San Jose State University
Department of Computer Engineering

CMPE 127 Lab Report

Final Project Report

Title: The Line Follower Robot

Semester: Fall 2019 **Date:** Dec 6, 2019

By

Name: Wenyi Cai	**Name:** Luan Nguyen
SID: 011232000	**SID:** 012652419S
Email: wenyi.cai@sjsu.edu	**Email:** luan.nguyen01@sjsu.edu

***Abstract*— Making a robot follows a given black line on a white given platform and searches the path to the treasure.**

I. Objective

The final project required to design a *Line Follower Maze Robot* using TI Robotics System Learning Kit by leveraging the skills learned in different phases of the lab throughout the semester. The goal of the project was to integrate the line sensor and the bump sensor to solve a maze that searches for a treasure. The project was a combinational lab with six modules: Module 6 for GPIO interface the line sensor, Module 10 for SysTick interrupts, Module 12 and 13 for bump sensor, motors and PWM control, and Module 13 for line follower building. To achieve the goal of the project, the robot should start from the start point on the platform, self-drive via the line, know how to find the right path to the treasure and how to back up for lost, and stop at the treasure spot.

II. Required Components

•Robot chassis, DC motors, and wheels.

•Motor drive and power distribution board.

Line IR sensors

• Bump sensors

• MSP432P401R MCU LaunchPad

• Rechargeable battery, pack of 6, metal hydride, 1300mAh, 1.2V, AA

III. Design Methodology

The first part of the lab was to test the reflectance of the sensor in module 6. At this step, there are nine cases need to be recorded:

1. The robot stayed at the center: Position >-47 && Position <47.
2. Slightly off to the left: Position <= -47 && Position > -142
3. Slightly off to the right: Position >= 47 && Position <142
4. Off to the left: Position <= -142 && Position >-237
5. Off to the right: Position >= 142 && Position < 237
6. Way off to the left: Position <= -237 && Position > -332
7. Way off to the right: Position >= 237 && Position < 332
8. Lost: Data == 0b11111111 && Position == 0
9. At the treasure's position: Data == 0b00000000 && Position == 333

   Theoretically, when the robot stayed at the center position, it should go forward until the next instruction came. When the robot was not at the center, it should fix its path back to the center.

The second part of the lab was to test the bump interruption. Theoretically, when any of the bumps were activated, meaning there was a wall exist, the robot should back up and search for a new path.

The third part of the lab was to set the same PWM on both motors when the robot ran, but the time delay is different for each of the 9 cases that should be different. This design was to easily control the robot.

Combine 3 parts above, we have flow chart as Figure 1.

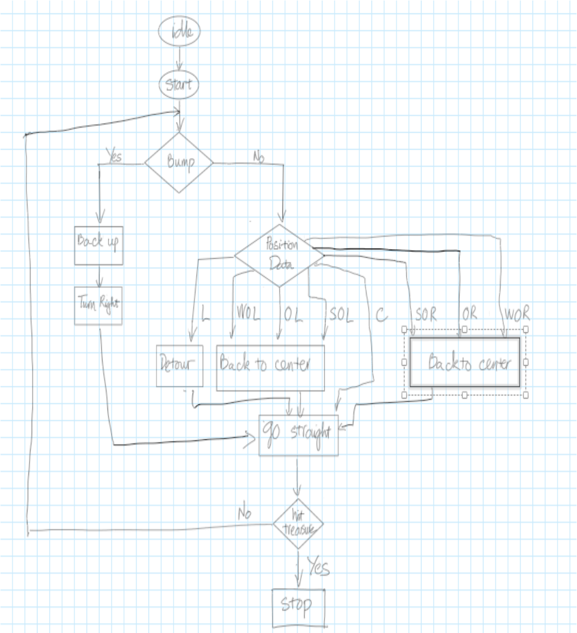

Figure 1: Flow chart of design.

IV. Testing Result

The demonstration is performed on the platform as figure 2.

