# Embedded System Design - Line Tracer

(Team 22)
2020011167 Lim Youbin
2020079689 Shin Dahye

## 1. Functions & Variables

(1) Macro constant

| | |
|---|---|
| `#define TURN_TIME_MS 500` | Rotate Time |
| `#define DOT_DETECT_DELAY 60000` | Delay to prevent duplicate dot detection |
| `#define nstartline 0`<br>`#define ndot 1`<br>`#define nline 2`<br>`#define nfinishline 3` | A constant representing the each status |

(2) Global variable

| | |
|---|---|
| `int count` | Indicate the current vertex number |
| `int status`<br>`int prev_status` | Store current status (nstartline \| ndot \| nline \| nfinishline)<br>Store previous status |
| `int dot_detect_counter` | Variable to prevent duplicate dot detection |
| `int graph[8][8]` | 2D array to store track information<br>graph[i][j] = -1, if (i and j are not connected)<br>graph[i][j] = 0~3, if (i and j are connected) |
| `int Euler_path[16][2]` | 2D array to store euler path<br>Eulerpath[path_num][0]<br>: the starting point of the path_num-th path.<br>Eulerpath[path_num][1]<br>: the edge number to take for the path_num-th path. |
| `int path_num` | An index for storing the Euler path |

(3) Function

- Initializing and Necessary action

| | |
|---|---|
| `void LED_Init(void)`<br>`void IR_Init(void)` | Initialize the LED and IR sensor |
| `void Move(uint16_t, uint16_t)` | Assign leftduty and rightduty for move |
| `void Left_Forward()`<br>`void Left_Backward()`<br>`void Right_Forward()`<br>`void Right_Backward()` | Assign direction values for each motor |

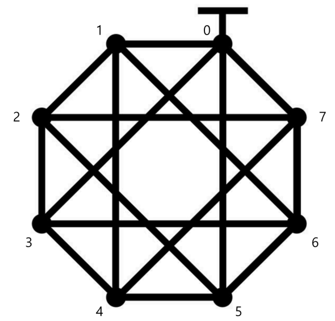
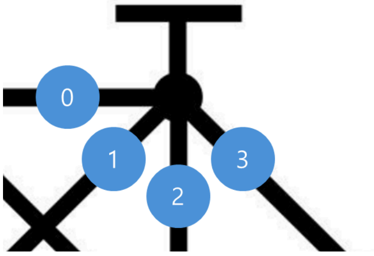
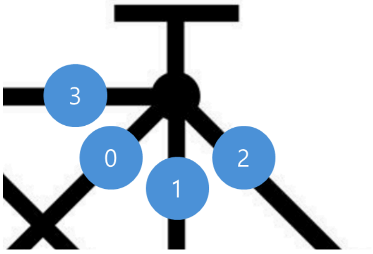
- Rotation

| | |
|---|---|
| `void Left_rotate_45_degrees()`<br>`void Right_rotate_45_degrees()`<br>`void Left_rotate_90_degrees()`<br>`void Right_rotate_90_degrees()`<br>`void Left_rotate_135_degrees()`<br>`void Right_rotate_135_degrees()`<br>`void Right_rotate_135_degrees_p2()` | Rotate to approximate angles. To minimize errors caused by hardware characteristics, functions used for rotation in phase 2 are distinguished by appending 'p2' to their names. |
| `void Rotate_Clock(int)`<br>`void Rotate_CounterClock(int)`<br>`void Rotate_CounterClock_p1(int)` | Rotates clockwise or counter clockwise and stops at the position of the target line passed as a parameter. Rotate_CounterClock_p1 is used in phase 1. |

- Specific action

| | |
|---|---|
| `void find()`<br>`void startline();`<br>`void dot();`<br>`void line();`<br>`void reverse_find();`<br>`void Back_line()` | Distinguish the line/dot/startline while performing different actions accordingly. When executing the find() function, it determines the action to be taken through sensors and proceeds to execute it using respective functions.<br>Corrections due to leaving the line and requiring rotation are handled separately within out_line_left/out_line_right if statements.<br>reverse_find() function is similar to find(), but is designed assuming backward movement instead of forward |

