Intelligent navigation vehicle based on path planning

Shi Cheng

Table of contents:

1. Introduction

In recent years, with the rapid development of Internet of Things technology and artificial intelligence, more and more intelligent technologies have been integrated into production, life, and promoted the development of all walks of life. As a typical carrier of intelligent robots, intelligent car can realize automatic control or remote control through controller programming. It is an important assisting invention for intelligent technology to promote social development. This electronic design intends to make the car have a certain intelligent guiding role through intelligent path planning, helping or replacing humans to realize actions and explorations in complex environments. Finally, I chose to start from the point of view of the smart car and further think about the details of the relevant direction.

2. Background of topic selection

As a navigation vehicle capable of intelligent path planning, the navigation vehicle has the function of exploring and avoiding risks. When designing the car, I hope that the car can follow the route specified by the PC, avoid obstacles, prompt and other functions.

3. Design ideas

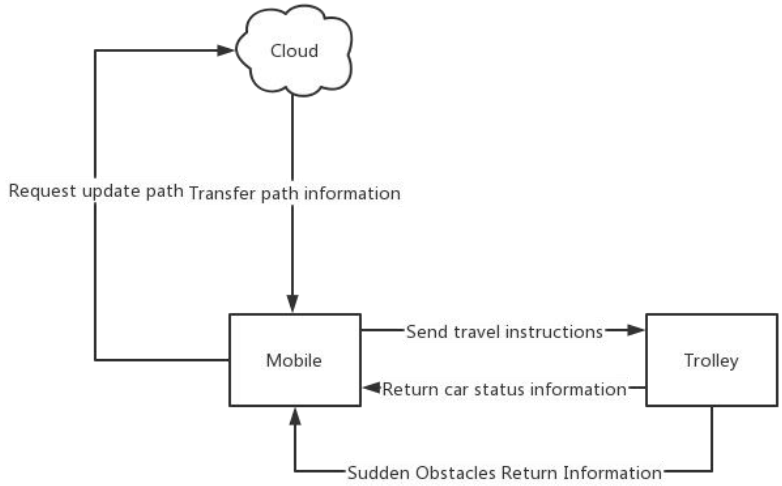
For the car itself, it is expected to implement the underlying basic control circuits and programs, and complete the underlying interface for the car control. Then realize the function of controlling the car through the mobile terminal, and the function of the car moving correspondingly by receiving Bluetooth data and moving forward according to the specified path. At the same time, the car can detect obstacles in the surrounding environment, and can send the corresponding information back to the computer through Bluetooth, which can complete real-time interaction. At the same time, the PC side can process the obstacle information and re-plan the path.

For the setting of the experimental environment, considering the need to fit the actual situation, an ideal road is set to provide an ideal environment for the operation of the car. Under such conditions, the final function that this group wants to achieve is to simulate the process of the intelligent transportation system: collect road data through external cameras and other equipment, and process the data on the computer to obtain the overall route forward; The bluetooth is sent to the car, and the car plans according to the route, and then performs intelligent driving to realize basic functions such as going straight and turning. In addition, additional functions include: in the face of emergencies such as obstacles, local route planning can be carried out, and obstacles can be automatically avoided; sound prompts can be issued for specific situations during operation; backward distance detection function: there is a rear The distance sensor can detect the distance of the backward object; when there is an instruction input from the computer, the car can complete the modification of the motion state and execute it according to the instruction.

It is hoped that the car can take into account practicability and robustness while pursuing sufficient intelligence, so that it can satisfy a wider range of application scenarios.

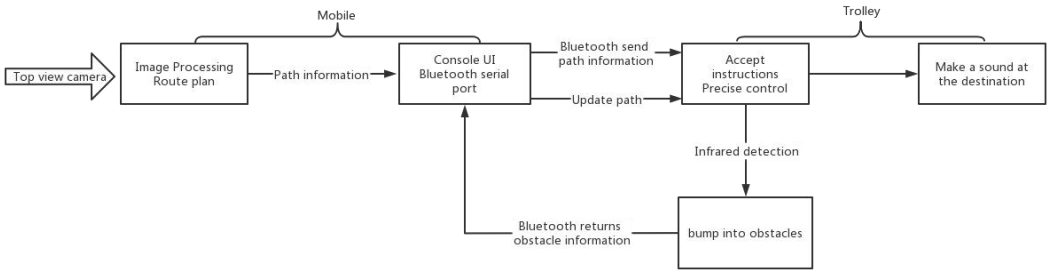
4. Introduction to the subject

The entire process that you want to simulate is shown in the following figure: The main nodes of the change are divided into three parts: the cloud, the mobile terminal and the car terminal. The cloud represents the data center's processing of network data streams and the location of geographic paths. The mobile terminal controls the car and the medium for interacting with the car. The car is the specific executor and the main carrier of services.



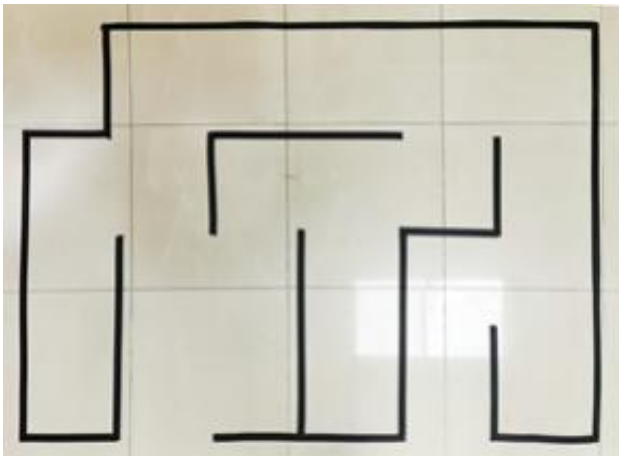
The main logic of the model is: when the mobile terminal enters the desired location, the cloud will return the path to provide the car with path information; after obtaining the information from the mobile terminal, it will guide the car to interact, and send the path information and related information to the car. command; At the same time, the trolley continuously returns status information to the mobile terminal. If a sudden obstacle is encountered, the trolley terminal will send information to the mobile terminal, and then the means will exchange information with the cloud to request a new path plan, thus looping.

