# Design and Implementation of a Line-Following Robot - MSP432P401R

Jeremy Kawahigashi
Embedded Systems
3/21/25

---

## Abstract

This report details the design, implementation, and evaluation of a line-following robot built using the MSP432 Launchpad. The robot employs infrared (IR) sensors to detect a dark line on a lighter surface and maneuvers accordingly by controlling two sets of motors. The project successfully achieved its primary goal of reliably following a curved line and executes motor functions for forward motion, left and right turns, and halting. The report also discusses potential improvements, such as incorporating additional timers to enable pulse-width modulation (PWM) for finer control over motor speed and direction.

---

## Introduction

Line-following robots are a classic application in embedded systems, combining sensor input, control algorithms, and actuator responses. This project, part of the ECEGR 3210 Embedded Systems course at Seattle University, demonstrates the integration of IR sensors with a microcontroller to create a robot capable of autonomously following a path. The MSP432 Launchpad was chosen due to its powerful capabilities and suitability for real-time motor control. Key functionalities include:

- **Line Detection:** Using an array of IR sensors to differentiate between a dark line and a light background.
- **Motor Control:** Executing predefined motion commands (forward, turn left, turn right, and stop) based on sensor input.
- **Obstacle Sensing (Optional):** Incorporating bumper switches and the potential integration of a time-of-flight sensor for future expansion.

## Materials and Methods

### Hardware Components

- **MSP432 Launchpad:** The main microcontroller for processing sensor data and controlling motors.
- **IR Sensors:** Arranged in an array to detect a dark line on a white surface. Each sensor outputs analog signals based on the reflection of IR light.
- **Motors and Motor Drivers:** Two sets of motors (left and right) are used to drive the robot. Motor drivers control the direction and enable/disable motor operation.
- **Bumper Switches:** Six switches connected to GPIO pins for detecting physical obstructions.
- **Optional Sensor Module:** Pololu 3-channel wide FOV Time-of-Flight Distance sensor (for future integration to detect obstacles).

### Software Implementation

The code is written in C using the TI DriverLib. Below is a high-level overview of the software architecture:

#### Flowchart Overview

```
A[Start: Initialize Hardware (Watchdog Timer, SysTick, Motors, Bumpers, IR LEDs)]
B[Enter Main Loop]
C[Activate IR LEDs]
D[Charge IR Sensor Capacitors]
E[Set Sensors to Input and Wait]
F[Read Sensor Values]
G[Compute Weighted Sums for Left and Right]
H[Read Bumper Switch States]
I[Deactivate IR LEDs]
J{All Sensors on Line?}
K[Stop Motors]
```

```
L{Tape Detected on Right?}
M[Turn Right]
N{Tape Detected on Left?}
O[Turn Left]
P[Move Straight]
Q[Loop Back to Main Loop]

A --> B
B --> C
C --> D
D --> E
E --> F
F --> G
G --> H
H --> I
I --> J
J -- Yes --> K
J -- No --> L
L -- Yes --> M
L -- No --> N
N -- Yes --> O
N -- No --> P
K --> Q
M --> Q
O --> Q
P --> Q
```

*Figure 1: Flowchart depicting the overall logic of the line-following algorithm.*

## Motor Control Implementation

The IR sensor and overall tracking of the line following are implemented using weighted values, as shown in Figure 2. By allowing weighted values, the system determines deviations from the line and corrects the movement accordingly. The motor functions utilize a for loop and a SysTick timer. The for loop, combined with SysTick delays, mimics a pulse-width modulator (PWM) by maintaining a constant frequency. This effectively allows the motor to run in small pulses, making the overall drive smoother and more controlled.

## Sensor Weighting and Decision Making

Below is an excerpt from the main control loop that governs the robot's motion based on sensor readings:

```
Sensor0 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN0) ? -3 : 0;

Sensor1 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN1) ? -2 : 0;

Sensor2 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN2) ? -1 : 0;

Sensor3 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN3) ? 0 : 0;

Sensor4 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN4) ? 0 : 0;

Sensor5 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN5) ? -1 : 0;

Sensor6 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN6) ? -2 : 0;

Sensor7 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN7) ? -3 : 0;


// Compute weighted sums for left and right sensors

SensorRight = Sensor0 + Sensor1 + Sensor2 + Sensor3;

SensorLeft = Sensor4 + Sensor5 + Sensor6 + Sensor7;
```

**Figure 2**: Pseudocode snippet showing the weighted values used to compute left and right sensor sums.

The motor functions use a PWM-like approach to maintain smooth movement. Below is the function to move the robot straight:

```
void Motor_Straight(void){
    for (del=0; del<50; del++) {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN6); // both motors on
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(1);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(3);
    }
}
```