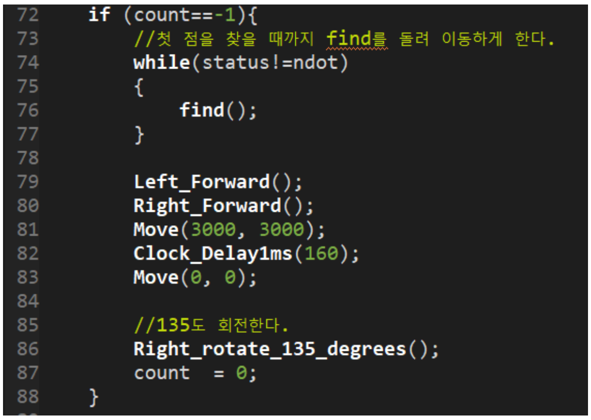
## 2. Map index

| Entire track vertex | vertex_list[ ][ ] (Phase 1) | graph[ ][ ] (For Phase 2) |
|---|---|---|
|  |  |  |

## 3. [Phase 1] Map memorization part

Exploring a map using the characteristics of a map

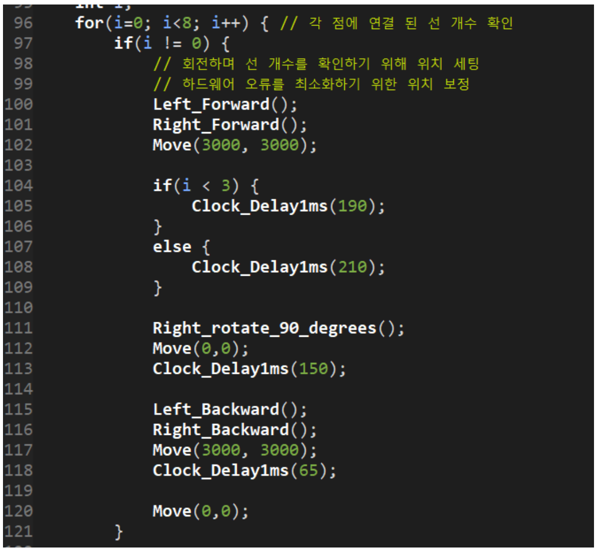
### (1) start & initialization



```
    if (count==-1){
        //첫 점을 찾을 때까지 find를 돌려 이동하게 한다.
        while(status!=ndot)
        {
            find();
        }

        Left_Forward();
        Right_Forward();
        Move(3000, 3000);
        Clock_Delay1ms(160);
        Move(0, 0);

        //135도 회전한다.
        Right_rotate_135_degrees();
        count  = 0;
    }
```

the device starts from the start line, and it keeps moving until when it finds the first dot. when it finds the first dot, it rotates approximately 135 degrees to the right. This initialization is performed to minimize errors such as duplicate checks or ignoring the edges in rotation and edge count process. After rotating 135 degrees, 3,4 sensors are on a blank area.

