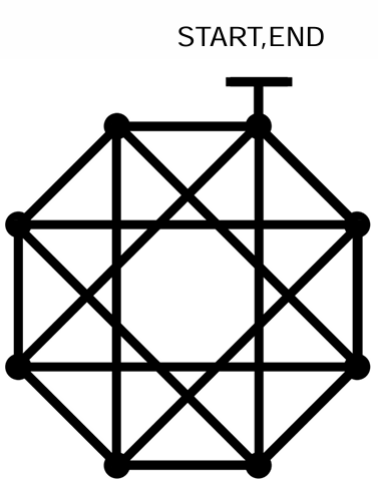
Linetracer Final Report

Overall, our group's Linetracer completed the second stage of the one-stroke drawing process very well, and it was completed in exactly one minute. In addition, we did not specify a trajectory for the car, but declared each edge endpoint and angle in the car code, and used the DFS algorithm to get a one-stroke path. In this way, the car can know each point of the one-stroke and the angle to turn between two points in actual operation. We also tried to complete the first phase, but encountered problems when collecting edge information on the vertices.

Below is our idea and code for the first and second phases.

Phase 1. Memorize the route using the memorize algorithm.

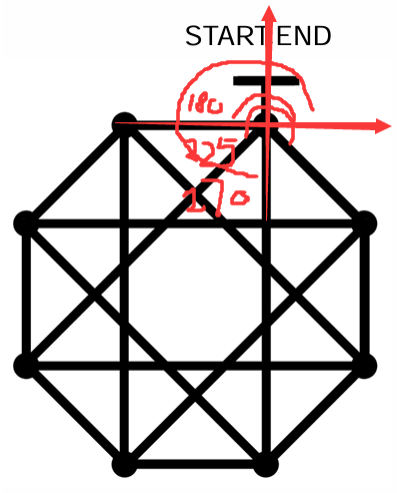
If we just specify the path for the car manually, it is relatively easy. However, how to let the car recognize the ID of the current vertex and the angle of each edge is a more difficult problem, which is what we need to complete in the first stage.



The relative angle of the car and the absolute angle of the edge

When the car moves at a vertex, it needs to turn to enter a certain edge, but due to the difference in the incoming edge, the angle of the outgoing edge will also change accordingly. However, we found that no matter how the car moves, the absolute angle of each edge in the map will not change.

The absolute angle of an edge refers to the angle between the edge and the positive direction of the x-axis in the xy coordinate system established in the map, ranging from 0 to 360. Here is a simple example. It is worth noting that since each edge is actually a bidirectional edge, the absolute angles in different directions must be considered separately.



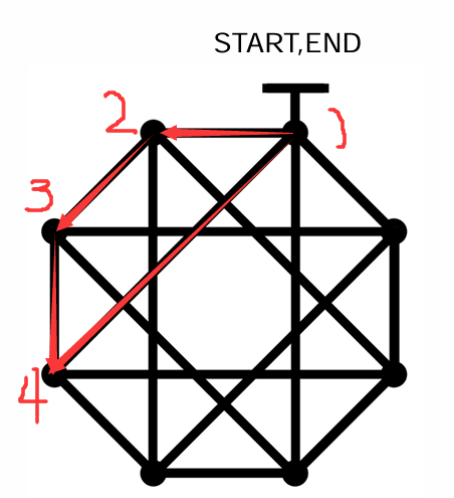
This part is an important part of the car's steering module. Since the information collection stage was not successfully implemented, we try to tell the car relatively original information for the car to process. Below is the source code where we set the absolute angle of each edge. The number of each vertex starts from 0 at the starting point and increases in counterclockwise order.

```
void set_matrix(AdjacencyMatrix* matrix){
    setEdge(matrix, 0, 1, 180);
    setEdge(matrix, 1, 0, 0);
    setEdge(matrix, 0, 3, 225);
    setEdge(matrix, 3, 0, 45);
    setEdge(matrix, 0, 5, 270);
    setEdge(matrix, 5, 0, 90);
    setEdge(matrix, 0, 7, 315);
    setEdge(matrix, 7, 0, 135);

    setEdge(matrix, 1, 2, 225);
    setEdge(matrix, 2, 1, 45);
    setEdge(matrix, 1, 4, 270);
    setEdge(matrix, 4, 1, 90);
    setEdge(matrix, 1, 6, 315);
    setEdge(matrix, 6, 1, 135);

    setEdge(matrix, 2, 3, 270);
    setEdge(matrix, 3, 2, 90);
    setEdge(matrix, 2, 5, 315);
    setEdge(matrix, 5, 2, 135);
    setEdge(matrix, 2, 7, 0);
    setEdge(matrix, 7, 2, 180);

    setEdge(matrix, 3, 4, 315);
    setEdge(matrix, 4, 3, 135);
    setEdge(matrix, 3, 6, 0);
    setEdge(matrix, 6, 3, 180);

    setEdge(matrix, 4, 5, 0);
    setEdge(matrix, 5, 4, 180);
    setEdge(matrix, 4, 7, 45);
    setEdge(matrix, 7, 4, 225);

    setEdge(matrix, 5, 6, 45);
    setEdge(matrix   , 6, 5, 225);
```

```
    setEdge(matrix, 6, 7, 90);
    setEdge(matrix, 7, 6, 270);
}
```

