2 to White

TURN OFF RSLK LEDs

TURN OFF Wheels

End of Function 12

Function 13: **RGBDriver**

SET LED2 Color Based on Input

End of Function 13

Function 14: **sensorReader**

SWITCH Condition:

CASE ALL_LEFT:

CALL RGBDriver(RED)

CALL rotate(ALL_LEFT)

CASE MORE_LEFT:

CALL RGBDriver(YELLOW)

CALL rotate(MORE_LEFT)

CASE MIDDLE:

CALL RGBDriver(GREEN)

CALL rotate(MIDDLE)

CASE MORE_RIGHT:

CALL RGBDriver(CYAN)

CALL rotate(MORE_RIGHT)

CASE ALL_RIGHT:

CALL RGBDriver(BLUE)

CALL rotate(ALL_RIGHT)

CASE OFF:

CALL RGBDriver(WHITE)

CALL rotate(OFF)

IF Condition != OFF THEN

SET LED1 to ON

ELSE

SET LED1 to OFF

End SWITCH

End of Function 14

4. Synthesized Design

```c
/*
    Sebastian Cox
    Keegan McGilvray
    12/06/2024
    Lab #10
    This code allows the robot so self navigate a path of a black line
 */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
#include <stdint.h>
#include <stdbool.h>

#define PERIOD 100  // PWM
#define DUTY 25 // percent ON
#define CLOCKDIVIDER 48  // Clock Divider Value
#define LEFTCHANNEL TIMER_A_CAPTURECOMPARE_REGISTER_4
#define RIGHTCHANNEL TIMER_A_CAPTURECOMPARE_REGISTER_3

typedef enum {ALL_LEFT, MORE_LEFT, MIDDLE, MORE_RIGHT, ALL_RIGHT, OFF, STANDBY} Condition;
typedef enum {
  LED2OFF,
  RED,
  GREEN,
  BLUE,
  YELLOW,
  CYAN,
  MAGENTA,
  WHITE,
LED2MAINTAIN}
led2;
typedef enum {motorOFF, motorFoward, motorReverse} motorState;
typedef enum {buttonON, buttonOFF} buttonStates;
typedef enum{lastLeft, lastRight} whiteturning;
typedef enum{Middle, glastLeft,gwlastLeft, glastRight, gwlastRight} greenturning;
typedef enum{wMiddle ,wLeft, wRight} allturning;
typedef enum{reset, stable} speed;

Timer_A_UpModeConfig timerConfig;
Timer_A_PWMConfig  timerPWMConfig;

void config432IO(void);
void configRobotIO(void);
void readBumperSwitches(void);
void powerUp(void);
void standBy(void);
void tableII(void);
void rotate(Condition);
void bumperSwitchesHandler(void);
void readSensor(void);
void Input_Output(void);
void Tracking(void);
void Lost(void);
void sensorReader(void);
void configPWMTimer(uint16_t clockPeriod, uint16_t clockDivider, uint16_t duty,uint16_t channel);
void RGBDriver(led2 LEDColor);
buttonStates b0, bn;
motorState leftMotorState, rightMotorState;
whiteturning wturn = lastLeft;
greenturning gturn = Middle;
allturning aturn = wMiddle;
Condition Con = OFF;
led2 LED2State = LED2OFF;
speed izer = stable;
speed stabl = reset;
speed stabr = reset;
uint8_t LEFT = 0, RIGHT = 0, NumSensor = 0;


int main(void) {

    b0 = buttonOFF;

    MAP_WDT_A_holdTimer();


    config432IO(); // Config MSP432 LEDs
    configRobotIO(); // Config Robot
    powerUp();

   configPWMTimer(PERIOD,CLOCKDIVIDER,DUTY,LEFTCHANNEL); // Start L wheel timer
   configPWMTimer(PERIOD,CLOCKDIVIDER,DUTY,RIGHTCHANNEL); // Start R wheel timer

   MAP_Timer_A_startCounter(TIMER_A0_BASE,TIMER_A_UP_MODE);

   leftMotorState = motorOFF;
   rightMotorState = motorOFF;

   __enable_interrupt();

   MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN3); // CNTL EVEN LOW
   MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P9, GPIO_PIN2); // CNTL ODD LOW

   bumperSwitchesHandler();

