Project Report

Shen Jiyuan

5130309194

1 Introduction

In this project, given a mini-car and references included, we will implement the requirement that the mini-car can automatically solve maze games.

Overview: The following statements will be exactly the same as the implementation process on the codes for my project, in other words, we will see a procedure including image banarization on input maze, path tracing by cpp maze algorithm, extraction on key point and checking for turnings.

2 Preparation

(1) Mini-car: I get one provided by our teacher, you can also buy one. Most of them are well capsulated, we only have to make changes on the steering engine.

(2) Singel-chip: I get one provided by our teacher, you can also buy one. It is tied on mini-car.

3 Structure Sketch

What we need to do is to embed programs in mini-car by USB, then the computer program sends messages to mini-car to run corresponding codes in the single-chip. In this way, we can construct the relationship between computer and mini-car. That is the real program for maze games runs in computer side, and the messages which are delivered to mini-cars control the car movements.

3.1 Single Chip side

Here we only need to embed codes for controlling mini-car. Normally the wheel controls forwarding while the steering engine controls turning. However, it demands more codes than expected. Therefore, we can tear down the steering engine and make use of the different turnings and speeds of the wheel. By implementing it in this way, we can finally achieve spinning around action as well, which I think is very helpful for mini-car to solve maze games where straight lines construct.

The commands in this side are 6-digit chars:

Bit 0 is $ as start bit.

Bit 1 is direction bit: forward or backward.

Bit 2 is direction bit: left or right.

Bit 3 is angle value bit.

Bit 4 is speed value bit.

Bit 5 is # as stop bit.

Then, we can determine control commands by carefully observing chars.

Here I only list back motor codes for reference:

```
void MOTOR_RIGHT_BACK(void)
{
	delay(1000);
	P2SEL &= ~BIT4;
	delay(2000);
	P2OUT &= ~BIT4;
	P2SEL |= BIT5;
}

void MOTOR_RIGHT_FORWARD(void)
{
	delay(1000);
	P2SEL &= ~BIT5;
	delay(2000);
	P2OUT &= ~BIT5;
	P2SEL |= BIT4;
}
```

3.2 Real-time Image Processing and mini-car Controlling

Image processing methods are approximately similar. But here we notice the special characteristic of maze, then we need to inverse the color of maze and thus we can get fined path of capable ways not walls. This processing is prepared for easier maze algorithm.

For inversing of color, just need to set valve of selected domain as close to white.

Take as an example below:

```
cvInRangeS(destimage1, CV_RGB(100, 100, 100), CV_RGB(255, 255, 255),
finalimage);
```

The important thing we should pay attention to is making judgement of what the mini-car need to do next according to the current image.

First, how to represent path? Discrete points! (turning points are best) Because mini-car is small, if points are too dense, movements will be frequent and make the whole process not fluent. Now we can get some turning points.

Second, how to represent location? Vectors! Observe the relationship between current location and next turning point. Use a 2-dimensional vector to represent location. Select middle point of mini-car head as beginning and middle point of mini-car tail as ending, then we have a vector. Then calculate the relative vector of location and turning point mentioned above, thus we have another vector. Now we can get two important vectors.

Third, how to adjust and move? Corners! It means the angle between the two vectors above. In math, we have cos and sin methods. We should use sin method. Because cos can cause confusion on left-right by always being positive numbers.

3.3 Maze Algorithm (A* Search Algorithm)

A* Search Algorithm is based on Breadth-First-Search and Priority Queue:

It solves problems by searching among all possible paths to the solution (goal) for the one that incurs the smallest cost (least distance travelled, shortest time, etc.), and among these paths it first considers the ones that *appear* to lead most quickly to the solution. It is formulated in terms of weighted graphs: starting from a specific node of a graph, it constructs a tree of paths starting from that node, expanding paths one step at a time, until one of its paths ends at the predetermined goal node.

At each iteration of its main loop, A* needs to determine which of its partial paths to expand into one or more longer paths. It does so based on an estimate of the cost (total weight) still to go to the goal node. Specifically, A* selects the path that minimizes $f(n)=g(n)+h(n)$ where n is the last node on the path, $g(n)$ is the cost of the path from the start node to n, and $h(n)$ is a heuristic that estimates the cost of the cheapest path from n to the goal. The heuristic is problem-specific. For the algorithm to find the actual shortest path, the heuristic function must be admissible, meaning that it never overestimates the actual cost to get to the nearest goal node.

4 Usage

Select a proper position to place camera in order to assure the whole maze is in.

Step1: Computer connect Bluetooth.

Step2: Click the four corners in the image for perspective transforming and binarilization processing.

Step3: (C++ Maze Processing to find all turning points)

Step4: Place mini-car in maze.

Step5: Click head and tail of mini-car.

Step6: (C++ Maze Algorithm Processing movements)