Distinguish each different vertex

If the car is traveling counterclockwise along the outermost edge, it only needs to turn left 45 degrees when it encounters each vertex to drive to the next vertex. But the more critical question is how to distinguish each vertex that has been passed.



So the following question arises: how to determine whether the two paths 1-2-3-4 and 1-4 reach the same vertex.

Coordinate method

At the beginning, I wanted to use the position of the coordinate system to uniquely identify each vertex, that is, record the angle and length of each edge, and get the rectangular coordinate system coordinates. If the coordinates obtained by the two paths are equal or approximate, then it can be determined to be the same vertex.

This method is easier to think of, but due to the existence of the 45-degree angle, the square root of 2 makes the calculation and judgment process extremely difficult, so I chose to give up later.

Absolute angle identification method

When observing each vertex, it is found that the absolute angle of each edge under each vertex is different, so comparing the absolute angle matrix of the edges of the vertices may be a better choice.

Below is my definition of point in the code. Although it is commented out, I have tried it before.

```
typedef struct{
    int id;
    int edge_number;
    int edge_absolute_angle[MAX_EGDES];
} Point;

typedef struct{
    int id;
    int edge_absolute_angle[8];
} Point;
```

These are two ways to define points. One is to directly use the edge matrix to store edge information. However, due to the different orders in which edges are recognized when scanning edges, the order of storage will also be different, so it is necessary to handle the disorder when judging. Another method is relatively simple. It directly divides 360 into 8 absolute angles. If there is an edge at a certain absolute angle position, it is 1, otherwise it is 0.

Compare the differences in the edge matrices to determine the differences in the points

The most direct way is to compare each element in the matrix one by one. Another way is to HASH the absolute angle of each edge and then get a unique value that can distinguish different vertices.

Euler circuit

Here, we use the adjacency matrix information collected earlier and a simpler DFS method to obtain a one-stroke path connected by vertices. Below is the source code.

```
void dfs(Graph* graph, int v, int* path, int* path_index) {
    int u;
    for (u = 0; u < graph->num_nodes; u++) {
        while (graph->matrix[v][u] > 0) {
            graph->matrix[v][u]--;
            graph->matrix[u][v]--;
            dfs(graph, u, path, path_index);
        }
    }
    path[(*path_index)++] = v;
}

    Graph graph = {
        .matrix = {
            {0, 1, 0, 1, 0, 1, 0, 1},
            {1, 0, 1, 0, 1, 0, 1, 0},
            {0, 1, 0, 1, 0, 1, 0, 1},
            {1, 0, 1, 0, 1, 0, 1, 0},
            {0, 1, 0, 1, 0, 1, 0, 1},
            {1, 0, 1, 0, 1, 0, 1, 0},
            {0, 1, 0, 1, 0, 1, 0, 1},
            {1, 0, 1, 0, 1, 0, 1, 0},
        },
        .num_nodes = 8
    };
    int path[MAX_NODES * MAX_NODES];
    int path_index = 0;
    dfs(&graph, 0, path, &path_index);
    // PRINT PATH:
    // [CORTEX_M4_0] 0 1 2 3 0 5 2 7 4 1 6 3 4 5 6 7 0
```