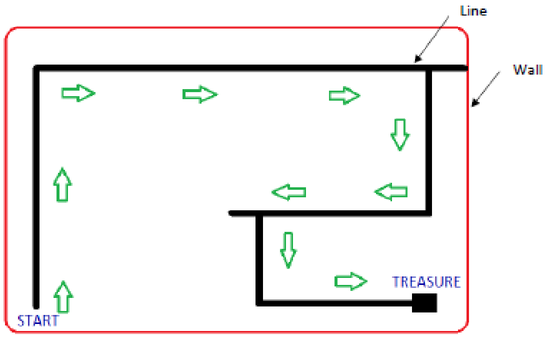

Figure 2: Robot explorer path to the treasure.

The successfully rate after 20 tests is 95%. During the failed test, the robot was encountered a situation where the sensors array was perpendicular to the track (Figure 3), so the robot thought that it hit the treasure so it stopped. This is a room for improvement.

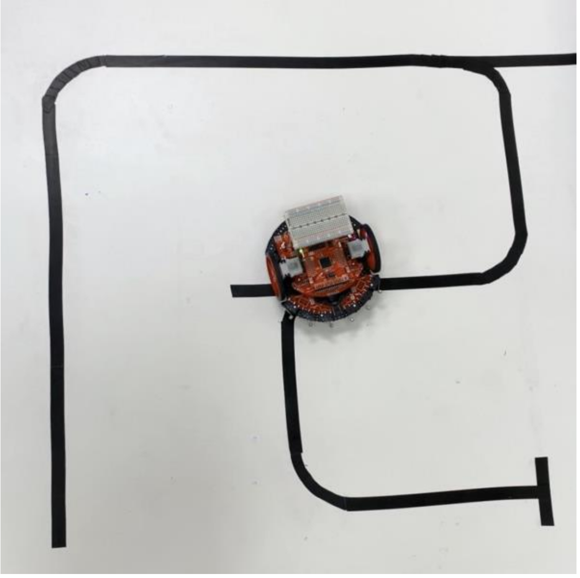

Figure 3: The sensors array is perpendicular to the track.

V. Challenge Encountered

There were two major challenges existed during the designing process and testing processing: reading line sensor error and design robot path with different time delay and PWM motors.

The first challenge was reading the line sensors error. Before the lab started, the line sensors worked perfectly, they captured data immediately. However, after calling this function in module 13, all sensors stopped working. Later, all sensors can finally capture data, but the responding time was too slow. When the robot was testing on the board, the line sensors stopped working again. We assumed that the problem came from our coding design, but after we changed to the new batteries, the sensors were working fine again.

The second challenge was to design the robot to take a detour when it is on the white space. The solution for this was to set the condition as Data = 0b1111 1111, Position = 0 in the if statement in if-else statement set. If we do not do this, the robot will keep running straight instead of take detour as Figure 4.

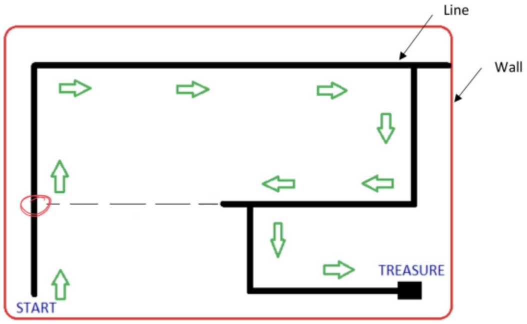

Figure 4: Wrong Path Detecting.

VI. Conclusion & Improvements

The project is very fun and helpful. The robot can run on the given platform with 95% successful rate. All the challenges have been solved.

The project provided a chance to practice:

- Periodic SysTick interrupts to measure the line sensor.
- Periodic Timer A1 interrupts to run the high-level strategy.
- Edge triggered interrupts for collisions.
- Main program for debugging and low priority tasks.

Beside that, the robot all so need two improvements:

- Try to reach 99.99% successful rate. Figure has shown the failed case. We should add another array of sensor (Figure 5), so the robot can distinguish the failed case and the treasure. Failed case: QTR-8RC is seeing the black line while QTR-2RC is seeing the white space, then the robot keep running forward. Treasure case, both QTR-8RC and QTR-2RC are seeing black line, the robot stops. By doing this, we can eliminate the encountered failed case.

