Joseph Maloba
Dr. Shruti Singh
ECEGR 3210: Embedded Systems
March 20, 2025

# Project Report: Line-Following Robot Using MSP432P401R

This project was all about designing and building a line-following robot using the MSP432P401R micro-controller. The objective was to have the robot, with infrared (IR) sensors on either side and bumper switches right at the front of its body, follow a black line on a white surface. At first, it sounded quite simple but actually turned into the biggest troubleshooting headache I have ever seen.

The basic idea behind the design was based on differential steering, where two motors drive the wheels independently. And to use eight IR sensors to detect the position of the black line. These sensors work by emitting infrared light and detecting how much reflects back—black absorbs light while white reflects it. I used weighted sensor values to determine whether the robot was centered on the line or if it needed to adjust its direction. If the sensors on the right detected the line, the robot needed to turn left, if the sensors on the left among these detected the line, it had to turn right. The core of my code continuously read the sensor values and processed them using a weighted sum approach. Based on these values, it sent commands to the motors through GPIO pins P2.6 and P2.7. This allowed the robot to move forward, turn left or right, and stop when necessary.

While coding the logic was one part of the challenge, troubleshooting the sensors nearly drove me crazy. My robot refused to follow the line correctly, no matter how many times I adjusted the delays, re-calibrated the sensors, and rechecked my logic. I spent sleepless nights staring at my code, tweaking delays, adjusting sensor values, adjusting microsecond delays for sensor readings, and wondering why nothing seemed to work. It was frustrating. I thought that my sensors had to be bad.

After trying different delay times, testing my motor code separately, and even replacing my batteries, I finally picked up my robot and flipped it over. That’s when I saw it—my IR sensors

were mounted completely upside down! I spent days testing different fixes, getting more frustrated each time, until I realized the problem was so simple. I couldn't believe it. And the worst part? I had checked everything but the physical location of my sensors. Once they were turned correctly, immediately the readings made sense again but now I only had one motor working, so I had to replace my batteries and rewrite my code. Then, finally, my robot started following the line exactly as expected. Unbelievable! ─I had made a stupid mistake that took me so long to find.

After the sensors were on track, I adjusted the flexibility of turning logic and speed of robot so that movements became smoother. P4 has also been fitted with limitation-switches. I also added limit switches to stop the robot if it hit an obstacle. After days of frustration, finally watching my robot follow the line perfectly was the best feeling. It was one of those moments where I realized all the late nights actually paid off.

After this experience, I know one thing for sure—always check your hardware before blaming the code!

CCS Code :

