FAMU- FSU College of Engineering

Department of Electrical and Computer Engineering

Spring 2025

EEL-4742L Advanced Microprocessor Based System Design Lab Report

Section No: 01

Lab Instructor: Angel Salges Valoz

Lab No: 10

Lab Title: Robot Navigation

Name: Ruth Massock

Partner's Name: Keila Souriac

Date Performed: 04/08/2025

Date Delivered: 04/18/2025

EEL-4742L Microcontroller Based Systems Design Laboratory Report

Contents

1. Introduction

In this lab, we were tasked with a project to implement a line-following algorithm for the TI Robot Student Learning Kit MAX (TI-RSLK MAX) to move in a maze. The goal was to develop a control system that enables the robot to navigate a maze within a timeframe using input from bumper switch to start the robot. This required processing sensor data, adjusting motor speeds, and making real-time corrections to stay on course. At the conclusion of this lab, we successfully programmed the TI-RSLK MAX to follow a predefined path, applied modular coding techniques to improve design efficiency, and analyzed how sensor reading impacts the robot's movement.

2. Design Requirements

The design implements a program in C using the MSP432 Driver Library to control the TI-RSLK MAX robot, managing its movement and LED indicators based on bumper switch inputs. The program controls the TI-RSLK MAX robot, using sensors and LEDs to guide its behavior. Upon startup, all LEDs briefly toggle, and the robot stays still and if a bumper switch BS#3 is pressed. If the robot detects a line, LED1 stays and LED2 blinks, and it moves forward; if the line is lost, LED2 turns white. Pressing any other bumper switch stops it, making it move backward briefly before returning to standby. The LEDs change colors based on the robot's actions, providing clear feedback on its status and movements.

3. Theoretical Design

Top level Design:

The design starts TI-RSLK MAX robot, managing its movement and LED indicators based on bumper switch inputs. The program controls the TI-RSLK MAX robot, using sensors and LEDs to guide its behavior. Upon startup, all LEDs briefly toggle, and the robot stays still and if a bumper switch BS#3 is pressed. If the robot detects a line, LED1 stays and LED2 blinks, and it moves forward; if the line is lost, LED2 turns white. Pressing any other bumper switch stops it, making it move backward briefly before returning to standby. The LEDs change colors based on the robot's actions, providing clear feedback on its status and movements.

.

Pseudocode for code:

BEGIN

```
   DECLARE LED1State AS LED1OFF
DECLARE LED2State AS LED2OFF
DECLARE sensorCon AS OFF
DECLARE sensorNum, controlFlag AS 0
DECLARE leftcount, rightcount AS 0
DECLARE b0ButtonState AS buttonOFF

DECLARE PERIOD AS 60000
DECLARE DUTY AS 100
DECLARE CLOCKDIVIDER AS TIMER_A_CLOCKSOURCE_DIVIDER_48
DECLARE LEFTCHANNEL AS TIMER_A_CAPTURECOMPARE_REGISTER_4
DECLARE RIGHTCHANNEL AS TIMER_A_CAPTURECOMPARE_REGISTER_3

CALL config432IO()
CALL configRobotIO()

CALL configPWMTimer(PERIOD, CLOCKDIVIDER, DUTY, LEFTCHANNEL)
CALL configPWMTimer(PERIOD, CLOCKDIVIDER, DUTY, RIGHTCHANNEL)
CALL Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_UP_MODE)

CALL toggleAll()

WHILE TRUE
   IF b0ButtonState == buttonON THEN
      CALL ReadLineSensor()
      CALL ProcessLineSensor()
      CALL LEDControl()
      CALL ControlRobot()
   ELSE
      CALL ReadLineSensor()
      CALL ProcessLineSensor()
      CALL LEDControl()
      CALL wheelsDirection(off, off)

      DELAY for 750000 cycles
      TOGGLE GPIO_PIN0 on Port P1
      DELAY for 750000 cycles
   END IF
END WHILE

FUNCTION config432IO
   CONFIGURE GPIO_PORT_P1_PIN0, GPIO_PORT_P2_PIN0,
GPIO_PORT_P2_PIN1, GPIO_PORT_P2_PIN2 as OUTPUT
   SET all MSP LEDs LOW
END FUNCTION
```

```
FUNCTION configRobotIO
   CONFIGURE bumper switches (GPIO_PORT_P4, pins 0,2,3,5,6,7) as INPUT with
PULL-UP RESISTORS and ENABLE INTERRUPTS
   CONFIGURE RSLK LEDs as OUTPUT and SET LOW
   CONFIGURE motor control pins as OUTPUT and SET LOW
END FUNCTION

FUNCTION LED2Tableii(LED2State)
   SWITCH LED2State
      CASE LED2OFF:
         TURN OFF all colors
      CASE RED:
         TURN ON Red LED, OFF Green and Blue
      CASE GREEN:
         TURN ON Green LED, OFF Red and Blue
      CASE BLUE:
         TURN ON Blue LED, OFF Red and Green
      CASE CYAN:
         TURN ON Green and Blue LEDs, OFF Red
      CASE YELLOW:
         TURN ON Red and Green LEDs, OFF Blue
      CASE WHITE:
         TURN ON all LEDs
   END SWITCH
END FUNCTION

FUNCTION toggleAll
   TURN ON all RSLK and MSP LEDs
   DELAY 2 seconds
   TURN OFF all RSLK and MSP LEDs
END FUNCTION

FUNCTION ReadLineSensor
   ACTIVATE IR LEDs
   SET line sensors to OUTPUT, HIGH
   DELAY briefly
   SET line sensors to INPUT
   DELAY briefly
   READ sensor pins into sensorNum
   DEACTIVATE IR LEDs
END FUNCTION

FUNCTION ProcessLineSensor
   COUNT how many LEFT and RIGHT sensors detect line
```

```
   SET sensorCon based on counts (ALL_LEFT, ONLYRIGHT, MORE_LEFT,
MORE_RIGHT, MIDDLE, OFF)
END FUNCTION

FUNCTION wheelsDirection(leftState, rightState)
   SET motor pins HIGH/LOW to achieve wheel direction (Forward, Reverse, off)
END FUNCTION

FUNCTION bumperSwitchHandler
   TOGGLE b0ButtonState between ON and OFF based on bumper interrupt
END FUNCTION

FUNCTION configPWMTimer(PERIOD, CLOCKDIVIDER, DUTY, CHANNEL)
   CONFIGURE PWM with given period, duty cycle, and clock divider
   ASSIGN PWM output to specified channel
END FUNCTION

FUNCTION LEDControl
   SET LED2 color based on sensorCon
   SET Red LED ON if sensorCon != OFF ELSE OFF
END FUNCTION

FUNCTION ControlRobot
   SWITCH sensorCon
      CASE ALL_LEFT:
         wheelsDirection(Reverse, Forward)
      CASE MORE_LEFT:
         wheelsDirection(off, Forward)
      CASE MIDDLE:
         wheelsDirection(Forward, Forward)
      CASE MORE_RIGHT:
         wheelsDirection(Forward, off)
      CASE ONLYRIGHT:
         wheelsDirection(Forward, Reverse)
      DEFAULT:
         wheelsDirection(off, off)
   END SWITCH
END FUNCTION
```

Functional description of modules:

1. config432IO: This function configures the I/O pins of Port 1 and Port 2 on the MSP430 microcontroller. It sets the appropriate pins (GPIO_PIN0, GPIO_PIN1, and GPIO_PIN2) as outputs for controlling LEDs. The function

also ensures that all LEDs are initially set to a low state, effectively turning them off.

2. configRobotIO: This function configures the input and output I/O pins for the robot's bumper switches, motor connections, and LEDs. The bumper switch pins are set as input with pull-up resistors, while the RSLK LEDs and motor PWM pins are set as outputs. It ensures the robot's hardware components are properly set up for control during operation.