**Figure 3**: Pseudocode snippet demonstrating motor straight.

## Decision-Making Based on Sensor Inputs

The following pseudocode snippet shows how the robot determines movement based on sensor readings:

```
if (/* All sensor values indicate presence of line */) {
    Motor_Stop();
    SysTick_Wait1us(1000);

} else if ((SensorRight != 0x00) && (SensorLeft == 0x00)) {

    Motor_TurnRight();

} else if ((SensorRight == 0x00) && (SensorLeft != 0x00)) {

    Motor_TurnLeft();

} else {

    Motor_Straight();

    SysTick_Wait1us(1);

}
```

**Figure 4**: Pseudocode snippet showing decision-making logic based on sensor inputs.

### Materials

- **Hardware:** MSP432 Launchpad, IR sensors, motor drivers, bumper switches.
- **Software:** C programming using TI DriverLib, SysTick for timing, GPIO for sensor and motor control.

---

## Results

The implemented line-following robot demonstrated reliable performance in following a curved line drawn on a contrasting surface. The key observations include:

- **Sensor Accuracy:** The IR sensors effectively distinguished between the dark line and the white surface.
- **Motor Functions:** The robot correctly executed forward motion, left and right turns, and stopped when the line was fully detected by all sensors.
- **Bumper Feedback:** Bumper switches correctly identified obstacles, enabling safe operation.
- **Response Time:** The use of SysTick allowed for precise timing in sensor readings and motor control, though it is noted that the system could benefit from additional timers for more refined PWM control.

---

## Discussion

The project successfully met its objectives. The robot reliably followed the designated path and the motor functions operated as expected. The design choices—using IR sensors for line detection and a simple threshold-based decision algorithm—proved effective for the task.

### Areas for Improvement

- **PWM Implementation:** Currently, motor speed is controlled by basic delays using SysTick. Incorporating additional timers to generate PWM signals would allow for more granular control over motor speed and direction. This could enhance the robot's ability to navigate more complex curves or adjust speed dynamically. For example instead of turning off one motor completely when turning, the motor could be slowed to initiate a slow speed, or reversed to increase speed.
- **Obstacle Avoidance:** While bumper switches provide basic collision detection, integrating the Pololu Time-of-Flight Distance sensor could significantly improve obstacle detection and avoidance.

---

## Conclusion

The line-following robot project successfully demonstrated the integration of sensor data, real-time processing, and motor control using the MSP432 Launchpad. By utilizing IR sensors to detect the line on the surface, the robot could follow a complex path, including straight lines, sharp turns, intersections, and complete stops when all sensors detect a black line. The data captured from the IR sensors was interpreted using weighted values to determine the robot's position relative to the line.

The robot effectively followed a lop-sided infinity sign and a maze created by Gary Fernandes, showing its ability to navigate a challenging course. While the current setup was functional, future improvements such as implementing PWM for more precise motor control and enhancing obstacle detection could further increase the robot's performance and adaptability in more complex environments.

---

## References

- – Provided by Texas Instruments, detailing the use of driver libraries for MSP432.
- ECEGR 3210 Course Materials, Seattle University.