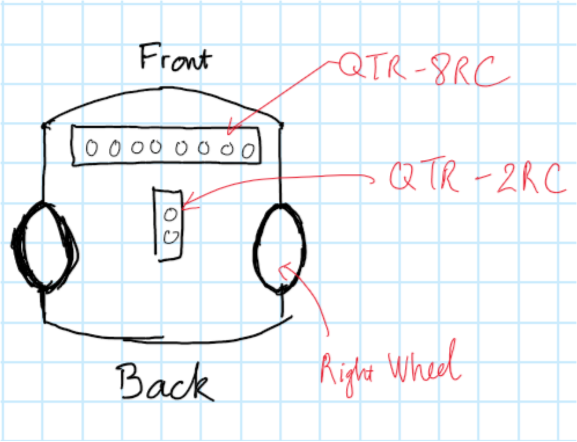

Figure 5: Robot after adding QTR-2RC sensor.

- Currently, the average time to reach the treasure is 60 seconds. Therefore, another key improvement will be speed up the robot by modifying the algorithm and delay time, so it can finish the task faster than the present version.

There were many challenges and problems existed during lab time, but during the demonstration, the robot passed all the tests within the first trial. The robot followed the line onboard correctly and stopped on the treasure spot perfectly.

VII. Appendices and References

Header files:

```
#include "msp.h"
#include "..\inc\bump.h"
#include "..\inc\Clock.h"
#include "..\inc\SysTick.h"
#include "..\inc\CortexM.h"
#include "..\inc\LaunchPad.h"
#include "..\inc\Motor.h"
#include "..\inc\TimerA1.h"
#include "..\inc\TExaS.h"
#include "..\inc\Reflectance.h"
```

Driver Module:

```
void TimedPause(uint32_t time){
  Clock_Delay1ms(time);          // run for
a while and stop
  Motor_Stop();
  while(LaunchPad_Input()==0);  // wait for
touch
  while(LaunchPad_Input());     // wait for
release
}

uint8_t Data; // QTR-8RC
int32_t Position; // 332 is right, and -332
is left of center

int main(void){
    TExaS_Init(LOGICANALYZER_P2);
    Clock_Init48MHz();
    LaunchPad_Init(); // built-in switches
and LEDs
    TimerA1_Init(&bump_interrupt,50000);  //
10 Hz
    EnableInterrupts();
    Bump_Init();      // bump switches
    Motor_Init();     // your function
    Reflectance_Init();

    int speed = 1500, backup_speed = 4000;
    int time1 = 50,
        time2 = 100,
        time3 = 150,
        time_backup = 300;

    TimedPause(1000);

    while(1){
        Data = Reflectance_Read(1000);
        Position =
Reflectance_Position(Data);
        Clock_Delay1ms(10);

        if (Data == 0b11111111 && Position
== 0) { // 2nd t-join
            Motor_Right(2000,2000);
            Clock_Delay1ms(1500);
            Motor_Stop();
            Motor_Forward(4000,4000);
            Clock_Delay1ms(500);
            Motor_Stop();
        }
        else if (Position >-47 && Position
<47) { //center
            Motor_Forward(speed,speed);
            Clock_Delay1ms(time3);
            Motor_Stop();
            //break;
        } else if (Position <= -47 &&
Position > -142) { //slightly off to the
left
            Motor_Left(speed,speed);
            Clock_Delay1ms(time1);
            Motor_Stop();
        } else if (Position >= 47 &&
Position <142) { //slightly off to the right
            Motor_Right(speed,speed);
            Clock_Delay1ms(time1);
            Motor_Stop();
        } else if (Position <= -142 &&
Position >-237) { //off to the left
            Motor_Left(speed,speed);
            Clock_Delay1ms(time2);
            Motor_Stop();
        } else if (Position >= 142 &&
Position < 237) { // off to the right
            Motor_Right(speed,speed);
            Clock_Delay1ms(time2);
            Motor_Stop();
        } else if (Position <= -237 &&
Position > -332) { // way off left
            Motor_Left(speed,speed);
            Clock_Delay1ms(time3);
            Motor_Stop();
        } else if (Position >= 237 &&
Position < 332) { // way off right
            Motor_Right(speed,speed);
            Clock_Delay1ms(time3);
            Motor_Stop();
        } else if (Data == 0b00000000 &&
Position == 333) { // goal
            Motor_Stop();
            //break;
        } else {

Motor_Backward(backup_speed,backup_speed);
            Clock_Delay1ms(time_backup);
            Motor_Stop();
        }
    }
}
```