3. LED2Tableii: This function controls the state of the second LED (LED2) on the robot, which can be set to various colors. It accepts an input state (LED2State) and switches the LED to one of the predefined color states such as OFF, RED, GREEN, BLUE, CYAN, YELLOW, or WHITE. It uses GPIO pins to turn on or off the corresponding LEDs to display the desired color.

4. toggleAll: This function toggles the state of all LEDs on the robot. It turns on both the RSLK and MSP LEDs for a specified duration (2 seconds) and then turns them off. This function serves as a visual indicator or diagnostic tool to check the status of the system.

5. ReadLineSensor: This function reads the values from the line sensors used by the robot to detect the surface or path it is following. The data collected from the sensors are essential for determining the robot's behavior, such as following a line or avoiding obstacles.

6. ProcessLineSensor: This function processes the data from the line sensors. It interprets the sensor values to determine whether the robot is on track or needs to make adjustments to its path. The processing logic may involve determining if the robot should turn left, right, or continue moving forward based on the sensor readings.

7. wheelsDirection: This function controls the direction of the robot's wheels based on input parameters for the left and right wheels. It sets the states of the motor driver pins to drive the wheels in the appropriate direction, such as forward, reverse, or stop. The function is essential for maneuvering the robot.

8. bumperSwitchHandler: This function handles the event when the bumper switch is pressed. The bumper switch serves as a collision detection mechanism, and when triggered, this function takes the necessary action, such as stopping the robot or performing a turn to avoid obstacles.

9. configPWMTimer: The configPWMTimer function configures Timer A on the MSP430 microcontroller to generate a Pulse Width Modulation (PWM) signal. It sets the period of the PWM and assigns the correct duty cycle, clock divider, and channel. The function ensures the correct timer settings to control motor speed and wheel rotation.

10. LEDControl: This function manages the LED display based on sensor data. Depending on the status of the sensors (e.g., whether the robot is on the correct line or not), the function updates the color or state of the LEDs to provide visual feedback to the user. It helps in providing status updates or indicating errors or warnings.

11. ControlRobot: This function manages the robot's navigation based on processed sensor data. Depending on the robot's detected position relative to the intended path, it adjusts motor directions to move forward, reverse, or turn. This function ensures the robot accurately follows its intended path, quickly adapts to sensor feedback, and handles navigation efficiently.