Phase 2. Perform one-brush drawing based on a memorized route

This is the main function that drives the car in the MAIN function.

```
    start_from_start_line();
    int i = path_index - 1;
    int edge_angle;
    while(i > 0) {
        edge_angle = getEdgeAngle(&matrix, path[i], path[i-1]);
        i--;
        rotate_with_given_angle(current_angle, edge_angle);
        current_angle = edge_angle;
        move_straight_to_point();
    }
    // move to end line
    rotate_with_given_angle(current_angle, 90);
    start_to_end_line();
```

`start_from_start_line();`

This function allows the car to stop and detect the sensor, and turn on the green light when the start line is detected. At this time, pressing the switch can start the car after a delay of 2s. The car will move to the first vertex and stop.

`edge_angle = getEdgeAngle(&matrix, path[i], path[i-1]);`

This line queries the previously stored edge absolute angle information, thereby obtaining the edge absolute angle information from the current vertex to the next vertex in the stroke path.

`rotate_with_given_angle(current_angle, edge_angle);`

Then, the relative angle will be calculated based on the current angle of the car and the angle of the required driving side, and the relative angle will be used to determine whether to turn left or right.

`move_straight_to_point();`

This is the main function for the car's movement along the line, which includes straight driving, deviation correction, top condition checking, etc.

```
void move_straight_to_point(void){
    move_forward_time(200);
    int sensor_straight, sensor, sensor_point;
    while(1){
        sensor = read_sensor();
        sensor_straight = sensor & sensor_straight_mask;
        // there exist 11111111 black line so cannot judge sensor_point_mask
        if(sensor_straight == sensor_straight_mask){
            Led_Off();
            move_forward_time(50);
            Clock_Delay1ms(10);
        }
        else if(sensor_straight == 0x08 || sensor_straight == 0x10){
            correct_line(sensor_straight);
        }
        else if(sensor_straight == 0x00){
            // 00000000
            Led_Off();
            Move(0, 0);
```

```
            P2->OUT |= 0X01;
            break;
        }
        else break;
    }
}
```

We use LEDs to indicate the current state of the robot's motion. For example, a blue light will be on during the correction phase.

`correct_line(sensor_straight);` In the correction function, perhaps it is an option to let the car rotate, that is, the left and right wheels are not in the same direction of movement, but because spinning in place will affect the speed of the car, it is chosen that the left and right wheels move forward at the same time, but the speed of one wheel is 0.

```
void correct_line(int sensor_straight){
    if(sensor_straight == 0x08){
        // 00001000
        Led_Off();
        right_rotate_slight();
        P2->OUT |= 0X04;
    }
    else if(sensor_straight == 0x10){
        // 00010000
        Led_Off();
        left_rotate_slight();
        P2->OUT |= 0X04;
    }
}
void right_rotate_slight(void){
    Left_Forward();
    Right_Forward();
    Move(3000, 0);
    Clock_Delay1ms(10);
}
void left_rotate_slight(void){
    Left_Forward();
    Right_Forward();
    Move(0, 3000);
    Clock_Delay1ms(10);
}
```

left code

When you finish the stroke and return to vertex 0, you need to return to the starting line along the edge with an absolute angle of 90.

issues to note

During the actual operation of the car, due to battery loss, the waiting time for each turn of the same angle will change. Therefore, it is recommended to test the time required for a 180 or 360-degree rotation in advance before executing the main function, and perform the corresponding conversion.