Bump Module:

```
#include <stdint.h>
#include "msp.h"
void Bump_Init(void){
        P4->SEL0 &= ~0xED;
        P4->SEL1 &= ~0xED;
        P4->DIR &= ~0xED;
        P4->REN |= 0xED;
        P4->OUT |= 0xED;
}
    uint8_t result =0;

    result = ~P4->IN & 0xED;
    return result;
}
```

Motor Module:

```
#include <stdint.h>
#include "msp.h"
#include "../inc/CortexM.h"
#include "../inc/PWM.h"
```

```
// ------------Motor_Init------------
void Motor_Init(void){

    P3 -> SEL0 &= ~0xC0;
    P3 -> SEL1 &= ~0xC0;
    P3 -> DIR |= 0xC0;

    P5 -> SEL0 &= ~0x30;
    P5 -> SEL1 &= ~0x30;
    P5 -> DIR |= 0x30;

    PWM_Init34(15000, 0, 0);
    P3 -> OUT &= ~0xC0;

    return;
}

// ------------Motor_Stop------------
    P5->OUT &= ~0x30;
    P2->OUT &= ~0xC0;   // off
    P3->OUT |= 0xC0;   // low current sleep
mode
    PWM_Init34(15000, 0, 0);
    return;
}

// ------------Motor_Forward------------
    P3 -> OUT |= 0xC0;  // nSleep = 1
    P5 -> OUT &= ~0x30; // PH = 0
    PWM_Duty3(rightDuty);
    PWM_Duty4(leftDuty);
    return;
}

// ------------Motor_Right------------
void Motor_Right(uint16_t leftDuty, uint16_t
rightDuty){
    P3 -> OUT |= 0xC0;  // nSleep = 1
    P5 -> OUT &= ~0x10; // P5.4 PH = 0
    P5 -> OUT |= 0x20; // P5.5 PH = 1
    PWM_Duty4(leftDuty);
    PWM_Duty3(rightDuty);
}

// ------------Motor_Left------------
void Motor_Left(uint16_t leftDuty, uint16_t
rightDuty){
    P3 -> OUT |= 0xC0;  // nSleep = 1
    P5 -> OUT |= 0x10; // P5.4 PH = 1
    P5 -> OUT &= ~0x20; // P5.5 PH = 0
    PWM_Duty4(leftDuty);
    PWM_Duty3(rightDuty);
}

// ------------Motor_Backward------------
void Motor_Backward(uint16_t leftDuty,
uint16_t rightDuty){
    P3 -> OUT |= 0xC0;  // nSleep = 1
    P5 -> OUT |= 0x30; // PH = 1
    PWM_Duty3(rightDuty);
    PWM_Duty4(leftDuty);
    return;

}
```