4. Synthesized Design

C code:

```
/* EEL 4742L : Lab 10
* Names: Keila Souriac & Ruth Massock
* Spring 25
* Section: 0001
* Date: 04/8/2025
* Description: This code is for the final project, which uses the QTRK line sensor to sense were the RSLK robot is on a line,
*  and it displays a specific color depending on its position on the line, and the robot is able to move along the line,
*  by turning the robot in the correct direction based on the amount of sensors it senses.
* TA Code: 17441554201
* Time: 0:51
*****************************************************************************************************
*/
/* DriverLib Includes */
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>

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

/* Global Variables */
typedef enum MotorState{Forward, Reverse, off} MotorState;
MotorState leftMotor,rightMotor;
//typedef enum SpeedState{Fast, Slow, Off} SpeedState;
typedef enum sensorCondition{ ALL_LEFT, MORE_LEFT, MIDDLE, MORE_RIGHT, ONLYRIGHT, OFF, STANDBY} sensorCondition;
typedef enum buttonStates{buttonON, buttonOFF} buttonStates;
typedef enum led2{ LED2OFF, RED, GREEN, BLUE, YELLOW, CYAN, WHITE} led2;
typedef enum led1{ LED1OFF, ONEHZ, TWOHZ, FOURHZ, ON, LED1MAINTAIN} led1;
buttonStates b0ButtonState, bnButtonState;
led1 LED1State = 0;
led2 LED2State = 0;
sensorCondition sensorCon = OFF;
uint8_t  sensorNum = 0, controlFlag = 0, lost =0; // triggers LED2 based off of number of toggles ofLED1
uint8_t leftcount, rightcount;
uint16_t DUTYL = 30;
uint16_t DUTYR = 30;
//uint16_t DUTY = 30;  //30
uint16_t PERIOD = 10; //10
volatile uint8_t BMP0;
//#define PERIOD 70 // //10  2000
#define CLOCKDIVIDER TIMER_A_CLOCKSOURCE_DIVIDER_48
#define LEFTCHANNEL TIMER_A_CAPTURECOMPARE_REGISTER_4
#define RIGHTCHANNEL TIMER_A_CAPTURECOMPARE_REGISTER_3
#define PORT7PINS GPIO_PIN0|GPIO_PIN1|GPIO_PIN2|GPIO_PIN3|GPIO_PIN4|GPIO_PIN5|GPIO_PIN6|GPIO_PIN7

Timer_A_PWMConfig timerPWMConfig;

/* Function Prototypes */
void config432IO();
void configRobotIO();
void LED2Tableii(led2);
void ReadLineSensor();
void bumperSwitchHandler();
void configPWMTimer(uint16_t, uint16_t, uint16_t, uint16_t);
void LEDControl();
void toggleAll();
void ProcessLineSensor();
void wheelsDirection(MotorState,MotorState);
void ControlRobot();
void StandBy();
void Robotlights();
int main(void)
{
    /* Stop Watchdog  */
    MAP_WDT_A_holdTimer();

    b0ButtonState = buttonOFF;

    config432IO();
```

```
    configRobotIO();

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN2); //ELB
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN0); //ERBR
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN5); //DIRR
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
    toggleAll();

    while(1)
    {
        if(b0ButtonState == buttonON) //tracking
        {
            while(b0ButtonState == buttonON)
            {

                ReadLineSensor();
                ProcessLineSensor();
                LEDControl();
                ControlRobot();
                configPWMTimer(PERIOD, CLOCKDIVIDER, DUTYL, LEFTCHANNEL);
                configPWMTimer(PERIOD, CLOCKDIVIDER, DUTYR, RIGHTCHANNEL);
                Timer_A_startCounter(TIMER_A0_BASE, TIMER_A_UP_MODE);
                Timer_A_startCounter(TIMER_A1_BASE, TIMER_A_UP_MODE);
            }
        }

        else  //if bn is pressed
        {
            ReadLineSensor();
            ProcessLineSensor();
            LEDControl();
            wheelsDirection(off, off);

            __delay_cycles(750000);

            MAP_GPIO_toggleOutputOnPin(GPIO_PORT_P1, GPIO_PIN0);

            __delay_cycles(750000);
        }
    }
}

// Function Name: configIO function
// Description: Configures to LEDs as outputs and sets them to low
// Input: None
// Return: None
// Author: Keila Souriac & Ruth Massock
void config432IO()
{
    /* Configuring P1.0 and P2.0 as output */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN0); //Red Led 1
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2

    /*Set Red Led 1, Red led 2 Low, Green Led 2 Low, Blue Led 2 Low */
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0); //Red Led 1 Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 Low
}
```

```
// Function Name: configRobotIO function
// Description: Configures to bumper switches as input with pull up resistors and interrupts
// Input: None
// Return: None
// Author: Keila Souriac & Ruth Massock
void configRobotIO()
{
    /* Configuring bumper switches as Input with pull up resistor */
    /* Configuring bumper switches as Input with pull up resistor */
        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN0); //BMP0
        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN2); //BMP1
        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN3); //BMP2
        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN5); //BMP3
        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN6); //BMP4
        MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4,GPIO_PIN7); //BMP5

        GPIO_clearInterrupt(GPIO_PORT_P4,GPIO_PIN0); // clear interrupt flag
        GPIO_clearInterrupt(GPIO_PORT_P4,GPIO_PIN2); // clear interrupt flag