In the design process of this trolley, taking into account the need to make full use of time and site resources, reduce costs as much as possible while maximizing the performance of the trolley. So I built a simplified model, the specific structure is shown below:



First of all, a black-frame map (as shown in the figure below) is built in the blank site, the starting point and key points are set, and at the same time, it is displayed by taking a bird's-eye view, which is used as a map for path planning to simulate a real map. Intuitive handling of maps. The image is then processed by a computer to get the final path.

After getting the path of the car, send the command to the car through the Bluetooth serial port, so that the car can travel according to the planned path. If the road is smooth and the car reaches the end point smoothly, the car will make a specific sound to notify that it has arrived. If the car encounters an obstacle, it will prompt the mobile terminal through the buzzer and signal, and then send a new path planning request to the mobile terminal. This means that the mobile terminal will take the current position of the car as a new starting point and re-route the route. Plan, and then issue corresponding instructions to make the car complete the entire path.



5. Detailed function introduction

5.1 The assembly of the car and the application of the ROS platform

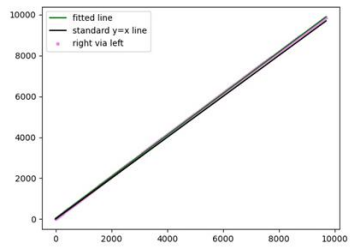
In the early stage of the experiment, the assembly of the car and the learning of Ros were carried out first. In the initial learning process, it took a certain amount of time for me to learn the hardware part of the car. Therefore, I choose to use the official ROS libraries such as motor, tachometer, and UART to complete the basic control of the car, and then continue to further learn the underlying principles while improving the hardware control of the car.

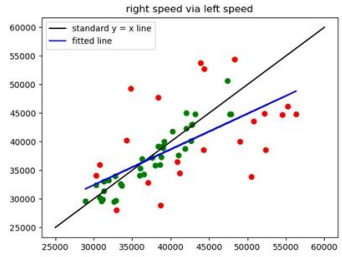
5.2 Data processing

After loading the car and having a basic understanding of ROS, the basic control of the car can be realized by calling the library function. At this time, this group uses the tachmeter library function in the library to judge the data of the car traveling. By letting the car move forward for a period of time, in the judgment, the returned data of the left and right wheel speeds and the number of rotations are obtained. By directly observing the data, it can be found that even when the set speed is the same, there are still many strange differences in the speed of the left and right wheels, and they are extremely unstable. On this basis, use python to fit points, save and input data through txt files, and the experimental code is shown in the following figure:

```
x = np.linspace(min(l_s), max(l_s), 300)
plt.scatter(l_s, -r_s, label='right via left', c='violet', s=10)
output = np.polyfit(l_s, -r_s, deg=1)
y = x * output[0] + output[1]
plt.plot(x, y, label='fitted line', c='green')
plt.plot(x, x, label='standard y=x line', c='black')
plt.legend()
plt.savefig('./l_r_step.jpg')
plt.show()
```

The image shown below is obtained: (The left picture is the fitting image of the left and right wheel speeds, and the right picture is the fitting value of the left and right wheel rotation angles, that is, the fitting value of the absolute distance)



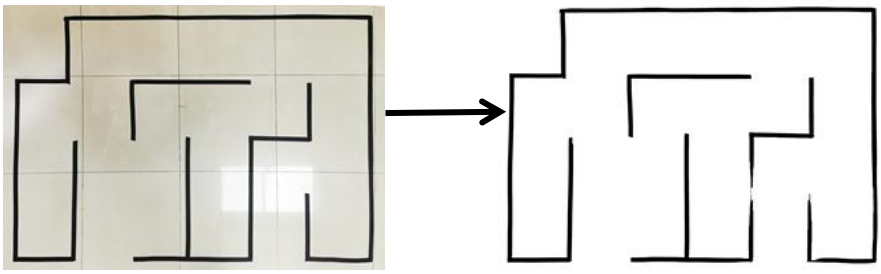


The left picture is the speed point diagram of the left and right wheels measured by the tachometer, in which the horizontal axis represents the left wheel speed, the vertical axis represents the right wheel speed, and the red point represents the speed error ratio of the left and right wheels when the set speed is the same For the points exceeding 10% (that is, the point with a large error), the black line represents the theoretical curve (that is, the left wheel speed is equal to the right wheel speed: y=x), and the dark blue represents the fitted curve. It can be found that the left and right wheel speeds returned by the tachometer are very unreliable, and the error is very large. The possible reasons are considered here: Since the motor is a stepper motor controlled by level pulses, instability may occur.

The picture on the right is the result of fitting the angles of the left and right wheels. The dark blue line in the picture is the fitting curve, and the black line is the standard curve. The points representing the number of rotations of the left and right wheels are covered by the blue line. The linearity of the number of rotations is good, and it may be used as a criterion for adjusting the straight attitude of the body.

5.3 Labyrinth Skeleton Extraction

In order to simulate specific navigation, a relatively simple map needs to be designed for path planning. In order to facilitate image processing, take a picture directly above the map, and then perform simple processing and denoising on the obtained image: first read the grayscale image through Python's cv library, and then use the halcon library to complete the extraction of the skeleton to eliminate redundant pixels. Interference of points (eg gaps between floor tiles in the map), complete the binarization of 0 (black) and 255 (white). The final effect is to convert the actual picture above into a black and white skeleton, as shown below:



5.4 Path planning algorithm

After obtaining the skeleton, the final feasible shortest path can be obtained through the path planning algorithm. The principle is as follows:

(1) Preprocess the image skeleton:

Mark the start and end points as well as the location of the trolley and the occupied area of the trolley. Here, the size of the car on the image is considered, and it is necessary to ensure that the size of the car is as close to the width of the path as possible, which can reduce the difficulty of path search; or in other words, no one wants to get the path in the white Parts are folded back and forth. So the most direct idea here is to simulate a "car" to conduct trial and error in the path, and finally when it reaches the end point, it is only necessary to keep the path from the starting point to the end point, and the rest of the trial and error paths can be abandoned. At the same time, this is also a side note, before the final path appears, it is necessary to mark all the points that the car has moved. When using Python to process the algorithm, in order to reduce the time complexity and consider increasing the space complexity, a tensor mark_map of (image.shape[0], image.shape[1], 3) is re-opened for saving marker, which serves the breadth-first search algorithm) , and the previous point coordinates of a point. The map here is done using a class, which is convenient for taking into account the entire algorithm.

```
class car_map:
    image: np.array
    car_center: list
    car_length: int
    mark_map: np.array
    bfs_queue: list
    route: list
```