PWM module:

```
#include "msp.h"

void PWM_Init1(uint16_t period, uint16_t
duty){
  if(duty >= period) return;     // bad
input
  P2->DIR |= 0x10;               // P2.4
output
  P2->SEL0 |= 0x10;              // P2.4
Timer0A functions
  P2->SEL1 &= ~0x10;             // P2.4
Timer0A functions
  TIMER_A0->CCTL[0] = 0x0080;    // CCI0
toggle
  TIMER_A0->CCR[0] = period;     // Period
is 2*period*8*83.33ns is 1.333*period
  TIMER_A0->EX0 = 0x0000;        //
divide by 1
  TIMER_A0->CCTL[1] = 0x0040;    // CCR1
toggle/reset
  TIMER_A0->CCR[1] = duty;       // CCR1
duty cycle is duty1/period
  TIMER_A0->CTL = 0x0230;        //
SMCLK=12MHz, divide by 1, up-down mode
}
void PWM_Init12(uint16_t period, uint16_t
duty1, uint16_t duty2){
  if(duty1 >= period) return; // bad input
  if(duty2 >= period) return; // bad input
  P2->DIR |= 0x30;          // P2.4, P2.5
output
  P2->SEL0 |= 0x30;         // P2.4, P2.5
Timer0A functions
  P2->SEL1 &= ~0x30;        // P2.4, P2.5
Timer0A functions
  TIMER_A0->CCTL[0] = 0x0080;      // CCI0
toggle
  TIMER_A0->CCR[0] = period;       // Period
is 2*period*8*83.33ns is 1.333*period
  TIMER_A0->EX0 = 0x0000;        //
divide by 1
  TIMER_A0->CCTL[1] = 0x0040;      // CCR1
toggle/reset
  TIMER_A0->CCR[1] = duty1;        // CCR1
duty cycle is duty1/period
  TIMER_A0->CCTL[2] = 0x0040;      // CCR2
toggle/reset
  TIMER_A0->CCR[2] = duty2;        // CCR2
duty cycle is duty2/period
  TIMER_A0->CTL = 0x02F0;        //
SMCLK=12MHz, divide by 8, up-down mode
}

void PWM_Duty1(uint16_t duty1){
  if(duty1 >= TIMER_A0->CCR[0]) return; //
bad input
```

```
  TIMER_A0->CCR[1] = duty1;        // CCR1
duty cycle is duty1/period
}

void PWM_Duty2(uint16_t duty2){
  if(duty2 >= TIMER_A0->CCR[0]) return; //
bad input
  TIMER_A0->CCR[2] = duty2;        // CCR2
duty cycle is duty2/period
}

void PWM_Init34(uint16_t period, uint16_t
duty3, uint16_t duty4){
    if(duty3 >= period) return; // bad input
    if(duty4 >= period) return; // bad input
    P2->DIR |= 0xC0;          // P2.6, P2.7
output
    P2->SEL0 |= 0xC0;         // P2.6, P2.7
Timer0A functions
    P2->SEL1 &= ~0xC0;        // P2.6, P2.7
Timer0A functions
    TIMER_A0->CCTL[0] = 0x0080;      // CCI0
toggle
    TIMER_A0->CCR[0] = period;       //
Period is 2*period*8*83.33ns is 1.333*period
    TIMER_A0->EX0 = 0x0000;        //
divide by 1
    TIMER_A0->CCTL[3] = 0x0040;      // CCR3
toggle/reset
    TIMER_A0->CCR[3] = duty3;        // CCR3
duty cycle is duty3/period
    TIMER_A0->CCTL[4] = 0x0040;      // CCR4
toggle/reset
    TIMER_A0->CCR[4] = duty4;        // CCR4
duty cycle is duty4/period
    TIMER_A0->CTL = 0x02F0;        //
SMCLK=12MHz, divide by 8, up-down mode
}

void PWM_Duty3(uint16_t duty3){
    if(duty3 >= TIMER_A0->CCR[0]) return; //
bad input
    TIMER_A0->CCR[3] = duty3;        // CCR3
duty cycle is duty3/period
}

void PWM_Duty4(uint16_t duty4){
    if(duty4 >= TIMER_A0->CCR[0]) return; //
bad input
    TIMER_A0->CCR[4] = duty4;        // CCR4
duty cycle is duty4/period
}
```

Reflectance Module:

```
#include <stdint.h>
#include "msp432.h"
#include "..\inc\Clock.h"

void Reflectance_Init(void){
    // write this as part of Lab 6
    //Clock_Init48MHz();

    P5->SEL0 &= ~0x08; //P5.3 set to GPIO
    P5->SEL1 &= ~0x08; //P5.3 set to GPIO
    P5->DIR |= 0x08; //P5.3 set to output
    P5->OUT &= ~0x08; //P5.3 set to low

    P9->SEL0 &= ~0x04;
    P9->SEL1 &= ~0x04;
    P9->DIR |= 0x04;
    P9->OUT &= ~0x04;

    P7->SEL0 &= ~0xFF; //P7.0 set to GPIO
    P7->SEL1 &= ~0xFF; //P7.0 set to GPIO
    //P7->DIR &= ~0xFF; //P7.0 set to input

}

uint8_t Reflectance_Read(uint32_t time){

    uint8_t result;
    //P5->SEL0 &= ~0x08;
    //P5->SEL1 &= ~0x08;
    //P5->DIR |= 0x08; //set 5.3 as output
    P5->OUT |= 0x08; //set 5.3 as high
    //P9->DIR |= 0x04; //set 9.2 as output
    P9->OUT |= 0x04; //set 9.2 as high

    //P7->SEL0 &= ~0xFF;
    //P7->SEL1 &= ~0xFF;
    //Clock_Delay1us(10);
    P7->DIR |= 0xFF; //set P7.7-P7.0 as
output
    P7->OUT |= 0xFF; //set P7.7-P7.0 as high
    Clock_Delay1us(10);

    P7->DIR = 0x00; //set P7.7-P7.0 as input
    Clock_Delay1us(time);

    result = (P7->IN & 0xFF);
    P5->OUT &= ~0x08; //turn off IR LED
    P9->OUT &= ~0x04; //turn off IR LED
    return ~result;
}

int32_t Reflectance_Position(uint8_t data){

    int W[8] = {-332, -237, -142, -47, 47,
142, 237, 332};
    uint8_t Mask[8] = {1, 2, 4, 8, 16, 32,
64, 128};

    int sum, count, result;
    sum = 0;
    count = 0;
    result = 0;
    for(int i = 0; i < 8; i++){
        if(data & Mask[i] ){
            count++;
            sum = sum + W[i];
        }

    }
    if(data == 0){
        return 333;
    }
    else{
        result = sum / count;
        return result;
    }
}
```