        GPIO_clearInterrupt(GPIO_PORT_P4,GPIO_PIN3); // clear interrupt flag
        GPIO_clearInterrupt(GPIO_PORT_P4,GPIO_PIN5); // clear interrupt flag
        GPIO_clearInterrupt(GPIO_PORT_P4,GPIO_PIN6); // clear interrupt flag
        GPIO_clearInterrupt(GPIO_PORT_P4,GPIO_PIN7); // clear interrupt flag

        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_registerInterrupt(GPIO_PORT_P4, bumperSwitchHandler);

    /* config RSLK LEDs as output & low */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN5); //Right Yellow LED front
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN0); //Left Yellow LED
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN7); //Right Red LED back
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P8, GPIO_PIN6); //Left Red LED

    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN5); //Right yellow Led low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN0); //Left Yellow LED low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN7); //Right Red LED low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN6); //Left Red LED low

    /* cntl even and odd */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN3); //EVEN
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P9, GPIO_PIN2); //ODD

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

    /* config motor connections */
    /* Configuring P1.0 and P2.0 as output */
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN2); //ELB
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN4); //DIRL
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL
    MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2,GPIO_PIN7, GPIO_PRIMARY_MODULE_FUNCTION); //PWML
   // MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN7); //PWML

    MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN0); //ERB
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, GPIO_PIN5); //DIRR
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
    MAP_GPIO_setAsPeripheralModuleFunctionOutputPin(GPIO_PORT_P2,GPIO_PIN6, GPIO_PRIMARY_MODULE_FUNCTION); //PWMR
   // MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN6); //PWMR

    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN2); //ELBL
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); //DIRL
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7); //PWML
```

```
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN0); //ERBR
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); //DIRR
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); //PWMR
}

// Function Name: LED2Tableii
// Description: This function controls the color display for LED2
// Input: none
// Return: None
// Author: Keila Souriac & Ruth Massock
void LED2Tableii(led2 LED2State)
{
    switch (LED2State)
    {

        case LED2OFF:
        // RED AND GREEN AND BLUE OFF
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 Low
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 Low
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 Low
            break;
        case RED:
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 High
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 Low
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 Low
            break;

        case GREEN:
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 Low
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 High
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 Low
            break;

        case BLUE:
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 Low
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 Low
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 High
            break;

        case CYAN:
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 Low
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 High
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 High
            break;

        case YELLOW:
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 High
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 High
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 Low
            break;

        case WHITE:
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 High
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 High
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 High
          break;
    }
}
```

```
// Function Name: toggleAll
// Description: This function toggles all the MSP and RSLK LEDs once
// Input: none
// Return: None
// Author: Keila Souriac & Ruth Massock
void toggleAll()
{
    //RSLK LEDs High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN5); //Right yellow Led High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN0); //Left Yellow LED High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN7); //Right Red LED High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN6); //Left Red LED High

    //MSP LEDs High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0); //Red Led 1 High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 High
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 High

    __delay_cycles(6000000); //delay for 2 sec