(2) Path planning algorithm:

Use the breadth-first search method for the "car area" that occupies an area, search from the starting point in four directions, up, down, left, and right (just like the car moves in the image), and the search step is set to the pixel length of the car body; that is, only the center of the car is moved. Then, by checking the azimuth occupied by the car area, whether there is a pixel point of 0 to judge whether the car has encountered an obstacle, mark the feasible path without obstacle position, and record the coordinates of the previous point to the point. If it is judged that the car has traveled to the end point, the search is exited, and then the shortest path from the start point to the end point is obtained by backtracking. Here the grayscale pixel value of the starting point is set to (255 + 127) / 2 = 191 , and the relative pixel value of the end point is set to (255 - 127) / 2 = 64 (there is no additional meaning here, it is just used to distinguish). The path obtained by the BFS algorithm here is the shortest path of the entire map. At the beginning, I considered using DFS for traversal, which may be faster than BFS, but the result may not be the shortest path. The specific function code is as follows:

```
    # using BFS algorithm to get the best way to the end line
    def bfs_route(self, rate):
        self.bfs_queue.append(self.car_center)
        self.mark_map_assignment(1, self.car_center[0], self.car_center[1])
        while len(self.bfs_queue) > 0:
            temp_x, temp_y = self.bfs_queue[0][0], self.bfs_queue[0][1]
            self.car_center = [int(temp_x), int(temp_y)]
            # print(self.bfs_queue)
            # print(self.car_center)
            # self.init_car_in_map()
            # self.show_map(5)
            # self.destroy_car_in_map()
            mark = self.mark_map[temp_x, temp_y][0]
            del self.bfs_queue[0]
            if self.move_vertical(rate):  # go upward, car center modified
                if self.mark_map[self.car_center[0], self.car_center[1]][0] == 0:
                    self.mark_map_assignment(mark + 1, temp_x, temp_y)
                    self.bfs_queue.append(deepcopy(self.car_center))
                    if self.judge(64):
                        break
            self.car_center = [temp_x, temp_y]
```

```
            if self.move_vertical(-rate):  # go downward
                if self.mark_map[self.car_center[0], self.car_center[1]][0] == 0:
                    self.mark_map_assignment(mark + 1, temp_x, temp_y)
                    self.bfs_queue.append(deepcopy(self.car_center))
                    if self.judge(64):
                        break
            self.car_center = [temp_x, temp_y]
            if self.move_horizontal(rate):  # turn left
                if self.mark_map[self.car_center[0], self.car_center[1]][0] == 0:
                    self.mark_map_assignment(mark + 1, temp_x, temp_y)
                    self.bfs_queue.append(deepcopy(self.car_center))
                    if self.judge(64):
                        break
            self.car_center = [temp_x, temp_y]
            if self.move_horizontal(-rate):  # turn right
                if self.mark_map[self.car_center[0], self.car_center[1]][0] == 0:
                    self.mark_map_assignment(mark + 1, temp_x, temp_y)
                    self.bfs_queue.append(deepcopy(self.car_center))
                    if self.judge(64):
                        break
            self.car_center = [temp_x, temp_y]
```

In the same way, the function here needs to be judged. When traversing the pixel points of the car body to confirm whether the car encounters an obstacle (or whether it touches the starting point and the end point), you can choose to only detect the surrounding edges of the car; because this detection method is in the current environment. The effect of going down and traversing the entire square pixel is equivalent, that is, it is impossible to have only black pixels inside the car without touching the border of the car, so choosing only traversing the border is better, which can speed up the execution of the algorithm. Reduced time complexity.

(3) Find the final path:

After the algorithm reaches the end point and exits, the coordinates of the end point can be used to complete the extraction of the entire path by finding the previous point corresponding to each point on the path marked on the mark_map. The experimental code is as follows:

```
# after bfs, get the shortest route
def get_route(self):
    self.route = []
    while True:
        temp = deepcopy(self.car_center)
        temp[0], temp[1] = int(temp[0]), int(temp[1])
        self.route.append(temp)
        if self.mark_map[temp[0]][temp[1]][0] == 1:
            break
        self.car_center[0] = int(self.mark_map[temp[0], temp[1]][1])
        self.car_center[1] = int(self.mark_map[temp[0], temp[1]][2])
    self.route.reverse()  # from the beginning to the end
```

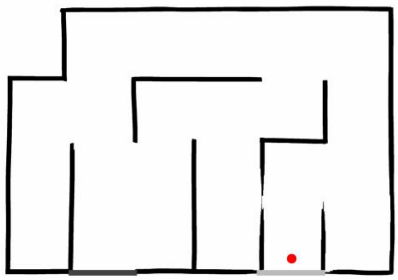
(4) Simplified path:

Finally, when the path is obtained, it can be found that there may be many points on the same line, so here you can choose to remove the redundant points on each line, and only keep the points at the corners to simplify the path. The experimental code is as follows:

```
# cut out the points in one line, only keep the points on the turning
def simplify_route(self):
    original_route = deepcopy(self.route)
    self.route = []
    for i in range(len(original_route)):
        if i == 0:
            self.route.append(original_route[i])
        elif i == len(original_route) - 1:
            self.route.append(original_route[i])
        else:
            if original_route[i][0] == original_route[i - 1][0] \
                    and original_route[i][0] == original_route[i + 1][0]:
                continue
            elif original_route[i][1] == original_route[i - 1][1] \
                    and original_route[i][1] == original_route[i + 1][1]:
                continue
            else:
                self.route.append(original_route[i])
```

(5) Draw path images and dynamic graphs:

So far, the path map has been obtained, and then the code can be used to complete the drawing of the motion path image or motion map. In the process of animation drawing, you can add sampling to make the points on the line more dense and the animation effect is better. The image obtained by the program is shown in the following figure:



(6) Export instruction document:

After the simplified path is obtained, in order to serve the PC's control of the car, the relative angle is used to judge The angle is trimmed; the relative angle refers to the trimming based on the angle after each turn, not the initial angle). According to the simplified path, you can get the direction (left and right) of each turn of the car and the distance of each advance, and then import the instructions into the txt file by means of "angle" + "speed" + "distance".