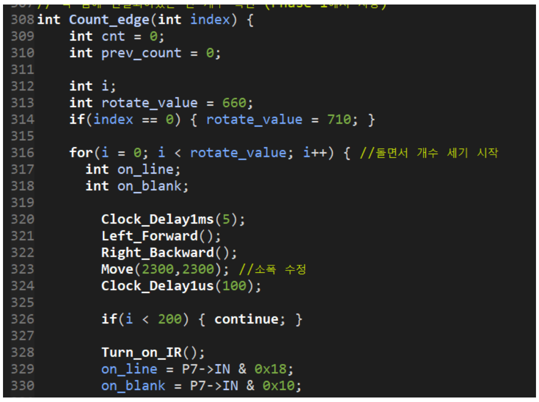
## (2) Position adjustment for rotation



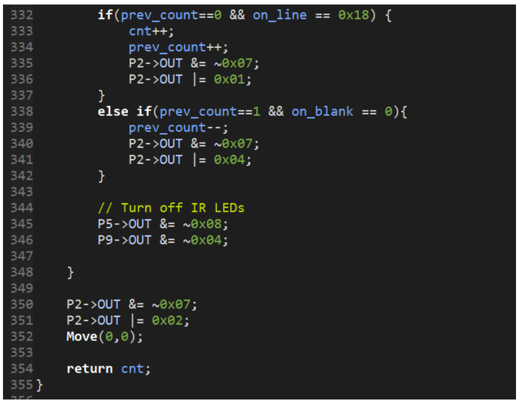
```c
    for(i=0; i<8; i++) { // 각 점에 연결 된 선 개수 확인
        if(i != 0) {
            // 회전하며 선 개수를 확인하기 위해 위치 세팅
            // 하드웨어 오류를 최소화하기 위한 위치 보정
            Left_Forward();
            Right_Forward();
            Move(3000, 3000);

            if(i < 3) {
                Clock_Delay1ms(190);
            }
            else {
                Clock_Delay1ms(210);
            }

            Right_rotate_90_degrees();
            Move(0,0);
            Clock_Delay1ms(150);

            Left_Backward();
            Right_Backward();
            Move(3000, 3000);
            Clock_Delay1ms(65);

            Move(0,0);
        }
```

This is the position adjustment performed at each dot (except for dot 0). point 0 has an edge to the start line, only the initialization is carried out. Because of the hardware issue of gradually moving backward when rotating, the device is moved to a position where it can check all lines to minimize errors. Additionally, from point 3 onwards, errors become significant, so additional adjustments are made accordingly.

## (3) Counting edges while rotating



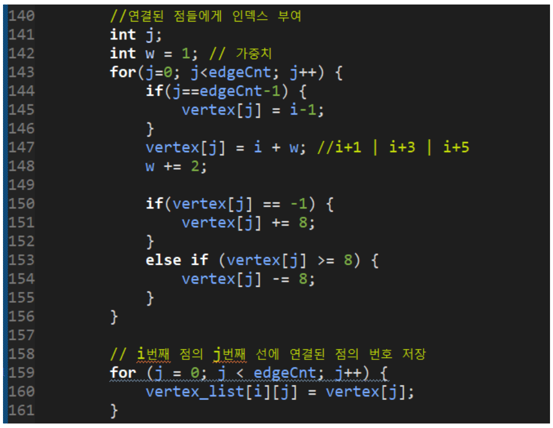
```c
int Count_edge(int index) {
    int cnt = 0;
    int prev_count = 0;

    int i;
    int rotate_value = 660;
    if(index == 0) { rotate_value = 710; }

    for(i = 0; i < rotate_value; i++) { //돌면서 개수 세기 시작
      int on_line;
      int on_blank;

        Clock_Delay1ms(5);
        Left_Forward();
        Right_Backward();
        Move(2300,2300); //소폭 수정
        Clock_Delay1us(100);

        if(i < 200) { continue; }

        Turn_on_IR();
        on_line = P7->IN & 0x18;
        on_blank = P7->IN & 0x10;
```



```c
        if(prev_count==0 && on_line == 0x18) {
            cnt++;
            prev_count++;
            P2->OUT &= ~0x07;
            P2->OUT |= 0x01;
        }
        else if(prev_count==1 && on_blank == 0){
            prev_count--;
            P2->OUT &= ~0x07;
            P2->OUT |= 0x04;
        }

        // Turn off IR LEDs
        P5->OUT &= ~0x08;
        P9->OUT &= ~0x04;

    }

    P2->OUT &= ~0x07;
    P2->OUT |= 0x02;
    Move(0,0);

    return cnt;
}
```

Rotating in place and checking the number of times sensors 3 and 4 recognized the line. The Count_edge(i) function is used, and i represents the point number. In the case of dot 0, because the initial rotation degree is different, it rotates for a longer time. When Count_edge(i) is executed, it rotates to the left a set number of times and checks the number of edges. In order to avoid duplicate checks on the edges, empty spaces are detected and counted again. When rotation is completed, the number of edges are returned. If the returned count is correct, it goes to step 4, but if it is incorrect, the yellow LED lights up and stops.