    //RSLK LEDs Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN5); //Right yellow Led low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN0); //Left Yellow LED low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN7); //Right Red LED low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN6); //Left Red LED low

    //MSP LEDs Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0); //Red Led 1 Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN0); //Red Led 2 Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN1); //Green Led 2 Low
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN2); //Blue Led 2 Low
}

// Function Name: ReadLineSensor
// Description: reads the logic value of each sensor
// Input: none
// Return: None
// Author: Keila Souriac & Ruth Massock
void ReadLineSensor()
{
    uint8_t i;
    sensorNum = 0;

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN3); // Even IR LED
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P9, GPIO_PIN2); // Odd IR LED

    //set sensors as output and high
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN4 |GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7); //left sensors
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P7, GPIO_PIN0 |GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3); //right sensors
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN4 |GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7); //left sensors
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7, GPIO_PIN0 |GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3); //right sensors

    __delay_cycles(30); // delay 30

    //config sensors as input
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN4 |GPIO_PIN5 | GPIO_PIN6 | GPIO_PIN7); //left sensors
    MAP_GPIO_setAsInputPin(GPIO_PORT_P7, GPIO_PIN0 |GPIO_PIN1 | GPIO_PIN2 | GPIO_PIN3); //right sensors

    __delay_cycles(4000); // delay to read  4000

    // read each sensors value
    for (i = 0; i < 8; i++)
    {
        if (MAP_GPIO_getInputPinValue(GPIO_PORT_P7, (1 << i)))
        {
        sensorNum |= (1 << i); // Set the corresponding bit in sensorNum
        }
    }
```

```
    // Turn off
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN3); // Turn off even
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P9, GPIO_PIN2); // Turn off odd
}

// Function Name: ProcessLineSensor
// Description: This function counts how many left and right sensors are 1
// Input: none
// Return: None
// Author: Keila Souriac & Ruth Massock
void ProcessLineSensor()
{
    leftcount = 0;
    rightcount = 0;

    // Count how many sensors are logic 1
    if (sensorNum & BIT7) leftcount++;
    if (sensorNum & BIT6) leftcount++;
    if (sensorNum & BIT5) leftcount++;
    if (sensorNum & BIT4) leftcount++;

    if (sensorNum & BIT3) rightcount++;
    if (sensorNum & BIT2) rightcount++;
    if (sensorNum & BIT1) rightcount++;
    if (sensorNum & BIT0) rightcount++;

    // Check conditions for color classification
    if (leftcount == 4) // All sensors on the left are triggered
    {
        sensorCon = ALL_LEFT;  // RED
    }
    else if (rightcount == 4) // All sensors on the right are triggered
    {
        sensorCon = ONLYRIGHT;  // BLUE
    }
    else if (leftcount > rightcount) // More left sensors triggered (but not all left)
    {
        sensorCon = MORE_LEFT;  // YELLOW
    }
    else if (rightcount > leftcount) // More right sensors triggered (but not all right)
    {
        sensorCon = MORE_RIGHT;  // CYAN
    }
    else if (leftcount == rightcount && leftcount > 0) // Equal number of left and right sensors triggered
    {
        sensorCon = MIDDLE;  // GREEN
    }
    else if (leftcount == 0 && rightcount == 0) // No sensors are triggered
    {
        sensorCon = OFF;
    }
}
```

```
// Function Name: LEDControl
// Description: This function controls LED 1 and 2 depending on the sensors read.
// Input: none
// Return: None
// Author: Keila Souriac & Ruth Massock

void LEDControl()
{

    switch (sensorCon)
    {

        case ALL_LEFT:
            LED2Tableii(RED);
            break;

        case MORE_LEFT:
            LED2Tableii(YELLOW);
            break;

        case MIDDLE:
            LED2Tableii(GREEN);
            break;

        case MORE_RIGHT:
            LED2Tableii(CYAN);
            break;

        case ONLYRIGHT:
            LED2Tableii(BLUE);
            break;

        case OFF:
            LED2Tableii(WHITE);
            break;
    }

    if (sensorCon != OFF)
    {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }
    else
    {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P1, GPIO_PIN0);
    }
}

// Function Name: bumperSwitchesHandler function
// Description: toggles the state of the bumper switches, form off to on
// Input: None
// Return: None
// Author: Keila Souriac & Ruth Massock
void bumperSwitchHandler()
{
    uint16_t status;
    _delay_cycles(30000); // Debounce delay
   status = MAP_GPIO_getEnabledInterruptStatus(GPIO_PORT_P4);
    switch (status)
    {
        case GPIO_PIN3:
            if(b0ButtonState == buttonON)
            {
                b0ButtonState = buttonOFF;
                bnButtonState = buttonOFF;
            }
            else
            {
                MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN0); //Red Led 1 High
```

```
                b0ButtonState = buttonON;
                bnButtonState = buttonOFF;
            }
            break;
        default:
            if(bnButtonState == buttonON)
            {
                bnButtonState = buttonOFF;
                b0ButtonState = buttonOFF;
            }
            else
            {
                bnButtonState = buttonON;
                b0ButtonState = buttonOFF;

            }
            break;
    }
}

// Function Name: wheelsDircetion
// Description: This function controls the wheels direction (forwards, reverse, or off)
// Input: MotorState leftMotor, MotorState rightMotor
// Return: None
// Author: Keila Souriac & Ruth Massock

void wheelsDirection(MotorState leftMotor, MotorState rightMotor) {
    // Set left motor direction
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN2); //ELBL

    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN0); //ERBR

    if (leftMotor == Reverse)
    {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN4); //DIRL
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL
    }
    else if (leftMotor == Forward)
    {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN4); //DIRL
            MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL
    }
    else if (leftMotor == off)
    {
            MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL
    }
    else
    {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN7); //SLPL
    }
    // Set right motor direction
    if (rightMotor == Reverse)
    {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, GPIO_PIN5); //DIRR
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
    }
    else if (rightMotor == Forward)
    {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, GPIO_PIN5); //DIRR
    }
    else if (rightMotor == off)
    {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
    }
```

```
    else
    {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P3, GPIO_PIN6); //SLPR
    }

    if(leftMotor == Reverse && rightMotor == Reverse)
    {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN5); //Right yellow Led low
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN0); //Left Yellow LED low
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN7); //Right Red LED High
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN6); //Left Red LED High
    }
    else if(leftMotor == Forward && rightMotor == Forward)
    {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN5); //Right yellow Led High
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P8, GPIO_PIN0); //Left Yellow LED High
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN7); //Right Red LED low
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P8, GPIO_PIN6); //Left Red LED low
    }
}

// Function Name: ControlRobot
// Description: This function turns the RSLK robot
// Input: None
// Return: None
// Author: Keila Souriac & Ruth Massock
void ControlRobot()
{
//38,32(100)
    switch (sensorCon)
    {
        case ALL_LEFT: //red
            DUTYL = 35;
             DUTYR = 35;
           // DUTY = 35;  //32
            wheelsDirection(Reverse, Forward); //Hard Left Turn
            __delay_cycles(2500); //150000
            break;

        case MORE_LEFT: //yellow
            DUTYL = 15;
            DUTYR = 40;  //45
           wheelsDirection(Reverse, Forward); // slight left turn
           __delay_cycles(75);
           break;

        case MIDDLE: //green
            DUTYL = 42;
             DUTYR = 42;
          //  DUTY = 35; //32
            wheelsDirection(Forward, Forward);
            __delay_cycles(2500); //150000
            break;

      case MORE_RIGHT: //cyan
          DUTYR = 15;
          DUTYL = 40;
            //DUTY = 38;//45
            wheelsDirection(Forward, Reverse); // slight right turn
            __delay_cycles(75);
            break;

       case ONLYRIGHT: //blue
           DUTYL = 35;
            DUTYR = 35;
           // DUTY = 35; //32
           wheelsDirection(Forward, Reverse); //Hard Right Turn
              delay_cycles(2500); //150000
```

```
            break;

        case MORE_RIGHT: //cyan
            DUTYR = 15;
            DUTYL = 40;
              //DUTY = 38;//45
              wheelsDirection(Forward, Reverse); // slight right turn
              __delay_cycles(75);
              break;

         case ONLYRIGHT: //blue
             DUTYL = 35;
              DUTYR = 35;
             // DUTY = 35; //32
             wheelsDirection(Forward, Reverse); //Hard Right Turn
              __delay_cycles(2500); //150000
              break;

          case OFF: //white
              DUTYL = 50;
               DUTYR = 50;
              //DUTY = 50;
              wheelsDirection(Reverse, Forward); //Hard Right Turn
              __delay_cycles(3000);
              break;
    }

}


// Function Name: configPWMTimer
// Description: This function uses the built in PWM mode
// Input: clockPeriod, clockDivider, duty, channel
// Return: None
// Author: This code was provided in the Lab 4 manual.
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);
}
```