---

## Code

**The bump sensors on the MSP432P401R were ONLY initalized and viewable for debugging. THERE IS NO IMPLEMENTATION USING THE BUMP SENSORS WHEN TURNING**

```
#include <ti/devices/msp432p4xx/driverlib/driverlib.h>
//Define everything needed
#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;
// Bumper initialization: Configures GPIO pins for input with pull-up resistors
void Bump_Init(void){
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN7);
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN6);
   MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN5);
```

```
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN3);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN2);
    MAP_GPIO_setAsInputPinWithPullUpResistor(GPIO_PORT_P4, GPIO_PIN0);
}
// Read the current state of 6 bumper switches (returns a 6-bit result)
// Bit 5 Bump5, Bit 4 Bump4, Bit 3 Bump3, Bit 2 Bump2, Bit 1 Bump1, Bit 0 Bump0
uint8_t Bump_Read(void){
    uint8_t result=0;
    result = (result << 1) | MAP_GPIO_getInputPinValue(GPIO_PORT_P4, GPIO_PIN7);
    result = (result << 1) | MAP_GPIO_getInputPinValue(GPIO_PORT_P4, GPIO_PIN6);
    result = (result << 1) | MAP_GPIO_getInputPinValue(GPIO_PORT_P4, GPIO_PIN5);
    result = (result << 1) | MAP_GPIO_getInputPinValue(GPIO_PORT_P4, GPIO_PIN3);
    result = (result << 1) | MAP_GPIO_getInputPinValue(GPIO_PORT_P4, GPIO_PIN2);
    result = (result << 1) | MAP_GPIO_getInputPinValue(GPIO_PORT_P4, GPIO_PIN0);
    return result;
}
//declare SysTick
void SysTick_Init(void){
    SysTick->CTRL = 0x00000005;
}
// SysTick wait module
void SysTick_Wait(uint32_t n){
    SysTick->LOAD = n-1;
    SysTick->VAL = 0;
    while((SysTick->CTRL&0x00010000)== 0){};
}
// Wait for 10ms
void SysTick_Wait10ms(uint32_t delaym){
    for (i=0; i<delaym; i++){
        SysTick_Wait(480000);
    }
}
```

```
// Wait for 1us
void SysTick_Wait1us(uint32_t delayu){
    for (i=0; i<delayu; i++){
        SysTick_Wait(48);
    }
}
void Motor_Init(void){
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN6); // enable motors
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN6);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P1, GPIO_PIN7);
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P1, GPIO_PIN7);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN6); //initialize PWM pins
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P2, GPIO_PIN7);
    MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN6); // puts drivers to sleep
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN6);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P3, GPIO_PIN7);
    MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P3, GPIO_PIN7);
}
void Motor_Straight(void){
    for (del=0;del<50;del++) {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN6); // both motors on
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(1);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(3);
    }
}
void Motor_TurnLeft(void){
    for (del=0;del<25;del++) {
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN6); // right motor on
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(1);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(3);
    }
}
void Motor_TurnRight(void){
    for (del=0;del<25;del++) {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P2, GPIO_PIN7); // left motor on
        SysTick_Wait1us(1);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(3);
    }
}
void Motor_Stop(void){
    for (del=0;del<100;del++) {
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN6); // both motors off
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P2, GPIO_PIN7);
        SysTick_Wait1us(1000);
    }
}
int main(void){
    WDT_A_holdTimer();      // stop watchdog timer
    SysTick_Init();
    Motor_Init();
    Bump_Init();
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P5, IREVEN);
    MAP_GPIO_setAsOutputPin(GPIO_PORT_P9, IRODD);
    while(1)
    {
        // set IR sensor on
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P5, IREVEN);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P9, IRODD);
        //set sensors as high / charge capacitors
        MAP_GPIO_setAsOutputPin(GPIO_PORT_P7,
GPIO_PIN0|GPIO_PIN1|GPIO_PIN2|GPIO_PIN3|GPIO_PIN4|GPIO_PIN5|GPIO_PIN6|GPIO_PIN7);
        MAP_GPIO_setOutputHighOnPin(GPIO_PORT_P7,
GPIO_PIN0|GPIO_PIN1|GPIO_PIN2|GPIO_PIN3|GPIO_PIN4|GPIO_PIN5|GPIO_PIN6|GPIO_PIN7);
        SysTick_Wait1us(5);
        //set sensors as inputs
        MAP_GPIO_setAsInputPin(GPIO_PORT_P7,
GPIO_PIN0|GPIO_PIN1|GPIO_PIN2|GPIO_PIN3|GPIO_PIN4|GPIO_PIN5|GPIO_PIN6|GPIO_PIN7);
        SysTick_Wait1us(50);
        //define Sensors to respond to IR input
        // Read sensor values
        Sensor0 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN0) ? -3 : 0;
        Sensor1 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN1) ? -2 : 0;
        Sensor2 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN2) ? -1 : 0;
        Sensor3 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN3) ? 0 : 0;
        Sensor4 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN4) ? 0 : 0;
        Sensor5 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN5) ? -1 : 0;
        Sensor6 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN6) ? -2 : 0;
        Sensor7 = MAP_GPIO_getInputPinValue(GPIO_PORT_P7, GPIO_PIN7) ? -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
        // Check bumper switches (assuming the bumper switches are mapped as Sensor0 through Sensor7)
        bump_data0 = BUMP_SWITCH(bump_data, 0);
        bump_data1 = BUMP_SWITCH(bump_data, 1);
        bump_data2 = BUMP_SWITCH(bump_data, 2);
        bump_data3 = BUMP_SWITCH(bump_data, 3);
        bump_data4 = BUMP_SWITCH(bump_data, 4);
        bump_data5 = BUMP_SWITCH(bump_data, 5);
        //5 4 3
        //turn off IR LEDS
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P5, IREVEN);
        MAP_GPIO_setOutputLowOnPin(GPIO_PORT_P9, IRODD);
        SysTick_Wait1us(10);
        if (((int)Sensor0 == -3) && ((int)Sensor1 == -2) && ((int)Sensor2 == -1) && ((int)Sensor3 == 0)
                && ((int)Sensor4 == 0) && ((int)Sensor5 == -1) && ((int)Sensor6 == -2) && ((int)Sensor7 == -3)) {
            Motor_Stop();
            SysTick_Wait1us(1000);
        } else if ((SensorRight != 0x00) && (SensorLeft == 0x00)) { //if tape detected on right, turn right
            Motor_TurnRight();
        } else if ((SensorRight == 0x00) && (SensorLeft != 0x00)) { //if tape detected on left, go left
            Motor_TurnLeft();
        } else {
            Motor_Straight();
            SysTick_Wait1us(1);
        }
    }
}
```