한양대학교
HANYANG UNIVERSITY

Spring 2024

EIS1015 Embedded System Design

Line Tracer Project Report

June 11th, 2024

Realized by:

AHROUM, ADNANE (9073520241)

NG, IAN ZHI YANG (9062920245)

Group 23

Contents:

I. Abstract:

This report is destined for the embedded systems class. It describes the implementation of a program to control a robotic vehicle to navigate in a precise path mentioned by the instructor. The system uses an MSP432 microcontroller. The slides started by showing a basic register control, which is setting the pins corresponding to pin numbers, which made us able to manipulate LED's

II. Introduction:

The task involves programming a robotic vehicle to follow a specified path marked on a surface. The robot uses the QTRX sensor, which consists of infrared (IR) sensors to detect lines and control its movement accordingly, such that the sensors entitled from 0 to 3 are dedicated for input use, and from 4 to 7 for outputs. The MSP432 microcontroller also possesses an emitter that transmits IR light which should be directed towards the phototransistor. This setup can be viewed using an IR camera. These features effectively manage the motor control, IR sensor readings, and overall timing functions. The main goals are to:

1) Start from a designated line.
2) Move forward while maintaining alignment with the path.
3) Make precise turns to navigate the polygonal course.
4) Reach the end line successfully.

III. Design and Algorithm:

a. Phase 1:

The first phase of the algorithm is designed to control the overall movement of the robot, allowing it to move forward, maintain alignment, and perform a rotation at a necessary point of time. This phase is crucial for navigating and memorizing the route, as it is composed of both the detection and reaction to specific conditions such as the starting line and the stopping points of the shape. The code execution of the algorithm can be divided into two main sections :

Section 1: Moving forward & Keeping the alignment:

- Starting Condition: execution begins when “Starting Line” is detected. Flag setting ‘start_line_flag’ to true.
- Charging Capacitor: preparing the system’s sensors or circuits for tasks.
- Reading Sensors:

```
int sensor;
sensor = P7->IN & 0xFF;
```

The sensor readings are captured from port ‘P7’, which is connected to QTRX sensors. The “& 0xFF” operation ensures that bits 0 to 7 are read.

- Moving Forward Sequence :

```
int Is_Running;
Is_Running = Moving_Straight_Sequence( sensor, forward_duty); // Will terminate when it detects nothing
```

"Moving_Straight_Sequence" function handles the robot's movement based on two parameters: the sensor's input adjust the path to keep a straight movement, and the "forward_duty" parameter that control's the motor's speed and power.

- Updating Movement Status:

```
if (Is_Running){ // checks if 0 or 1
    movement_array[array_counter] = "Forward";
    Clock_Delay1ms(140); //200

    stopped = 0;
} else {
    Stop();
    stopped = 1;
}
IR_off();
```

If the robot is moving, it updates the movement array to record the forward movement and sets a delay for 140 milliseconds. Else, the robot stops the stop flag is updated. The IR sensors are then turned off to conserve power.

Section 2: Rotate:

- Rotation condition:

```
            //---------- 1b. Rotate ----------//
if (rotating_count < 8){

if (stopped) {
    printf("rotating_count: %d\n", rotating_count);
```

The robot will rotate 7 times (8 vertices – first vertice). Rotation will commence if the robot stops moving and the sensors detect nothing but white.

- Checking Sensor Status:

```
                    //--------Check Sensor--------//
Clock_Delay1ms(500);
Charge_capacitor();
sensor = P7->IN & 0xFF;

int bms1 ;
bms1 = sensor & 0xFF;

if (bms1 == 0)
{                                       //Change if buggy
                    //--------Move back------//


                    //--------Rotate 135 degrees CW------//
```

After 500ms, the sensor values are re-read. The condition 'if (bms1 == 0)' checks whether the sensors detect no line, which indicates that the robot is in a position to rotate.

- Rotating the robot:

```
    printf("Rotating \n");
    printf("Array Counter : %d\n", array_counter);
    Rotate_CW(rotation_duty);
    Clock_Delay1ms(335); //490
    Stop();
    rotating_count += 1;
    array_counter += 1;
    movement_array[array_counter] = "Rotate CW 135";
    stopped = 0;


    printf("Turn Right!!");
    Turn_Right_New(); //
    Clock_Delay1ms(505);
    Stop();
    movement_array[array_counter] = "Move Forward (s)";
        }

    IR_off();

    }
}else {
```

When the robot detects no line, it performs a clockwise rotation of 135°, which will let it turn to continue through the path. The number of delays are very precise due to the amount of testing that we performed on the prototype. After rotation, it updates the array and the flag. Then, the robot performs an additional turn to the right with another delay to complete the adjustment of the orientation.

Section 3: Running the circumference of the polygon

- Phase transition:

```
// -------------------Phase 2: When rotation hits 7, start tracing the polygon-----------------
              printf("Phase 2, rotating_count: %d\n", rotating_count);
              Clock_Delay1ms(500);
                Charge_capacitor();
                sensor = P7->IN & 0xFF;
                int bmx1, bmx2;
                bmx1 = sensor & 0xFF;
                bmx2 = sensor & 0xFF;
```

We begin the phase when the “rotating_count” reaches 7, as we have eight vertices, but we wouldn’t count the one we start in. 500ms delay is inputted to allow the system’s stabilization before charging the capacitor and reading the sensor values again. This reset’s purpose is to ensure that the system possesses accurate data before we can move on with our phase.