### (4) Save edge-dot connection information



```c
        //연결된 점들에게 인덱스 부여
        int j;
        int w = 1; // 가중치
        for(j=0; j<edgeCnt; j++) {
            if(j==edgeCnt-1) {
                vertex[j] = i-1;
            }
            vertex[j] = i + w; //i+1 | i+3 | i+5
            w += 2;

            if(vertex[j] == -1) {
                vertex[j] += 8;
            }
            else if (vertex[j] >= 8) {
                vertex[j] -= 8;
            }
        }

        // i번째 점의 j번째 선에 연결된 점의 번호 저장
        for (j = 0; j < edgeCnt; j++) {
            vertex_list[i][j] = vertex[j];
        }
```

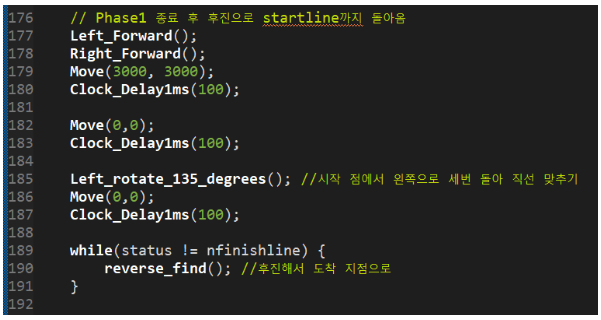
Stores the connection information of the edges connected to the point. The edges are rotated and designated as 0, 1, 2, 3 in the reverse order of recognition. Using the characteristics of the map and the current point number, edge 0 is +1 of the current point number, edge 1 is +3, edge 2 is +5, and edge 3 is -1 (+8 if a negative number is identified). (In addition, if a number greater than 8 is confirmed, add -8 to match the number.) By using this, we can assign a number and save the connection information. This connection information is stored in the vertex list, and the vertex list is stored in vertex_list, which stores the connection information of all dots.

### (5) iteration

Iterate steps 2 to 4 until all dots are checked.

### (6)

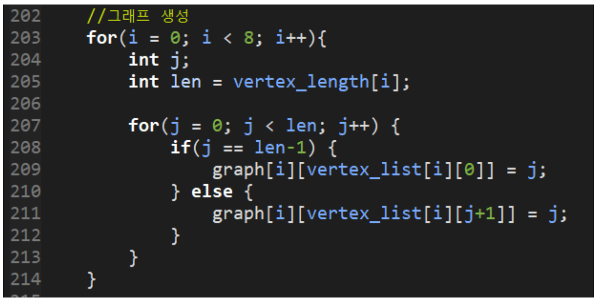
When all dots have been searched, return to dot 0, turn left, move backwards, and return to the starting position to end the search.



```c
    // Phase1 종료 후 후진으로 startline까지 돌아옴
    Left_Forward();
    Right_Forward();
    Move(3000, 3000);
    Clock_Delay1ms(100);

    Move(0,0);
    Clock_Delay1ms(100);

    Left_rotate_135_degrees(); //시작 점에서 왼쪽으로 세번 돌아 직선 맞추기
    Move(0,0);
    Clock_Delay1ms(100);

    while(status != nfinishline) {
        reverse_find(); //후진해서 도착 지점으로
    }
```