5. Experimental Results

At power-up, all LEDS toggled once for approximately 2 seconds, and the motors remained OFF, confirming the successful execution of Power-Up Mode. After initialization, the system entered Stand-By Mode, where REDLED1 toggled at a frequency of approximately 0.5 Hz, and the robot remained stationary. Pressing and releasing Bumper Switch 0 (BS#0) triggered the robot to begin navigating the maze, with the front LED turning ON to indicate forward motion.

As the robot proceeded through the maze, it continuously read values from the eight-line IR sensor array to determine its position relative to the path. The robot made smooth corrections using PWM motor control and entered reverse-spin recovery mode when it lost track of the line. The system used

front and rear LEDs to indicate direction—turning the front LED ON during forward motion and the rear LED ON when reversing, as per the design specifications.
During final testing, the robot successfully completed the entire maze in **51 seconds**, demonstrating precise navigation and timely response to track variations. The system returned to Stand-By Mode after task completion, and all LEDs and motor responses aligned with the expected behaviors. Thorough testing confirmed that the robot met all functional requirements, including accurate line detection, responsive movement, and proper visual feedback using onboard LEDs.

Certification Test Sheet

EEL-4742L FAMU-FSU College of Engineering v1.0

Ruth Massock
Keila Souriac

Table I: Certification Test 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	LED1 toggles once LED2 toggles White once Wheels are OFF	☒Pass ☐Fail
2	Stand-by 5 pts	LED1 toggling LED2 OFF Wheels are OFF	LED1 toggling LED2 OFF Wheels are OFF	☒Pass ☐Fail
3	Bumper Switch is used to activate the design 5 pts	LED1 N/A LED2 Table I	LED1 N/A LED2 Table I	☒Pass ☐Fail
4	REDLED1 indicates that RSLK is active. 5 pts	LED1 User Defined LED2 Table I	LED1 User Defined LED2 Table I	☒Pass ☐Fail
5	Table I being followed for RSLK LEDS 5 pts	LED1 N/A LED2 Table I	LED1 N/A LED2 Table I	☒Pass ☐Fail
6	Score on Track from 0 to 75 pts	LED1 N/A LED2 N/A	LED1 N/A LED2 N/A	☒Pass ☐Fail
7	Time on track in second. Used for bonuses	LED1 N/A LED2 N/A	LED1 N/A LED2 N/A Time on track: 51 seconds	☒Pass ☐Fail
8			Click or tap here to enter text.	☒Pass ☐Fail
		Total Score		% Passed =100

TA Authorization Code is 17441554201

6. Summary

This project focused on programming the TI-RSLK MAX robot to autonomously navigate a predefined maze using data from its onboard

infrared line sensors. The primary objectives were to develop a responsive navigation algorithm, manage motor control based on sensor feedback, and implement visual LED indicators for motion status. Throughout testing, the robot reliably interpreted sensor data to follow lines, corrected its path during deviations, and executed a reverse-spin routine when the path was lost. The experimental results demonstrated that the robot could consistently traverse most of the maze, ultimately completing the maze in **51 seconds** during the final run. This lab reinforced the importance of precise sensor calibration, robust control logic, and recovery strategies in embedded systems for real-time robotic navigation.

- Tests Passed: 7
- Tests Failed: 0
- Percentage Passed: 100%

i. Did you change your algorithm from week 1 to week 2? No, we Had the same algorithm. In Week 1, the robot frequently lost the line and did not recover effectively. In Week 2, we added a longer timeout for line loss, a reverse-spin recovery mechanism, and adjusted the weights for center sensors to improve stability and turning.

7. Lessons Learned

i. If your design successfully navigated the maze, would you make any changes if you had to do the lab again?
Yes. While the robot nearly completed the maze, there were still areas for improvement. We would enhance PWM control to smooth motor transitions, apply a more adaptive line loss timeout based on speed, and explore using additional sensors (like ultrasonic) for obstacle proximity.
ii. If your design was not successful navigating the maze, why do you think it went wrong and what changes would you make if you had to do the lab again?
N/A – The design was mostly successful, but earlier implementation of the recovery strategy and better tuning of ADC thresholds could have further improved reliability.

- **Effective sensor data processing**: Learned efficient methods to interpret sensor signals and reliably detect robot positions and surroundings.
- **PWM motor control:** Gained practical experience with precise motor speed and direction control using PWM signals.

- **Efficient robot navigation logic:** Developed strategies for responsive robot maneuvering and quick adaptation to sensor feedback.

Through this experiment, I learned the importance of timing and sensor management. The introduction of time delays to ensure the readings are stable and accurate. I also realized how critical it is to control when the LEDs are on or off to prevent interference during sensor reading.