```

```
   __delay_cycles(1500000);
    while (1) {

        Input_Output();

        if (b0 == buttonON){

            readSensor();
            sensorReader();
            MAP_Timer_A_startCounter(TIMER_A0_BASE,TIMER_A_UP_MODE);
            if(izer==stable){
            __delay_cycles(15000);
            }
            else{
                __delay_cycles(3050);
            }
        }

        else{

            standBy();
        }

        } // End of while
} // End of main


// Function configures the PWM timer
void configPWMTimer(uint16_t clockPeriod, uint16_t clockDivider, uint16_t duty, uint16_t channel) {
const uint32_t TIMER=TIMER_A0_BASE;
uint16_t dutyCycle = duty*clockPeriod/100;
timerPWMConfig.clockSource = TIMER_A_CLOCKSOURCE_SMCLK;
timerPWMConfig.clockSourceDivider = clockDivider;
timerPWMConfig.timerPeriod = clockPeriod;
timerPWMConfig.compareOutputMode = TIMER_A_OUTPUTMODE_TOGGLE_SET;
timerPWMConfig.compareRegister = channel;
timerPWMConfig.dutyCycle = dutyCycle;
MAP_Timer_A_generatePWM(TIMER, &timerPWMConfig);
MAP_Timer_A_stopTimer(TIMER);

}


// Function sets LEDs as outputs and have them begin off
void config432IO(void){


      MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0); // Red LED1
      MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0); // Red 2
      MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1); // Green
      MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2); // Blue

      // Turn OFF
      MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0); // Red LED1
      MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); // Red 2
      MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); // Green
      MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); // Blue


}

// Function sets all the bumpers as inputs
void configRobotIO(void){

    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN0); // Front left yellow LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN5); // Front Right yellow LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN7); // Back right red LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN6); // Back left Red LED

    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN0); // Set them all to Low L yellow
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN5); // R yellow
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN7); // R red
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN6); // L Red


    // Set bumpers as inputs
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN0);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN2);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN3);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN5);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN6);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN7);

    // Enable Interrupts
    MAP_GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN0);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN2);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN3);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN5);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN6);
    MAP_GPIO_enableInterrupt(GPIO_PORT_P4, GPIO_PIN7);


    MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN0,GPIO_HIGH_TO_LOW_TRANSITION); // Bumper 0
    MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN2,GPIO_HIGH_TO_LOW_TRANSITION); // Bumper 1
```

```
    MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN3,GPIO_HIGH_TO_LOW_TRANSITION); // Bumper 2
    MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN5,GPIO_HIGH_TO_LOW_TRANSITION); // Bumper 3
    MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN6,GPIO_HIGH_TO_LOW_TRANSITION); // Bumper 4
    MAP_GPIO_interruptEdgeSelect(GPIO_PORT_P4, GPIO_PIN7,GPIO_HIGH_TO_LOW_TRANSITION); // Bumper 5

    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, GPIO_PIN0); // B#0
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, GPIO_PIN2); // B#1
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, GPIO_PIN3); // B#2
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, GPIO_PIN5); // B#3
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, GPIO_PIN6); // B#4
    MAP_GPIO_clearInterruptFlag(GPIO_PORT_P4, GPIO_PIN7); // B#5



     // Right
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN0); // Enable Right
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN6); // PWM Right
     MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2,GPIO_PIN6, GPIO_PRIMARY_MODULE_FUNCTION);

     // Left
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN2); // Enable Left wheel
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Right wheel
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN7); // PWM Left wheel
     MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2,GPIO_PIN7, GPIO_PRIMARY_MODULE_FUNCTION);

     // Add Callback fuction
     MAP_GPIO_registerInterrupt(GPIO_PORT_P4, bumperSwitchesHandler);

     // Right
     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN0); // Enable Right wheel
     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN6); // PWM Right wheel


     // Left
     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN2); // Enable Left wheel
     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel
     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN7); // PWM Left wheel


     // Config Sensor
     MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN3); // CNTL EVEN
     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN3);


     MAP_GPIO_setAsOutputPin(GPIO_PORT_P9, GPIO_PIN2); // CNTL ODD
     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P9, GPIO_PIN2);







}



void powerUp(void){

    //TOGGLE LED 1 ONCE
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);   // LED1 ON-RED

    // RSLK LED TOGGLE ONCE
