EIS1015 Line Tracer Report

Name	Student Number
김재형	2018008068
조현수	2018009152

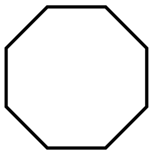
Team No.15

Line Tracer No.20

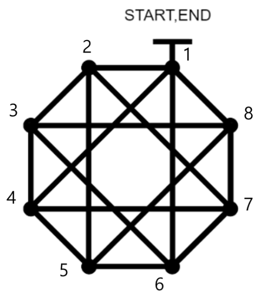
Phase 1 [Memorize the route]

📖 Prior knowledge of the map

1. The map was previously defined as an **octagon**
2. While traveling around each point, We checked whether **a line segment existed on a 45°** basis.
3. After touring all the maps, we assumed that all parts of the Linetracer Map would be stored.



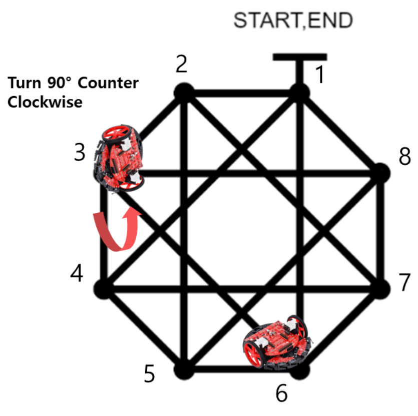
🎨 Design



1. **8×8 Graph Connection Relationships**
    - A two-dimensional array was used to express the connection relationship between points.
    - Since it is a two-way graph, the 8×8 two-dimensional array is diagonal symmetric.
    - It is expressed as 1 if it is connected, and 0 if it is not connected.

To \ From	Point 1	Point 2	Point 3	Point 4	Point 5	Point 6	Point 7	Point 8
Point 1	0	1	0	1	0	1	0	1
Point 2	1	0	1	0	1	0	1	0
Point 3	0	1	0	1	0	1	0	1
Point 4	1	0	1	0	1	0	1	0
Point 5	0	1	0	1	0	1	0	1
Point 6	1	0	1	0	1	0	1	0
Point 7	0	1	0	1	0	1	0	1
Point 8	1	0	1	0	1	0	1	0

2. **8×8 Turn Angle Connection Relationships**
    - Expression of **how many times the tracer have to turn 45°** counterclockwise to get from Point_x to Point_y (Suppose Line Tracer is looking at Point_(x+1) at Point_x)
    - Below Picture Says that If the line tracer wants to go Point_3 to Point_6, the tracer should turn 90° **[The table below also points to a value of 2 in (3,6)]**
    - [x,y] = -1 means that there is no Road Between Point_x to Point_y.



To \ From	Point 1	Point 2	Point 3	Point 4	Point 5	Point 6	Point 7	Point 8
Point 1	-1	0	-1	1	-1	2	-1	3
Point 2	3	-1	0	-1	1	-1	2	-1
Point 3	-1	3	-1	0	-1	1	-1	2
Point 4	2	-1	3	-1	0	-1	1	-1
Point 5	-1	2	-1	3	-1	0	-1	1
Point 6	1	-1	2	-1	3	-1	0	-1
Point 7	-1	1	-1	2	-1	3	-1	0
Point 8	0	-1	1	-1	2	-1	3	-1

In Phase 1, our final goal is to complete both of the above tables in a sequential rotation from point 1 to point 8.

🧑‍💻 Algorithm Implementation

1. Starting Part Implementation
    - The current point may deviate from the while syntax only when the T-shaped section is recognized by all infrared sensors. (scan = 0b1000 date and time, T-shaped section.)
    - The Find_node function means that it will continue to progress until it finds the Node.

```
// Main Function
  while(scan != 0b1000)
  {
      scan = direction();
  }
  find_node(NODE);
  stop(100);
  Rotate_Clockwise(90);
  stop(100);
```