5.5 Bluetooth communication and gyroscope module

For ports that require UART communication, such as Bluetooth serial ports and gyroscopes, firstly, you need to set the port selection bits PX.2, PX.3, and configure the port's register value as the communication state. According to the given UART1 example, the debugging of the Bluetooth serial port is completed (connected at (P2.2, P2.3), but the gyroscope (connected at P3.2, P3.3) cannot receive any information at all. After modifying all the pins in the UART2 file written by UART1, it still does not work. After studying the underlying principle with reference to the user manual, it is found that the external interrupt program caused by the external device is different. After the modification, the gyroscope finally The real-time body attitude data is returned to the PC. The above is the key code of the UART2 interrupt entry number.

```
  EUSCI_A1->CTLW0 = 0x00C1;
                            // set the baud rate
                            // N = clock/baud rate = 12,000,000/115,200 = 104.1667
  EUSCI_A1->BRW = 104;      // UCBR = baud rate = int(N) = 104

  EUSCI_A1->MCTLW = 0x0000;   // clear first and second modulation stage bit fields
// since TxFifo is empty, we initially disarm interrupts on UCTXIFG, but arm it on OutChar
  P2->SEL0 |= 0x0C;
  P2->SEL1 &= ~0x0C;          // configure P2.3 and P2.2 as primary module function
  NVIC->IP[4] = (NVIC->IP[4]&0xFF00FFFF)|0x00400000; // priority 2
  NVIC->ISER[0] = 0x00020000; // enable interrupt 18 in NVIC
  EUSCI_A1->CTLW0 &= ~0x0001; // enable the USCI module
                            // enable interrupts on receive full
  EUSCI_A1->IE = 0x0001;      // disable interrupts on transmit empty, start, complete
```

(1) Bluetooth module:

This is done via the UART library for Bluetooth and PC debugging. The interface of Bluetooth HC-05 mainly includes TXD (for sending), RXD (for receiving) has high and low level, corresponding to the connection of the development board. For the HC-05 module, we learned that it uses the UART protocol for communication. Based on the knowledge of UART, we reviewed its specific transmission protocol again, and we found that the official UART1 library already exists for the UART serial port, so We try to call this library to debug the bluetooth module, first use the mobile phone to debug. Connect the Bluetooth module to the corresponding pin, and use the Bluetooth serial port APP on the mobile phone to connect and send, and find that it does not receive information normally. So we suspect that the baud rate and master-slave mode of Bluetooth may be set incorrectly, so we found the connected wiring, use the serial debugging assistant on the computer to enter the AT mode, set the Bluetooth mode to slave mode, and set the baud rate to phase matches 38400. After completing these adjustments, test again and find that the serial port can finally receive the data sent by the mobile phone normally. Next, the Bluetooth module was tested for sending data. After a period of debugging, the transceiver function of the Bluetooth module was successfully implemented. After that, combine it with the motion function of the car to realize the function of the bluetooth remote control car.

Corresponding debugging is also required on the computer side. Finally, the receiving and sending of the serial port on the mobile phone and the computer can be completed. But this is still not enough. In order to complete the control and interaction more easily and efficiently, it is necessary to use more sophisticated programs to complete it. This is also the main reason why this group will use Python to complete the serial port programming and UI interface encapsulation.

(2) Gyroscope module:

In the experimental simulation of this group, the realization of PC-guided path planning requires the car to complete accurate straight ahead and turning independently without relying on other sensors, and it is necessary to strictly control the offset of the car's turning angle and

distance. The laboratory equipment cannot provide a host computer to display the current attitude and position of the car in real time. The trolley needs a method that can accurately return the angle, which can reduce the angular deviation of straight travel, and complete the steering with high precision at the same time. In the initial stage of the experiment, the curve of the left and right wheel speed and the number of rotations has been fitted by the Python program, which proves that the adjustment of the car cannot be completed by using the left and right wheel differential returned by the tachometer, so the gyroscope is ready to be used to measure the car's attitude . After the JY-61 gyroscope is installed on the trolley, the current angle and angular velocity returned by the trolley can be obtained in real time. Considering that the angular velocity is a dynamic quantity, it is difficult to measure accurately, and there will be a certain error, so it is only used to return it. The angle is used as the input of the negative feedback for our angle adjustment.

The JY-61 gyroscope uses UART serial port for data transmission, and three packets are sent to the PC side, namely acceleration packet, angular velocity packet and angle packet. The following is part of the key code:

```
//Get  Angle, AngleZ[2]
bool JY61_GetAngle(uint8_t* AngleZ)
{
    uint8_t Buffer[8];  //Cache
    uint8_t Sum = 0;    //Check sum
    uint8_t Receive;    //Accept the data
    uint8_t i;

    RxFIFO2_Init();
    while (1)
    {
        Receive = UART2_InChar();
        if (Receive == DATAHEAD)
        {
            Sum = Receive;  //Initialize check bits
            Receive = UART2_InChar();
            if (Receive == ISANGLESPEED)
            {
                Sum += Receive;
                for (i = 0; i < 8; i++)
                {
                    Buffer[i] = UART2_InChar();
                    Sum += Buffer[i];
                }
                Receive = UART2_InChar();
                if (Receive == Sum)
                {
//                  for(i = 0; i < 8; i++)
//                  {
//                      angular_speed[i] = Buffer[i];
//                  }
                }
            }
            else if (Receive == ISANGLE)
            {
                Sum += Receive;
                for (i = 0; i < 8; i++)
                {
                    Buffer[i] = UART2_InChar();
                    Sum += Buffer[i];
                }
                Receive = UART2_InChar();
                if (Receive == Sum)
                {
                    //printf("%d %d\n", Buffer[5], Buffer[4]);
                    AngleZ[1] = Buffer[4];  //Low-order
                    AngleZ[0] = Buffer[5];  //High-order
                    return 1;
                }
            }
        }
    }

}
```

```
float JY61_ReturnAngle()
{
    uint8_t AngleZHL[2];
    float AngleZ;

    if (JY61_GetAngle(AngleZHL))
    {
        AngleZ = (AngleZHL[0] * 256 + AngleZHL[1]) * 1800 / 32768;
        AngleZ /= 10;
        return AngleZ;
    }

    return 0;
}
```