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN0); // Set them all to Low L yellow
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN5); // R yellow
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN7); // R red
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN6); // L Red

    // TOGGLE LED2 WHITE
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);    // LED2 Red
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);   // LED2 GREEN
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);   // LED2 BLUE
    __delay_cycles(3000000);

    //TOGGLE LED 1 ONCE
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);   // LED1 ON-RED

    // RSLK LED TOGGLE ONCE
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN0); // Set them all to Low L yellow
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN5); // R yellow
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN7); // R red
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN6); // L Red

    // TOGGLE LED2 WHITE
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);    // LED2 Red
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);   // LED2 GREEN
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);   // LED2 BLUE
    __delay_cycles(3000000);
```

```
282
283     // WHEELS ARE OFF
284     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
285     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel
286
287 }
288
289 void standBy(void) {
290
291
292     // TOGGLE 1 HZ
293     MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);   // LED1 ON-RED
294     __delay_cycles(3000000);
295     MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);   // LED1 ON-RED
296     __delay_cycles(3000000);
297
298     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);    // LED2 Red
299     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);   // LED2 BLUE
300     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);   // GREEN ON
301
302     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
303     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel
304 }
305
306
307 void tableII(void){
308
309     // TABLE 2 LED 2 COLORS
310
311     if  ((LEFT == 0) && (RIGHT == 0)){// OFF
312
313             MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Blue ON
314             MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);    // Green ON
315             MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);    // Red OFF
316
317         }
318
319     else if ((LEFT > 0) && (RIGHT == 0)){ // Red
320         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0); // Red ON
321
322         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Blue OFF
323         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); // Green OFF
324
325
326     }
327
328     else if ((RIGHT > 0) && (LEFT == 0)){ // Blue
329
330             MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Blue ON
331
332             MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Green OFF
333             MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);    // Red OFF
334
335         }
336
337     else if (LEFT > RIGHT){ // Yellow
338
339         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0); // Red ON
340         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1); // Green ON
341
342         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Blue OFF
343     }
344
345     else if (LEFT == RIGHT){ // Green
346
347         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Green ON
348
349         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Blue OFF
350         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);    // Red OFF
351
352     }
353
354     else { // Cyan
355
356         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);    // Blue ON
357         MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);    // Green ON
358
359         MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);    // Red OFF
360
361     }
362
363
364
365
366
367
368
369
370 }
371
372
373
374
375 void rotate(Condition Cond) {
```

```
void rotate(Condition Cond) {


        if (Cond == ALL_LEFT){
           if(stabl == reset){
           configPWMTimer(PERIOD,CLOCKDIVIDER,35,LEFTCHANNEL);
           configPWMTimer(PERIOD,CLOCKDIVIDER,50,RIGHTCHANNEL);
           //izer = stable;
           }
           else{
               configPWMTimer(PERIOD,CLOCKDIVIDER,35,LEFTCHANNEL);
               configPWMTimer(PERIOD,CLOCKDIVIDER,26,RIGHTCHANNEL);

           }

            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel


            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel

            stabr = reset;
            izer = stable;
            wturn = lastLeft;

        }

        else if (Cond == MORE_LEFT){ // FRONT ON REAR OFF

          configPWMTimer(100,CLOCKDIVIDER,5,LEFTCHANNEL); // Start L wheel timer
           configPWMTimer(100,CLOCKDIVIDER,5,RIGHTCHANNEL);

            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel

            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel

            izer = reset;
            stabr = reset;
            wturn = lastLeft;
            gturn = glastLeft;
        }
        else if (Cond == MIDDLE){ // FRONT ON REAR OFF

//              if(gturn == glastLeft){
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,25,LEFTCHANNEL); // Start L wheel timer
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,22,RIGHTCHANNEL);
//              }
//              else if(gturn == glastRight){
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,22,LEFTCHANNEL); // Start L wheel timer
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,25,RIGHTCHANNEL);
//              }
//              else if(gturn == gwlastLeft){
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,25,LEFTCHANNEL);
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,22,RIGHTCHANNEL);