2. Sequential Rotation From point_1 to point_8
3. Turn 45° for each point 3 times and check with infrared sensor if there is line segment. (If present, mark it on the array.)
    - At the beginning, we really wanted to use the infrared sensor to check the existence of edges but there were some problems (explained at Difference (5)). So we just artificially checked edge every 45°.
    - What the get_nxtnode function means is It is a function of finding the number of the cnt-th connected point at the cur Point.

```
    for(cur=1; cur<=8; cur++)
    {
        for (cnt=0; cnt<4; cnt++)
        {
            if (1)
            {
                nxt = get_nxtnode(cur, cnt);
                angle[cur][nxt] = cnt;
            }
            if (cnt != 3)
            {
                Rotate_Counter_Clockwise(40);
                stop(300);
            }
        }
        Rotate_Clockwise(140);
        find_node(NODE);
        Rotate_Counter_Clockwise(40);
        stop(300);
    }
```

4. After defining the angle array, a new edge array is created through the defined array.

```
for (i=1; i<=8; i++)
{
    edge[i][j] = 0;
    for (j=1; j<=8; j++)
    {
        if (angle[i][j] >= 0)
            edge[i][j] = 1;
    }
}
```

Phase 2 [One-brush Drawing]

🎨 Design

1️⃣ Designing a One-Brush Drawing Algorithm Using Depth First Search (DFS)

We will design an Eulerian path finding algorithm using a depth-first search approach, utilizing a 2D adjacency matrix, `edge`, as the representation of the graph. The steps of the algorithm are as follows:

1. Start from a designated starting point `s`.
2. Search for vertices connected to `s` using the `edge` matrix:
   a. If a connection exists and the vertex has not yet been visited, visit this vertex and move to it.
   b. Repeat step 2 recursively, following the depth-first search methodology.
3. On completing the visits, traverse the vertices in the reverse order of their visiting time to form the Eulerian path.

As a result of the algorithm, the vertices will be visited in the following sequence to complete the Eulerian path:

1 → 2 → 3 → 4 → 1 → 6 → 3 → 8 → 5 → 2 → 7 → 4 → 5 → 6 → 7 → 8 → 1.

This sequence ensures that each edge is traversed exactly once, forming a valid Eulerian circuit if the graph allows for it.

2️⃣ Designing an Algorithm for Line Tracer Rotation Optimization

- If the device rotates more than 180 counterclockwise, it may be seen that it is more optimized to rotate clockwise.
- Thus, if the count value that the machine should turn is larger than 180°, We set the machine should turn for Clockwise, not for the Counter_Clockwise (Basic Driection)

👨‍💻 Algorithm Implementation

- **The One-Bush Drawing algorithm is as follows.**

```
void dfs(int cur)
{
    int nxt;
    for (nxt = 1; nxt <= 8; nxt++)
    {
        if (edge[cur][nxt])
        {
            edge[cur][nxt]--;
            edge[nxt][cur]--;
            dfs(nxt);
        }
    }
    ans[++top] = cur;
}
```

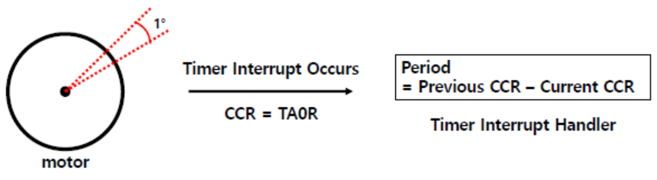
- **Line Tracer Rotation Optimization**

```
int tmp = 0;
cur = ans[top--];
while (top >= 0)
{
    nxt = ans[top--];
    cnt = angle[cur][nxt] + tmp;
    if(cnt <=4){
        while(cnt--)
        {
            Rotate_Counter_Clockwise(45);
        }
    }
    else{
        cnt = 8 - cnt;
        while(cnt--){
            Rotate_Clockwise(45);
        }
    }
    find_node(NODE);
    tmp = angle[cur][nxt] + 1;
    cur = nxt;
}
stop(100);
}
```

Difference Between expected and actual results

1️⃣ Motor Rotate is different from the real rotate.