5.6 Infrared sensing device

In order to complete the obstacle avoidance of the car, the car needs a device that can identify obstacles. After thinking about it for a while, the infrared rangefinder became my final choice. The principle of the infrared rangefinder is to emit a beam of infrared light, and when the infrared light is received, the distance measurement can be completed through the time difference. Returns a two-valued logic based on the magnitude of the distance to the standard distance (within this distance or outside this distance).

```
void Infrared_Init(void){
    P2 -> SEL0 &= ~0x20;
    P2 -> SEL1 &= ~0x20;   //configure P2.5 as GPIO
    P2 -> DIR  &= ~0x20;   //make P2.5 in
}

// if an obstacle is detected, then return a low-level voltage
int Infrared_Get(void){
    return P2 -> IN & 0x20;
}
```

5.7 PC side (Mobile) control program

As mentioned above, in order to facilitate data processing and simplify the control process, here we choose to use the serial library in Python to complete the serial port writing. In order to obtain a stable interactive serial port, the interaction becomes easier and more effective. In this debugging, the main problem lies in the selection of the interface function corresponding to reading the serial port content, because there are many types of functions, so . The writing code of the serial port here (the part of the code is shown here, mainly the overall serial port class and the interface for sending and receiving) is shown in the following figure:

```
class BTSerial:
    ReceiveData = []
    Reception = None
    TransData = None
    serial: serial.Serial

    def __init__(self, port, bps, time_out):
        self.Port = port            # port
        self.Bps = bps              # baud rate
        self.TimeOut = time_out    # time out
        self.IsConnected = False
        self.serial = serial.Serial(port, bps, timeout=time_out)
       if self.serial.is_open:
           print('Initialize the port successfully')
       # try:
       #     self.serial = serial.Serial(port, bps, timeout=time_out)
       #     if self.serial.is_open:
       #         print('Initialize the port successfully')
       # except Exception as e:
       #     print('Failed to initialize the port')

    def BasicInfo(self):
        print('Serial name: ', self.serial.name)
        print('Serial port: ', self.serial.port)
        print('Serial baudrate: ', self.serial.baudrate)
        print('Serial bytesize: ', self.serial.bytesize)
        print('Serial stopbits: ', self.serial.stopbits)
        print('serial parity: ', self.serial.parity)

    def OpenSerial(self):
        self.serial.open()

```

```
    def CloseSerial(self):
        self.serial.close()
        if not self.serial.is_open:
            print('The serial is closed')
        else:
            print('Fail to close')

    def Start(self):
        self.OpenSerial()
        if self.serial.is_open:
            print('Open successfully')
            return True
        else:
            print('Fail to open')
            return False

    def Receive(self, ReceiveFinished):
        while self.serial.is_open:
            try:
                if self.serial.in_waiting:
                    self.Reception = self.serial.read_all().hex()

                    print(self.Reception)
                    ReceiveFinished.emit(self.Reception)    # sending signal
                    self.ReceiveData.append(self.Reception)
                    sleep(0.1)
            except Exception as e:
                pass

    def Transmit(self):
        # print(self.TransData)
        self.serial.write(self.TransData.encode('utf-8'))
        # sleep(0.2)
```

Although the above serial port has all the functions of the serial port debugging assistant, it only has the function of the serial port debugging assistant: the sending of control commands needs to be completed on the console, and the received data cannot be used directly, or printed on the console. . So far, the function of the serial port is still not enough.

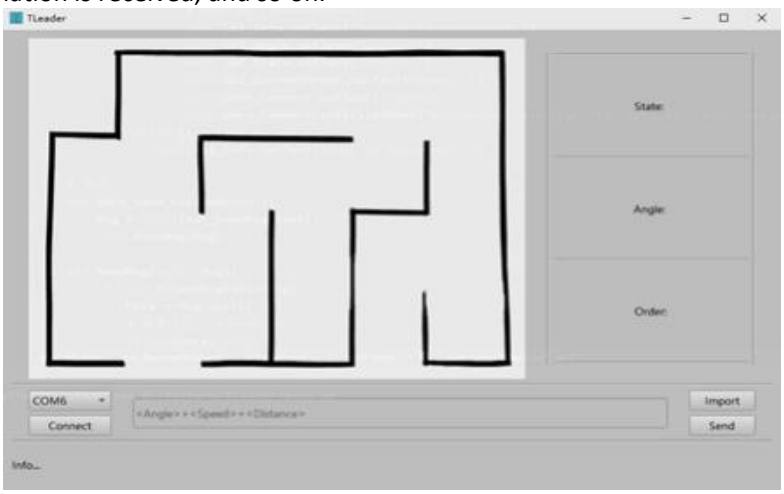
Therefore, in order to improve the overall control process, facilitate debugging, and make better use of multiple threads to complete more three-dimensional operations. I also completed the design of the PC interface through PyQt, and completed the sending and receiving of instructions at the same time through multi-threading. There are two ways to send instructions: that is, directly enter the format of the corresponding instruction in the command line, or import the txt file of the instruction, so that the car will execute in the order of the instructions in the txt.

The format of the command is <angle>+<speed level>+<distance>, the meaning of the command is: the car first rotates the corresponding angle according to the current facing direction as the benchmark, and then rotates at a certain speed (here we set three speed level) to advance a certain distance. The distance here represents the distance that the wheels of the car need to actually rotate. The code of its main body display part is as follows:

```
    def retranslateUi(self, TI_RSLK):
        _translate = QtCore.QCoreApplication.translate
        TI_RSLK.setWindowTitle(_translate("TI_RSLK", "TI-RSLK Car assistant"))
        self.lbl_Map.setText(_translate("TI_RSLK", "Map"))
        self.lbl_State.setText(_translate("TI_RSLK", "State: "))
        self.lbl_Angle.setText(_translate("TI_RSLK", "Angle: "))
        self.lbl_CurrentOrder.setText(_translate("TI_RSLK", "Order:"))
        self.cbx_ComSelect.setWhatsThis(_translate("TI_RSLK", "<html><head/><body><p>
port number</p></body></html>"))
        self.cbx_ComSelect.setItemText(0, _translate("TI_RSLK", "COM6"))
        self.cbx_ComSelect.setItemText(1, _translate("TI_RSLK", "COM7"))
        self.cbx_ComSelect.setItemText(2, _translate("TI_RSLK", "COM3"))
        self.cbx_ComSelect.setItemText(3, _translate("TI_RSLK", "COM4"))
        self.cbx_ComSelect.setItemText(4, _translate("TI_RSLK", "COM5"))
        self.pbtn_Connect.setWhatsThis(_translate("TI_RSLK", "<html><head/><body><p>link
</p></body></html>"))
        self.pbtn_Connect.setText(_translate("TI_RSLK", "Connect"))
        self.ledt_SendMsg.setPlaceholderText(_translate("TI_RSLK", "<Angle>+<Speed>+
<Distance>"))
        self.pbtn_Import.setText(_translate("TI_RSLK", "Import"))
        self.pbtn_Send.setWhatsThis(_translate("TI_RSLK", "<html><head/><body><p>send</p
></body></html>"))
        self.pbtn_Send.setText(_translate("TI_RSLK", "Send"))
        self.lbl_Info.setText(_translate("TI_RSLK", "Info..."))
```