## 4. [Phase 2] One Stroke Drawing

### (1) Saving the Graph



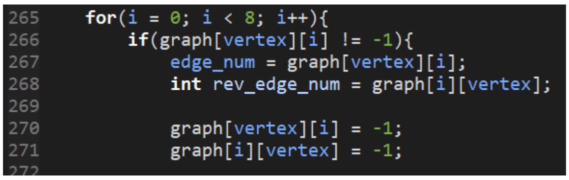
```
    //그래프 생성
    for(i = 0; i < 8; i++){
        int j;
        int len = vertex_length[i];

        for(j = 0; j < len; j++) {
            if(j == len-1) {
                graph[i][vertex_list[i][0]] = j;
            } else {
                graph[i][vertex_list[i][j+1]] = j;
            }
        }
    }
```

The `graph[i][j]` stores the edge number from vertex i to vertex j. The `vertex_list[ii][jj]` stores the vertex connected to the jj-th edge of vertex ii. Since the order of the stored edges is different, we need to match the order when saving the graph array using the vertex_list array. For example, if `vertex_list[0] = {1, 3, 5, 7}`, then `graph[0][3]=0`, `graph[0][5]=1`, `graph[0][7]=2`, and `graph[0][1]=3` should be aligned accordingly.

### (2) FindEuler(int vertex)

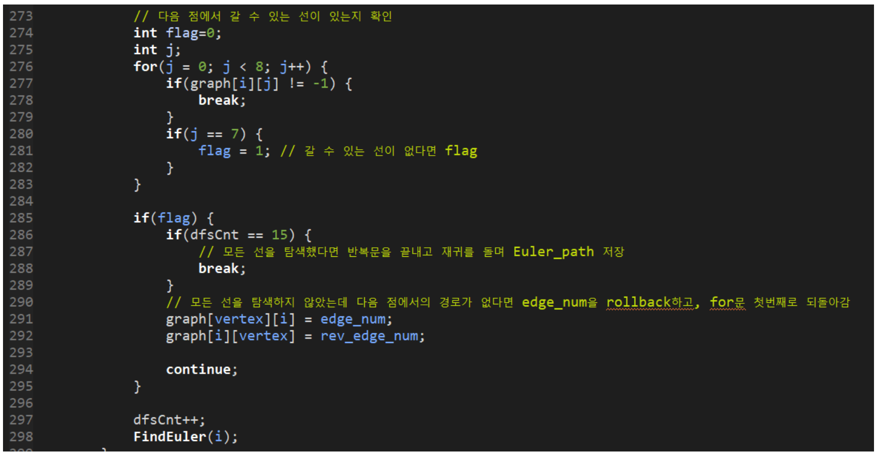
This is a DFS traversal. It explores in the order of 0 -> 1 -> 2 -> ⋯ -> 7, checking for available edges.

- If an edge(!= -1) exists, it records the path and changes the edge to -1.



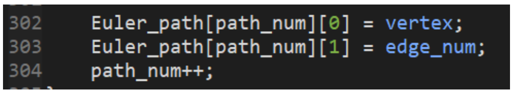
```
    for(i = 0; i < 8; i++){
        if(graph[vertex][i] != -1){
            edge_num = graph[vertex][i];
            int rev_edge_num = graph[i][vertex];

            graph[vertex][i] = -1;
            graph[i][vertex] = -1;
```

- If the entire path is not completed and there are no available edges from the next vertex, it restores the graph information and moves to the next iteration of the for-loop.



```
            // 다음 점에서 갈 수 있는 선이 있는지 확인
            int flag=0;
            int j;
            for(j = 0; j < 8; j++) {
                if(graph[i][j] != -1) {
                    break;
                }
                if(j == 7) {
                    flag = 1; // 갈 수 있는 선이 없다면 flag
                }
            }

            if(flag) {
                if(dfsCnt == 15) {
                    // 모든 선을 탐색했다면 반복문을 끝내고 재귀를 돌며 Euler_path 저장
                    break;
                }
                // 모든 선을 탐색하지 않았는데 다음 점에서의 경로가 없다면 edge_num을 rollback하고, for문 첫번째로 되돌아감
                graph[vertex][i] = edge_num;
                graph[i][vertex] = rev_edge_num;

                continue;
            }

            dfsCnt++;
            FindEuler(i);
```

- Once all 16 paths are explored, they are stored in Eulerpath[][] array.