Section 3.1: Conditional Rotation Actions:

- Rotate 90 Degrees (rotating_count == 8)

```
if (rotating_count == 8){
    // Moveback removed
     printf("Rotating CW 90 \n");
     Rotate_CW(rotation_duty);
     Clock_Delay1ms(40);
     array_counter += 1;
     movemefnt_array[array_counter] = "Rotate40";
     stopped = 0;
```

When the variable reaches 8, the robot turns 90-degree clockwise rotation, through ‘Rotate_CW’, with a delay of 40 ms. Recording in ‘movement_array’.

- Rotate 270 Degrees (rotate_count == 15)

```
} else if (rotating_count == 15){
    printf("Rotating 270 \n");
     Rotate_CW(rotation_duty);
     Clock_Delay1ms(670);
     array_counter += 1;
     movement_array[array_counter] = "Rotate670";
     stopped = 0;
     //Execute end sequence
     Stop();
     Clock_Delay1ms(30000);
     array_counter += 1;
     movement_array[array_counter] = "Stop10000";
     Execute_Track_Reversal(movement_array);
```

The delay is 670ms. After the rotation of 270 degrees, the robot stops for 30 seconds, this is so that we get enough time for attaching a pen and move in for the second phase. We will speak about the last function later.

- Rotate 45 degrees (Default Case): the delay is 55ms.

b. Phase 2:

This phase involves the execution of a stored movement array, containing the memorized path that the robot has to retrace. This is achieved by a function that split each command into a text and number part and then executes some specific actions according to commands.

Section 1: Split Function

- Structure definition :

```
typedef struct {
    char textPart[100];
    int numberPart;
} SplitResult;
```

- Function :

```c
//------------------------------ Function to split a string containing text and numbers
SplitResult SplitStringAndNumber(const char *input) {
    SplitResult result;
    int i = 0;
    int j = 0;
    int len = strlen(input);
    // Initialize the structure members
    result.numberPart = 0;
    memset(result.textPart, 0, sizeof(result.textPart));
    // Loop through the input string
    for (i = 0; i < len; i++) {
        // If the character is a digit, break the loop
        if (isdigit(input[i])) {
            break;
        }
        // Append the character to the text part
        result.textPart[j++] = input[i];
    }
    // Loop through the remaining part of the string to extract the number
    for (; i < len; i++) {
        if (isdigit(input[i])) {
            result.numberPart = (result.numberPart * 10) + (input[i] - '0');
        }
    }
    return result;
}
```

This function cycles through the string to extract the digits, which divides text in ‘TextPart’, and numeric components in ‘NumberPart’.

Section 2: Executing the movement commands:

- Main function:

```c
void Execute_Track_Reversal(const char **movement_array){
    // Call the function to split the string and number
    int i = 0;
    int array_size =10000;
    for (i=0; i < array_size; i++) {
            if (movement_array[i] != NULL) { // Check if the entry is not NULL
                SplitResult result = SplitStringAndNumber(movement_array[i]);
                // Print the results
                printf("Text: %s\n", result.textPart);
                printf("Number: %d\n", result.numberPart);
```

The function iterated through the ‘movement_array’ to process each entry and split.

- Command Execution:

```
            if (strcmp(result.textPart, "Forward") == 0) {
                printf("Moving Forward\n");
                Move_Forward_New(5000);
            } else if (strcmp(result.textPart, "Left") == 0) {
                printf("Moving Left\n");
                Turn_Left_New();
            } else if (strcmp(result.textPart, "Right") == 0) {
                printf("Moving Right\n");
                Turn_Right_New();
            } else if (strcmp(result.textPart, "Back") == 0) {
                printf("Moving Back\n");
                Move_Backward_New(5000);
            } else if (strcmp(result.textPart, "Rotate") == 0) {
                printf("Rotating\n");
                Clock_Delay1ms(500);
                Rotate_CW(10000);
            } else if (strcmp(result.textPart, "Stop") == 0){
                printf("Stopping\n");
                Stop();
                Clock_Delay1ms(10000); //Stop for 10 seconds
            }
            Clock_Delay1ms(result.numberPart);//time
        }
    }
```

For each command, the function compares the ‘TextPart’ to predefined movement types: (‘Forward", "Left", "Right", "Back", "Rotate", "Stop"), and executes the corresponding movement.

IV. Difference between expected and actual results:

Expected Results: The robot was designed to follow a specific line path by accurately reading and responding to inputs from sensors 1 through 6. This was supposed to ensure precise navigation along the intended route.

Actual Results: Instead of sticking to the designated line path, the robot occasionally read inputs from other lines, which introduced erratic data. This destabilized the system and caused the robot to deviate from its path.

Improvements for Next Time: To enhance the robot's performance, we should simplify the sensor setup by relying on only 2 sensors. Additionally, implementing PID (Proportional-Integral-Derivative) control will help regulate the robot's movements more effectively, ensuring smoother and more accurate line following.

V. Conclusion:

This report presented the implementation and operation of a robotic vehicle controlled by an MSP432 microcontroller to navigate a polygonal path. The code is structured to initialize necessary peripherals, control motor movement, read sensor data, and make navigation decisions. The robot successfully starts from a designated line, maintains path alignment, makes turns, and completes the navigation to an end line. The combination of hardware initialization, sensor data processing, and motor control functions ensures the robot performs the intended tasks accurately.