The final interface effect is shown in the following figure. The upper left of the interface is used to display the map; when the PC interacts with the car, the image will be converted into a specific path; the upper right of the interface displays the car's status, angle and current specific command. The bottom of the interface shows the connection to the serial port, the connection or

disconnection, the input of a certain command, the sending and the import, here is a guide to enter a txt file planned by the path algorithm program, and the interior is a series of instruction sets). The bottom is info, which shows the current status, such as whether the connection is successful, whether it is successfully disconnected, whether the command is sent, whether the information is received, and so on.



But only the instruction is not enough, because there is still a certain conversion relationship between the pixel point and the actual distance. In order to solve this problem, this group measured the scale of the maze and pixel conversion, and then wrote the pixel_point_connect function through this ratio to complete the conversion between the distance of the pixel point and the actual distance; then measured the perimeter of the left and right wheels of the car and averaged value, and then convert the actual distance into the number of turns the wheel needs to rotate, which is convenient for output. The experimental code is shown in the following figure:

```python
# convert a certain point from one picture to another with different scale
def pixel_point_convert(firsthand_image, target_image, pixel_posi):
    new_x = int(pixel_posi[0] / firsthand_image.shape[1] * target_image.shape[1])
    new_y = int(pixel_posi[1] / firsthand_image.shape[0] * target_image.shape[0])
    return [new_x, new_y]
```

5.8 Hardware and software control logic

After the software and algorithm parts are written and the hardware devices are debugged separately, they need to be integrated and designed into a whole logic control unit and hardware-software interaction system. Here, the control logic of the software is based on the instructions sent by the PC interface; the overall logic of the hardware is divided into a Bluetooth control module, a gyroscope module and a user interaction module. The control command logic of the hardware part is selected and executed according to the instructions sent by the software. After receiving the information sent by the software, the hardware part will read according to the angle, speed level and distance (number of rotations), and then execute the rotation at a certain angle, and then drive at three speed levels (L, M, H). The specified distance (that is, the corresponding number of laps completed). At the same time, if the infrared part reads that there is an obstacle ahead, it will send a command to the PC through Bluetooth to inform the PC of the appearance of the obstacle and remind the computer to use the path algorithm to replace the path. The main code of the hardware part is as follows:

```c
//Iniitializtion
void HC05_Init()
{
    UART1_Init();
    Motor_Init();
}

//Bluetooth sent uint8_t
void HC05_SendChar(uint8_t Data)
{
    UART1_OutChar(Data);
}

//Bluetooth sends the array
void HC05_SendString(uint8_t* Data)
{
    UART1_OutString(Data);
}

//The computer gives the Angle, speed level, and movement distance
```

```c
void HC05_Motor()
{
    uint8_t AngleH = UART1_InChar();
    //printf("%d", AngleH);
    uint8_t AngleL = UART1_InChar();
    //printf("%d", AngleL);
    uint8_t SpeedLevel = UART1_InChar();
    //printf("%c", SpeedLevel);
    uint8_t Distance = UART1_InChar();   //In 0.1 laps, range 0 to 255
    //printf("%d", Distance);
    int Speed;
    int Angle = (AngleH * 256 + AngleL) * 180 / 32768;
    uint8_t AngleZHL[2];
    float DeltaAngle = 0;

    if (Angle > 44 && Angle < 46)
    {
        Angle = 270;
    }

    //Choose the speed
    switch(SpeedLevel)
    {
    case HIGHSPEEDLEVEL:
        Speed = HIGHSPEED;
        break;
    case MIDSPEEDLEVEL:
        Speed = MIDSPEED;
        break;
    case LOWSPEEDLEVEL:
        Speed = LOWSPEED;
        break;
    default:
        Speed = MIDSPEED;
        break;
    }
    //Send the current absolute angle
    DeltaAngle = JY61_ReturnAngle();
    DeltaAngle =
            (DeltaAngle - INIT_ANGLE) >= 0 ?
                    (DeltaAngle - INIT_ANGLE) : (DeltaAngle - INIT_ANGLE + 360);
    HC05_AngleToBit8(DeltaAngle, AngleZHL);
    HC05_SendChar (AngleZHL[0]);     //High-order
    HC05_SendChar(AngleZHL[1]);     //Low-order

    TARGET_ANGLE = Ang1AndAng2(TARGET_ANGLE, Angle);
    PID_Turn(LOWSPEED, Angle);

    if (Infrared_Get())
    {
        PID_Forward(Speed, Distance * 0.1);
        HC05_SendChar('E');
    }
    else
    {
        HC05_SendChar('C');
    }
}
```

5.9 User interaction circuit

In order to simulate the reminder of the car for a specific situation, some external indicating circuits are needed to process the reminder, so here we choose the buzzer circuit and the LED circuit to remind some specific situations.

(1) Buzzer module:

Since the high level of the development board cannot provide enough current to drive the buzzer, an external analog circuit is adopted here, so that the buzzer can work normally, and can play specific music through program settings. The logic of the buzzer is: when encountering an obstacle, the buzzer will beep directly to give a prompt; when the key point is reached, the buzzer will play the specified music.

(2) The LED circuit:

The LED circuit is completed by the circuit of the car body, and different light-emitting diodes are selected by switching high-level continuously, and then the flashing is completed. Every time an obstacle is encountered, the LED light will flash as a reminder. After writing the corresponding code according to the official tutorial, the three-color LED module also works very well, and can achieve color-changing and flashing according to a certain frequency.