- Initially, we found the appropriate motor speed and time variables for the motor to rotate 45°. However, as the battery was used for a long time, the speed of the motor slowed down and we had to find a way to solve it. Because the speed of the motor was variable depending on the battery usage.
- For Implementation and Capture exact Motor Rotate value, We should set Timer Interrupt Setting. And the following Code is Below



```
void timer_A3_capture_init() {
    P10->SEL0 |= 0x30;
    P10->SEL1 &= ~0x30;
    P10->DIR &= ~0x30;

    TIMER_A3->CTL &= ~0x0030;
    TIMER_A3->CTL = 0x0200;

    TIMER_A3->CCTL[0] = 0x4910;
    TIMER_A3->CCTL[1] = 0x4910;
    TIMER_A3->EX0 &= ~0x0007;

    NVIC->IP[3] = (NVIC->IP[3]&0x0000FFFF) | 0x404000000;
    NVIC->ISER[0] = 0x0000C000;
    TIMER_A3->CTL |= 0x0024;
}

void Rotate_Clockwise(int degree){
    degree = degree * 2;
    count = 0;
    while(1){
        Left_Forward();
        Right_Backward();
        Move(SPEED, SPEED);
        if(count>degree){
            Move(0,0);
            stop(1);
            break;
        }
    }
}

void Rotate_Counter_Clockwise(int degree){
    degree = degree * 2;
    count = 0;
    while(1){
        Left_Backward();
        Right_Forward();
        Move(SPEED, SPEED);
        if(count>degree){
            Move(0,0);
            stop(1);
            break;
        }
    }
}
```

2️⃣ **Line tracer's Speed becomes slow if the battery used for a long time.**

- The longer the battery was used, the slower the speed of the motor was, so we should have prepared several extra batteries.

3️⃣ **Adjustment to go straight**

Since it was not possible to rotate completely, it was necessary to design an algorithm that had to be adjusted if it deviated from the straight path even a little while going straight (battery issue, motor issue).

If only one of the infrared sensors that detect straightness is being detected, it is not going straight properly, so it is designed to go straight properly.

Following Code is Below

```
while(!((scan=direction()) & node))
{
  moveForward(SPEED,SPEED,1);
  loadSensor();
  if (!(P7->IN & 0b00001000))
  {
      while((P7->IN & 0b00011000) != 0b00011000)
      {
          loadSensor();
          Left_Backward();
          Right_Forward();
          Move(0, SPEED);
      }
  }
  else if (!(P7->IN & 0b00010000))
  {
      while((P7->IN & 0b00011000) != 0b00011000)
      {
          loadSensor();
          Left_Forward();
          Right_Backward();
          Move(SPEED, 0);
      }
}
}
```

- If it is skewed to the right when going straight, only the right wheel is moved to the left.

4️⃣ **Set speed numbers to pass Phase 2 in 1 minute**

- In order to pass normally within time, the speed of the motor was initially set to 2500, but it was set to 3000 and finally set to 3300.

5️⃣ Detection of orientation through infrared detection by the sensor

- In order for the line tracer to head to the desired line segment, it is necessary to check exactly what the line tracer is currently detecting. Directions were detected with the following code.

```
int direction() {
    loadSensor();

    int res = 0;

    if (P7->IN == 0b11111111)
        res += 0b1000;
    else if (P7->IN == 0b00111100)
        res += 0b0100;
    else if((P7->IN & 0b00011000) == 0b00011000)
        res += 0b0010;
    // white
    else if (P7->IN == 0)
        res += 0b0001;
    return res;
}
```

5️⃣ Detection of linked edges on the current node.

- We just check the edge every 45° base on the information about the Prior knowledge of the map (an octagon).
- Because if we want to check every edge linked at current node with the infrared sensor, we have to move forth a little bit to check an edge and then back a little bit to return to the current node.
- This forth and back movements are inaccurate, so the distance of each movement is usually changed and that makes the line tracer's total movement also changed randomly.
- These are the reasons why we selected this solution (without sensing) unwillingly.

```
if (1)
{
    nxt = get_nxtnode(cur, cnt);
    angle[cur][nxt] = cnt;
}
```