TimerA1 Module:

```
#include <stdint.h>
#include "msp.h"

void (*TimerA1Task)(void);   // user
function
void TimerA1_Init(void(*task)(void),
uint16_t period){
    TimerA1Task = task;             // user
function
    TIMER_A1->CTL &= ~0x0030;       // halt
Timer A1
    // bits15-10=XXXXXX, reserved
    // bits9-8=10,       clock source to
SMCLK
    // bits7-6=10,       input clock divider
/4
    // bits5-4=00,       stop mode
    // bit3=X,           reserved
    // bit2=0,           set this bit to
clear
    // bit1=0,           no interrupt on
timer
    TIMER_A1->CTL = 0x0280;
    // bits15-14=00,     no capture mode
    // bits13-12=XX,     capture/compare
input select
    // bit11=X,          synchronize capture
source
    // bit10=X,          synchronized
capture/compare input
    // bit9=X,           reserved
    // bit8=0,           compare mode
    // bits7-5=XXX,      output mode
    // bit4=1,           enable
capture/compare interrupt on CCIFG
    // bit3=X,           read
capture/compare input from here
    // bit2=0,           output this value
in output mode 0
    // bit1=X,           capture overflow
status
    // bit0=0,           clear
capture/compare interrupt pending
    TIMER_A1->CCTL[0] = 0x0010;
    TIMER_A1->CCR[0] = (period - 1);   //
compare match value
    TIMER_A1->EX0 = 0x0005;    // configure
for input clock divider /6
  // interrupts enabled in the main program
after all devices initialized
    NVIC->IP[2] =
(NVIC->IP[2]&0xFF00FFFF)|0x00400000; //
priority
    NVIC->ISER[0] = 0x00000400;   // enable
interrupt 10 in NVIC
    TIMER_A1->CTL |= 0x0014;      // reset
and start Timer A1 in up mode
}


// ------------TimerA1_Stop------------
// Deactivate the interrupt running a user
task periodically.
// Input: none
// Output: none
void TimerA1_Stop(void){
 // write this as part of Lab 13
    TIMER_A1->CTL &= ~0x0030;       // halt
Timer A1
    NVIC->ICER[0] = 0x00000400;     //
disable interrupt 10 in NVIC
}


void TA1_0_IRQHandler(void){
 // write this as part of Lab 13
    TIMER_A1->CCTL[0] &= ~0x0001; //
acknowledge capture/compare interrupt 0
    (*TimerA1Task)();             // execute
user task
}
```

Bump Interrupt Module:

```
void bump_interrupt(void) {
    uint8_t bumpResult = Bump_Read();
    if (bumpResult != 0x00) {

        int backup_speed = 2500, backup_time
= 300;

Motor_Backward(backup_speed,backup_speed);
        Clock_Delay1ms(backup_time);
        Motor_Stop();

        int turn_speed = 3000, turn_time =
900;
        Motor_Right(turn_speed,turn_speed);
        Clock_Delay1ms(turn_time);
        Motor_Stop();

        Motor_Forward(4000,4000);
        Clock_Delay1ms(500);
        Motor_Stop();

    }
    return;
}
```