6. Difficulties encountered and failed attempts

6.1 Choice of gyroscope protocol

The original idea was to directly use the off-the-shelf GY-521 module, which uses the I2C protocol for serial communication, but its routine communication always had problems during the debugging process. In the end, when I was looking for a new gyroscope, I saw A JY-61 module that can use both the I2C protocol and the UART protocol, this module can indeed use the simpler UART protocol, and the accuracy is higher than the GY-521, so we decisively turned to choose the more convenient and accurate Higher JY-61 modules.

6.2 Implementation of Path Algorithm

The difficulty here is mainly that it is difficult to figure out how to deal with an image at first: it is almost unrealistic to get the path directly for only the pixels. Later, it was considered that if it is assumed that the car is actually driving in such an image maze, the corresponding design can be completed by using an algorithm. The complexity is briefly analyzed: if the method of judging the frame of the car is used, the complexity is about

[L(image_height)+L(image_width)]*[L(car_length)+L(car_height)]*n=L(image)*L(car_length)*n.

Since the lengths of the image and the car are both fixed, the complexity is almost linear, and the final algorithm can indeed be implemented quickly. This completes the theme of path planning.

6.3 PID selection and adjustment

Based on the parameters returned by the gyroscope, we could theoretically stop the motor as long as the angle is our desired turn reading, but obviously this can't be done. Since the program needs a certain time to run, each time the data is read from the gyroscope is actually a discrete sampling process with an uncertain interval, and the probability of getting it to a certain value in 360 degrees is zero. Therefore, we use a judgment statement, considering the angular velocity, we set the threshold to an angle smaller than the expected angle by a certain angle, when the current angle is less than the threshold, continue to turn, if it is greater than the threshold, then stop the work of the motor. The determination of the threshold is determined by the actual site conditions. But we found that no matter how we adjust the parameters, we can only achieve that the expected angle of the turn is equal to the expected angle, but there will still be a non-negligible variance. The reason is that because of the existence of the angular velocity, the value of the gyroscope read in the previous cycle has not exceeded the threshold, but when the value of the gyroscope is read next time, the increment is the integral of the angular velocity and the interval time, and the accumulated value may be If the angle is larger than the expected angle, it may also be smaller than the expected angle. When the difference reaches a certain range, the attitude of the car has already shifted significantly, which has a great impact on the traveling accuracy. And we use the relative angle adjustment method, and the errors will be stacked in an accumulative manner. Obviously, such accuracy cannot support our car to complete a complex travel route.

Because the error of the relative angle will have the characteristics of accumulation, so we initially adjusted to use the absolute angle instead of the relative angle. Taking the angle after the gyroscope is powered on as the initial angle, all the angles of the subsequent turning are calculated on this basis. In theory, the problem of accumulation of errors caused by relative angles can be eliminated. However, during the actual debugging process, the car failed. According to our expected movement, at the first turn, the car has a problem of non-stop rotation. We guess that if the absolute angle is used, even if the current angle of the car exceeds the threshold, it can still be smaller than the threshold after rotating more than 300 degrees, and the random placement of the final angle of the car still cannot be resolved, and can only be expected to go straight.

Regarding the straight-line PID adjustment, start with a simple (proportional coefficient adjustment method) adjustment. If the effect is good, the integral and differential adjustment does not need to slow down the operation speed of the overall program. Fortunately, the proportional operation method works very well in the field experiment, and even corrects the deviation angle in some cases, so the straight line finally uses a simple P for feedback. It is worth mentioning that due to the influence of the ground seams and the slippage of the tape, in the straight braking stage, the car often rotates a certain angle due to the ground. Test whether the current angle is the same as the initial angle of the straight line. Got very good results.

For the turning of the car, the self-created "small angle precision adjustment" algorithm is used, so that the car can adjust its posture after each turning action, and each adjustment is a small angle adjustment until the current angle and the target angle. to 0.5. within the range. Although this will require adjusting the attitude (shake the body) for each turn, such an algorithm enables the car to start the next step accurately after each turn, and realizes the precise control of the car.

Here is the code for the straight part of our PID module:

```
void PID_Forward(int Speed, float Distance)
{
    int leftDuty, rightDuty;
    leftDuty  = Speed;
    rightDuty = Speed - 100;
    int InitStep = Tachometer_GetStep();    //Initial steps
    float InitAngle = JY61_ReturnAngle();   //The initial angle
    float AngleT;
    float CurAngle = 0;
    Distance -= CIRCLEREPAIR;

    while(Tachometer_GetStep() - InitStep < Distance * 360)
    {
        CurAngle = Ang1SubAng2(JY61_ReturnAngle(), InitAngle);
        if(CurAngle > 1.0 && CurAngle < 90.0)        //Right avertence
        {
            rightDuty +=  FB * CurAngle;
        }
        else if(CurAngle < 359.0 && CurAngle > 270.0)  //Left avertence
        {
            rightDuty  -= FB * (360 - CurAngle);
        }
        Motor_Forward(leftDuty, rightDuty);
    }
    Motor_Stop();
    AngleT = Ang1SubAng2(JY61_ReturnAngle(), InitAngle);
    while ((AngleT > 0.5 && AngleT < 30) || (AngleT < 359.5 && AngleT > 330))
    {
        AngleT = Ang1SubAng2(JY61_ReturnAngle(), InitAngle);
        if (AngleT > 0.5 && AngleT < 30)
        {
            Motor_Left(5000, 5000);
            Clock_Delay1ms(40);
            Motor_Stop();
        }
        if (AngleT < 359.5 && AngleT > 330)
        {
            Motor_Right(5000, 5000);
            Clock_Delay1ms(40);
            Motor_Stop();
        }
    }
}
```