//              }
//              else if(gturn == gwlastRight){
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,22,LEFTCHANNEL); // Start L wheel timer
//                  configPWMTimer(PERIOD,CLOCKDIVIDER,25,RIGHTCHANNEL);
//              }
//              else{
              configPWMTimer(PERIOD,CLOCKDIVIDER,50,LEFTCHANNEL);
              configPWMTimer(PERIOD,CLOCKDIVIDER,50,RIGHTCHANNEL);
 //             }

                    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
                    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel

                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel


                    izer = stable;
                    aturn = wMiddle;
                    stabl = reset;
                    stabr = reset;
         }
        else if (Cond == MORE_RIGHT){ // FRONT ON REAR OFF
                  configPWMTimer(100,CLOCKDIVIDER,5,LEFTCHANNEL); // Start L wheel timer
                   configPWMTimer(100,CLOCKDIVIDER,5,RIGHTCHANNEL);

                    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel

                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel


                    izer = reset;
                    stabl = reset;
                    wturn = lastRight;
```

```
                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel


                    izer = stable;
                    aturn = wMiddle;
                    stabl = reset;
                    stabr = reset;
         }
        else if (Cond == MORE_RIGHT){ // FRONT ON REAR OFF
                  configPWMTimer(100,CLOCKDIVIDER,5,LEFTCHANNEL); // Start L wheel timer
                   configPWMTimer(100,CLOCKDIVIDER,5,RIGHTCHANNEL);

                    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel

                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
                    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel


                    izer = reset;
                    stabl = reset;
                    wturn = lastRight;
                    gturn = glastRight;
                  }
        else if (Cond == ALL_RIGHT){ // FRONT ON REAR OFF

            if(stabr == reset){
              configPWMTimer(PERIOD,CLOCKDIVIDER,50,LEFTCHANNEL); // Start L wheel timer
              configPWMTimer(PERIOD,CLOCKDIVIDER,35,RIGHTCHANNEL);
              //izer = stable;
            }
            else{
               configPWMTimer(PERIOD,CLOCKDIVIDER,26,LEFTCHANNEL); // Start L wheel timer
                configPWMTimer(PERIOD,CLOCKDIVIDER,35,RIGHTCHANNEL);

            }

               MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
               MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel

             MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel


//
              wturn = lastRight;
              izer = stable;
              stabl = reset;
//
               }

        else { // FRONT OFF REAR ON

           configPWMTimer(100,CLOCKDIVIDER,5,LEFTCHANNEL); // Start L wheel timer
           configPWMTimer(100,CLOCKDIVIDER,5,RIGHTCHANNEL);



            if(wturn == lastLeft){
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
            aturn = wLeft;
            }
            else{
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN5); // Direction Right wheel
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); // Direction Left wheel
            aturn = wRight;
            }

            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel

            izer = reset;
            stabl = stable;
            stabr = stable;


        }
}

  void bumperSwitchesHandler(){

        uint16_t status;
        __delay_cycles(300000);

        status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P4);

        switch(status){

        case GPIO_PIN0: // B0
            if(b0 == buttonON){
                b0 = buttonOFF;
            }
```

```
                b0 = buttonOFF;
            }
            else {
                b0 = buttonON;
            }
            break;


        }

    }









    void Input_Output(void){

        NumSensor = 0;

        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN3); // CNTL EVEN HIGH
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P9, GPIO_PIN2); // CNTL ODD HIGH


        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN0); // QTR0
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN1); // QTR1
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN2); // QTR2
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN3); // QTR3
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN4); // QTR4
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN5); // QTR5
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN6); // QTR6
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN7); // QTR7

        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN0);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN1);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN2);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN3);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN4);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN5);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN6);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN7);

        __delay_cycles(30); // Delay 10 us

        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN0); // QTR0
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN1); // QTR1
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN2); // QTR2
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN3); // QTR3
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN4); // QTR4
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN5); // QTR5
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN6); // QTR6