Explain your code

```
#include "msp.h"

#define IREVEN BIT3 // IR sensor even
#define IRODD BIT2  // IR sensor odd
#define BUMP_SWITCH(d,p) !((d>>p)&0x01) // Bumper switch macro, checks a specific switch state

uint32_t i;
int Sensor0, Sensor1, Sensor2, Sensor3, Sensor4, Sensor5, Sensor6, Sensor7;
int SensorRight, SensorLeft, SensorTotal;
int del;
uint8_t bump_data, bump_data0, bump_data1, bump_data2, bump_data3, bump_data4, bump_data5;


void Bump_Init(void) {          //Configures GPIO pins for input with pull-up resistors
   P4->DIR &= ~(BIT7 | BIT6 | BIT5 | BIT3 | BIT2 | BIT0);
   P4->REN |= (BIT7 | BIT6 | BIT5 | BIT3 | BIT2 | BIT0);
   P4->OUT |= (BIT7 | BIT6 | BIT5 | BIT3 | BIT2 | BIT0);
}

uint8_t Bump_Read(void) { // Read the current state of 6 bumper switches (6-bit result return)
   return (~P4->IN) & (BIT7 | BIT6 | BIT5 | BIT3 | BIT2 | BIT0);
}

void SysTick_Init(void) { // Initialize SysTick
   SysTick->CTRL = 0;
   SysTick->LOAD = 0x00FFFFFF;
   SysTick->VAL = 0;
   SysTick->CTRL = 0x00000005;
}
```

```
void SysTick_Wait(uint32_t delay) { // SysTick wait module
   SysTick->LOAD = delay - 1;
   SysTick->VAL = 0;
   while ((SysTick->CTRL & 0x00010000) == 0);
}


void SysTick_Wait10ms(uint32_t delay) { // Wait for 10ms
   for (i = 0; i < delay; i++) {
      SysTick_Wait(480000);
   }
}

void SysTick_Wait1us(uint32_t delay) { // Wait for 1us
   for (i = 0; i < delay; i++) {
      SysTick_Wait(48);
   }
}

void Motor_Init(void) {
   P1->DIR |= (BIT6 | BIT7);       // Set P1.6 and P1.7 as output
   P1->OUT |= (BIT6 | BIT7);       // Initialize P1.6 and P1.7 to high

   P2->DIR |= (BIT6 | BIT7);       // Set P2.6 and P2.7 as output
   P2->OUT &= ~(BIT6 | BIT7);      // Initialize P2.6 and P2.7 to low

   P3->DIR |= (BIT6 | BIT7);       // Set P3.6 and P3.7 as output
   P3->OUT |= (BIT6 | BIT7);       // Initialize P3.6 and P3.7 to high
}

void Motor_Straight(void) {
   for (del = 0; del < 50; del++) {
      P2->OUT |= (BIT6 | BIT7);   // Turn on both motor pins
      SysTick_Wait1us(1);         // Short delay
      P2->OUT &= ~(BIT6 | BIT7); // Turn off both motor pins
      SysTick_Wait1us(3);         // Longer delay
   }
}

void Motor_TurnLeft(void) {
   for (del = 0; del < 25; del++) {
      P2->OUT |= BIT6;            // Turn on left motor pin
      P2->OUT &= ~BIT7;           // Turn off right motor pin
      SysTick_Wait1us(1);         // Short delay
      P2->OUT &= ~(BIT6 | BIT7); // Turn off both motor pins
      SysTick_Wait1us(3);     // Longer delay
   }
}

void Motor_TurnRight(void) {
   for (del = 0; del < 25; del++) {
      P2->OUT &= ~BIT6;           // Turn off left motor pin
      P2->OUT |= BIT7;            // Turn on right motor pin
      SysTick_Wait1us(1);         // Short delay
      P2->OUT &= ~(BIT6 | BIT7); // Turn off both motor pins
      SysTick_Wait1us(3);         // Longer delay
   }
}

void Motor_Stop(void) {
   for (del = 0; del < 100; del++) {
```

```
        P2->OUT &= ~(BIT6 | BIT7);  // Ensure both motor pins are off
        SysTick_Wait1us(1000);  // Delay to keep motors stopped
    }
}

int main(void) {
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;     // Stop watchdog timer
    SysTick_Init();
    Motor_Init();
    Bump_Init();

    P5->DIR |= IREVEN;
    P9->DIR |= IRODD;

    while (1) {                   // Set IR sensor on
        P5->OUT |= IREVEN;
        P9->OUT |= IRODD;
        P7->DIR |= 0xFF;        // Set sensors as high / charge capacitors
        P7->OUT |= 0xFF;
        SysTick_Wait1us(5);
        P7->DIR &= ~0xFF;           // Set sensors as inputs
        SysTick_Wait1us(50);


        Sensor0 = P7->IN & BIT0 ? -3 : 0;
        Sensor1 = P7->IN & BIT1 ? -2 : 0;
        Sensor2 = P7->IN & BIT2 ? -1 : 0;
        Sensor3 = P7->IN & BIT3 ? 0 : 0;                // Read sensor values
        Sensor4 = P7->IN & BIT4 ? 0 : 0;
        Sensor5 = P7->IN & BIT5 ? -1 : 0;
        Sensor6 = P7->IN & BIT6 ? -2 : 0;
        Sensor7 = P7->IN & BIT7 ? -3 : 0;

        // Compute weighted sums for left and right sensors
        SensorRight = Sensor0 + Sensor1 + Sensor2 + Sensor3;
        SensorLeft = Sensor4 + Sensor5 + Sensor6 + Sensor7;

        bump_data = Bump_Read();        // Read the bump switches state

        bump_data0 = BUMP_SWITCH(bump_data, 0);
        bump_data1 = BUMP_SWITCH(bump_data, 1);
        bump_data2 = BUMP_SWITCH(bump_data, 2);         // Check bumper switches
        bump_data3 = BUMP_SWITCH(bump_data, 3);
        bump_data4 = BUMP_SWITCH(bump_data, 4);
        bump_data5 = BUMP_SWITCH(bump_data, 5);

        P5->OUT &= ~IREVEN;         // Turn off IR LEDs
        P9->OUT &= ~IRODD;

        SysTick_Wait1us(10);

        if (Sensor0 == -3 && Sensor1 == -2 && Sensor2 == -1 && Sensor3 == 0 &&
            Sensor4 == 0 && Sensor5 == -1 && Sensor6 == -2 && Sensor7 == -3) {
                Motor_Stop();
                SysTick_Wait1us(1000);
            } else if (SensorRight != 0 && SensorLeft == 0) {
                Motor_TurnRight();
            } else if (SensorRight == 0 && SensorLeft != 0) {
                Motor_TurnLeft();
            } else {
                Motor_Straight();
                SysTick_Wait1us(1);
```

```
        }
    }
}
```