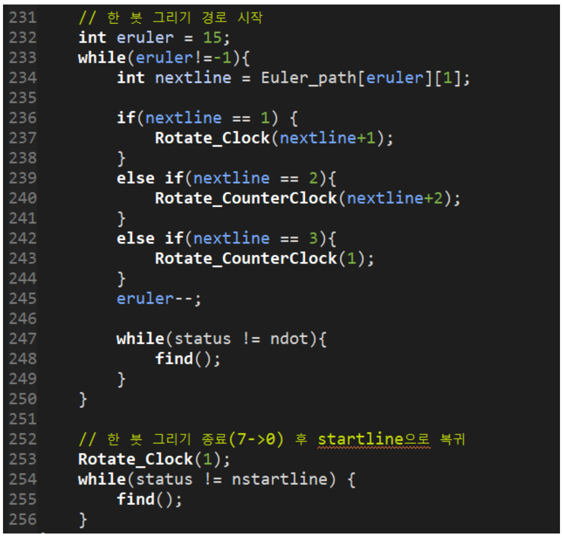
```
Euler_path[path_num][0] = vertex;
Euler_path[path_num][1] = edge_num;
path_num++;
```

After this function finishes, the movement path is stored as follows (for convenience, only the path is described):

0 1 2 3 0 5 2 7 4 1 6 3 4 5 6 7 0

Move outer line 3 times -> Move inner line 8 times -> Move outer line 5 times

### (3) One-stroke Drawing Move



```
// 한 붓 그리기 경로 시작
int eruler = 15;
while(eruler!=-1){
    int nextline = Euler_path[eruler][1];

    if(nextline == 1) {
        Rotate_Clock(nextline+1);
    }
    else if(nextline == 2){
        Rotate_CounterClock(nextline+2);
    }
    else if(nextline == 3){
        Rotate_CounterClock(1);
    }
    eruler--;

    while(status != ndot){
        find();
    }
}

// 한 붓 그리기 종료(7->0) 후 startline으로 복귀
Rotate_Clock(1);
while(status != nstartline) {
    find();
}
```

1. Rotate to find the line according to the edge number
2. Move with `find()` until reaching the dot
3. Repeat step 1-2
4. After the one-stroke drawing is completed, return to the start line and stop.

To minimize errors due to hardware issues (such as motor speed), we set the clockwise/counterclockwise rotation according to the edge number.

## 5. Trouble shooting

**Problem**: When writing and running the code, in addition to cases where there is a problem with the code itself, it does not operate completely ideally due to the characteristics of the hardware. For example, there were cases where the sensor did not stop at the correct location due to linear acceleration, cases where the sensor did not recognize a point at a certain distance due to subtle errors, errors due to friction and foreign substances, and errors due to remaining battery power, etc. Also, the speed of the two motors was slightly different, so there was a problem that the device was gradually pushed back when rotating.

This was solved using the following method.

1. Before moving on to the next movement, stop and reset to ensure that the previous movement, such as acceleration, has as little influence on the next movement as possible.

2. The speed of the motor is adjusted within a similar range as much as possible to minimize the influence of inertia, etc.

3. Reduce the probability of error occurrence through appropriate speed value and sensor recognition processing.

4. To reduce the phenomenon of being pushed away during rotation or errors gradually occurring due to acceleration, etc., a position correction operation is added to correct the position so that the device can arrive and operate at the ideal location.

5. (Phase 1) Replace with a code that minimizes the number of searches to reduce errors gradually accumulating and randomly determining success when searching multiple edges.