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN7); // QTR7
        __delay_cycles(3000); // Delay 10 ms


        __delay_cycles(30);

        uint8_t pin;

        for (pin = 0; pin < 8; pin++) {

          if (MAP_GPIO_getInputPinValue(GPIO_PORT_P7, (1 << pin))) {

            NumSensor |= (1 << pin); // Set the corresponding bit in sensorVals

          }

        }



        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN3); // CNTL EVEN LOW
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P9, GPIO_PIN2); // CNTL ODD LOW



    }

    void readSensor(void){

        LEFT = 0;
        RIGHT = 0;


            if (NumSensor & BIT0){ // Right
                RIGHT++;
            }

            if (NumSensor & BIT1){
                RIGHT++;
```

```
                RIGHT++;
            }

            if (NumSensor & BIT2){
                RIGHT++;
            }

            if (NumSensor & BIT3){
                RIGHT++;
            }



            if (NumSensor & BIT4){ // Left
                LEFT++;
            }

            if (NumSensor & BIT5){
                LEFT++;
            }

            if (NumSensor & BIT6){
                LEFT++;
            }

            if (NumSensor & BIT7){
                LEFT++;
            }

            if  ((LEFT == 0) && (RIGHT == 0)){// OFF
                Con = OFF;
            }

            else if ((LEFT > 0) && (RIGHT == 0)){ // Red
                Con = ALL_LEFT;
            }

            else if ((RIGHT > 0) && (LEFT == 0)){ // Blue
                Con = ALL_RIGHT;
            }

            else if (LEFT > (1+RIGHT)){ // Yellow
                Con = MORE_LEFT;
            }

            else if (LEFT == RIGHT){ // Green
                Con = MIDDLE;
            }

            else if(RIGHT > (1+LEFT)) { // Cyan
                Con = MORE_RIGHT;
            }

    }


    void Tracking(void){

        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0); // RED LED1
        tableII();

    }


    void Lost(void){

        // LED1
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1,GPIO_PIN0);

        // LED2
        // TOGGLE LED2 WHITE
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);    // LED2 Red
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);   // LED2 GREEN
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);   // LED2 BLUE

        // RSLK
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN0); // Set them all to Low L yellow
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN5); // R yellow
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN7); // R red
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN6); // L Red

        // Wheels
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); // Sleep Right wheel
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); // Sleep Left wheel



    }

void RGBDriver(led2 LEDColor)
```

```
721 void RGBDriver(led2 LEDColor)
722
723 {
724
725   switch (LEDColor)
726
727   {
728
729   case LED2OFF:
730
731     // RED AND GREEN AND BLUE OFF
732
733     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
734
735     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
736
737     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);
738
739     break;
740
741   case RED:
742
743     // RED ON
744
745     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);
746
747     // GREEN AND BLUE OFF
748
749     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
750
751     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);
752
753     break;
754
755   case GREEN:
756
757     // GREEN OFF
758
759     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);
760
761     // RED AND BLUE OFF
762
763     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
764
765     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);
766
767     break;
768
769   case BLUE:
770
771     // BLUE ON
772
773     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);
774
775     // RED AND GREEN OFF
776
777     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
778
779     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);
780
781     break;
782
783   case CYAN:
784
785     // BLUE AND GREEN ON
786
787     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);
788
789     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);
790
791     // RED OFF
792
793     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0);
794
795     break;
796
797   case YELLOW:
798
799     // RED AND GREEN ON
800
801     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);
802
803     MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);
804
805     // BLUE OFF
806
807     MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2);
808
809     break;
810
811   case MAGENTA:
812
813     // RED AND BLUE ON
814
```

```
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);

    // GREEN OFF

    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1);

    break;

  case WHITE:

    // RED AND GREEN AND BLUE ON

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0);

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1);

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2);

    break;

  case LED2MAINTAIN:

    break;

  }
}
void sensorReader(void){

    switch (Con)

  {

  case ALL_LEFT:
    RGBDriver(RED);
    rotate(ALL_LEFT);
    break;

  case MORE_LEFT:
    RGBDriver(YELLOW);
    rotate(MORE_LEFT);
    break;

  case MIDDLE:
    RGBDriver(GREEN);
    rotate(MIDDLE);
    break;

  case MORE_RIGHT:
    RGBDriver(CYAN);
    rotate(MORE_RIGHT);

    break;

  case ALL_RIGHT:
    RGBDriver(BLUE);
    rotate(ALL_RIGHT);
    break;

  case OFF:
    RGBDriver(WHITE);
    rotate(OFF);
    break;

  }

  if (Con != OFF)

  {

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);

  } else

  {

    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);

  }
}
```

5. Experimental Results

Table I: Certification Sheet for Lab 10

#	Test condition	Expected result, LEDs	Actual result	Pass/fail
1	Power-up 5 pts	LED1 toggles once LED2 toggles White once Wheels are OFF	Powers On	☒Pass ☐Fail
2	Stand-by 5 pts	LED1 toggling LED2 OFF Wheels are OFF	Waits for Bumper	☒Pass ☐Fail
3	Bumper Switch is used to activate the design 5 pts	LED1 N/A LED2 Table I	Bumper Started	☒Pass ☐Fail
4	REDLED1 indicates that RSLK is active. 5 pts	LED1 User Defined LED2 Table I	Active	☒Pass ☐Fail
5	Table I being followed for RSLK LEDS 5 pts	LED1 N/A LED2 Table I	Followed Table	☒Pass ☐Fail
6	Score on Track from 0 to 75 pts	LED1 N/A LED2 N/A	75	☒Pass ☐Fail
7	Time on track in second. Used for bonuses	LED1 N/A LED2 N/A	32	☒Pass ☐Fail
8			Click or tap here to enter text.	☒Pass ☐Fail
		Total Score		% Passed =100

TA Authorization Code is Sunday32

6. Summary

This lab was the final lab, which challenges students with a course that their TI-RSLK MAX must follow. The algorithm used throughout the weeks previously varied using the same strategy as the labs. The changes made between weeks one and two were significant to the design's success. Originally, it only contained a single type of turn, this was adjusted by adding fast and slow turns to the algorithm, allowing for sharp and curved turns. Along with that, the code keeps track of the previous direction, which then plugs it into an algorithm that considers the next direction. Table I shows that all seven tests were successfully completed, giving a pass rate of 100%. The experimental error for this design was merely zero, with room for argument to improve its current lap time. Overall, no design modifications were needed post-certification, as the lab team in question was able to create a design that completed the assigned tasks correctly.

7. Lessons Learned

This lab was the most advanced challenge given so far. One of the first learned things was how to successfully manipulate the speed of the wheels using the PWM timer and its duty value. This change helped the TI-RSLK MAX move faster through the course. The second learned topic was about adding a feature that can allow the robot to "navigate" without a line present. This helped the robot complete the 180-degree turn without the use of the QTRK line sensor. The final learned topic from this entire semester would be relating to states, more specifically, the sensor states. The implementation of more types of turns allows for the robot to navigate smoothly, compared to its jumpy,

robotic version from the previous Lab's. The addition of having the past direction stored allowed the robot to make perfect decisions. Since this design was able to successfully navigate the maze in an exceptional score, there is no need to change anything. This lab was arguably performed in an ideal way, with no obvious room for improvement.