The following is the turn module code:

```
void PID_Turn(int Speed, float Angle)
{
    float InitAngle = JY61_ReturnAngle();
    int leftDuty, rightDuty;
    float AngleS;

    leftDuty  = Speed;
    rightDuty = Speed - 100;

    //Turn right and turn clockwise
    if (Angle > 0 && Angle <= 180)
    {
        Angle -= TURNREPAIR;
        Motor_Right(leftDuty, rightDuty);
        while(Ang1SubAng2(JY61_ReturnAngle(), InitAngle) < Angle)
        {
            Motor_Right(leftDuty, rightDuty);
        }
        Motor_Stop();
        Angle += TURNREPAIR;
        AngleS = Ang1SubAng2(JY61_ReturnAngle(), InitAngle) - Angle;
        while(AngleS > 0.5 || AngleS < -0.5){
            AngleS = Ang1SubAng2(JY61_ReturnAngle(), InitAngle) - Angle;
            if(AngleS > 0.5)
            {
                Motor_Left(5000, 5000);
                Clock_Delay1ms(40);
                Motor_Stop();
            }
            if(AngleS < -0.5)
            {
                Motor_Right(5000, 5000);
                Clock_Delay1ms(40);
                Motor_Stop();
            }
        }
    }
```

```
    else if (Angle > 180 && Angle < 360)
    {
        Angle += TURNREPAIR;
        Motor_Left(leftDuty, rightDuty);
        while(Ang1SubAng2(JY61_ReturnAngle(), InitAngle) > Angle)
        {
            Motor_Left(leftDuty, rightDuty);
        }
        Angle -= TURNREPAIR;
        AngleS = Ang1SubAng2(JY61_ReturnAngle(), InitAngle) - Angle;
        while(AngleS > 0.5 || AngleS < -0.5){
            AngleS = Ang1SubAng2(JY61_ReturnAngle(), InitAngle) - Angle;
            if(AngleS > 0.5)
            {
                Motor_Left(5000, 5000);
                Clock_Delay1ms(40);
                Motor_Stop();
            }
            if(AngleS < -0.5)
            {
                Motor_Right(5000, 5000);
                Clock_Delay1ms(40);
                Motor_Stop();
            }
        }
    }
}

```

7. Innovation point

7.1 Path planning:

In order to simulate the satellite positioning of the car in the real world, this group uses the terminal as the indicator and the car as the simulation of the action terminal. Satellite positioning is done through path planning and trolley movement. Initially, it was envisaged to complete the path planning by completely relying on pixels to distinguish paths, but this operation would be too complicated, and there seems to be no good algorithm. Later, after consulting the information, I found a way to decompose the pixel maze into trees and then perform operations. From this, it comes to mind that you can collide in the maze by simulating the car, so that feasible points and walls can be determined, and then through the queue and breadth algorithm, when the end point is found, the loop is exited. The path obtained by backtracking is the shortest path from the start point to the end point. This simulation algorithm can also simplify the calculation amount by processing a series of details, which can complete the calculation almost instantaneously, and wait for a good result.

7.2 Encountered obstacles to re-plan:

If the infrared part of the front of the car recognizes an obstacle, the car will return a signal to the end, indicating that it has encountered an obstacle, "this road is blocked". This is that the computer will re-plan with the location of the car as the starting point, generate a new path control command, and then forward another route to the car, and the car will turn around and complete the passage along another path that can go together. This is also done in the actual test. It's done, and it works great.

7.3 PC control process:

It is very troublesome to directly interact with the car through a limited USB table and then modify the code to test, so the method of interactive control through Bluetooth is inevitable. After completing the bluetooth control of the car, I found that this is not enough, so the autonomy is not high enough, only the most basic control can be completed, and autonomous regulation cannot be achieved. Therefore, in the end, I made an efficient interface and control algorithm, which can directly complete the rotation and straight travel of the trolley through instructions, which is very convenient for debugging and completes the overall control of the trolley.

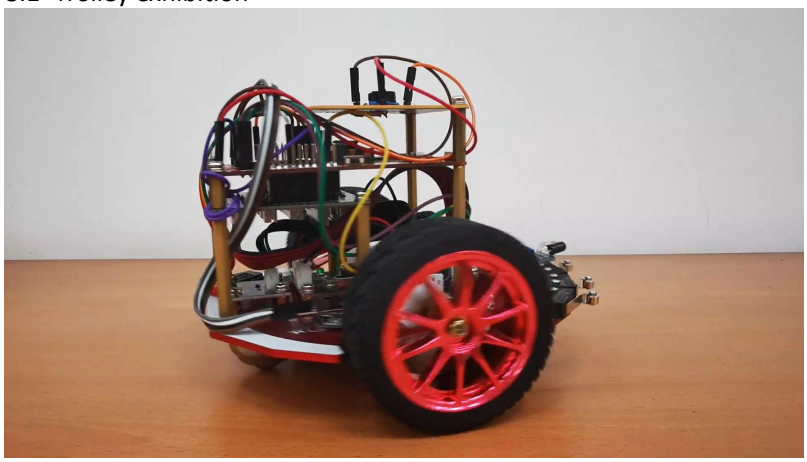
7.4 Trolley control algorithm:

If you directly use the motor library to complete the straight operation of the trolley, it is found that the trolley will directly shift to the right, that is, the left wheel moves faster. After going through a variety of debugging schemes (open loop, closed loop), I found that the detail adjustment that can be rotated at a small angle is the best. Although it takes a certain amount of time to adjust, the final accuracy is very good, and it can be completed on smooth ground. Or make a precise turn on the ground with a slight height difference (but it will have a big impact). After this is done, we thought that in the process of the car going straight, although it is relatively straight as a whole, it will occasionally slip at the final straight position, resulting in a deviation of the relative angle, which is very fatal. Therefore, we also added a small angle correction at the end of each straight run, so as to ensure that the final angles of straight

running and turning are relatively accurate. It turns out that the final implementation of this operation is much better than the previous open-loop operation.

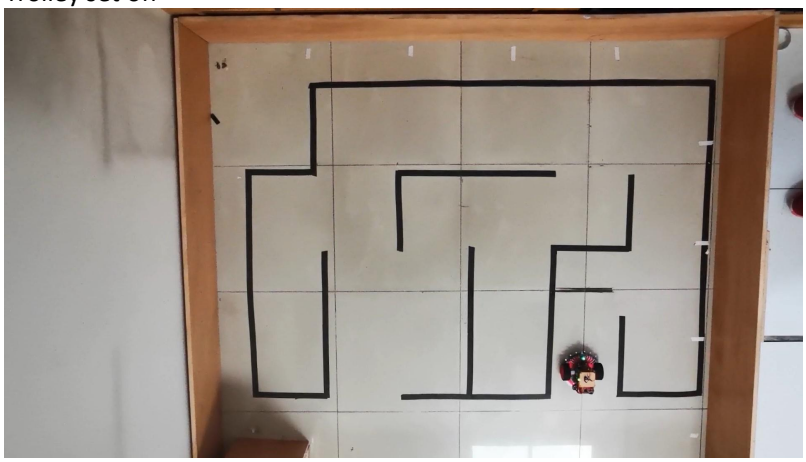
8. Experimental results

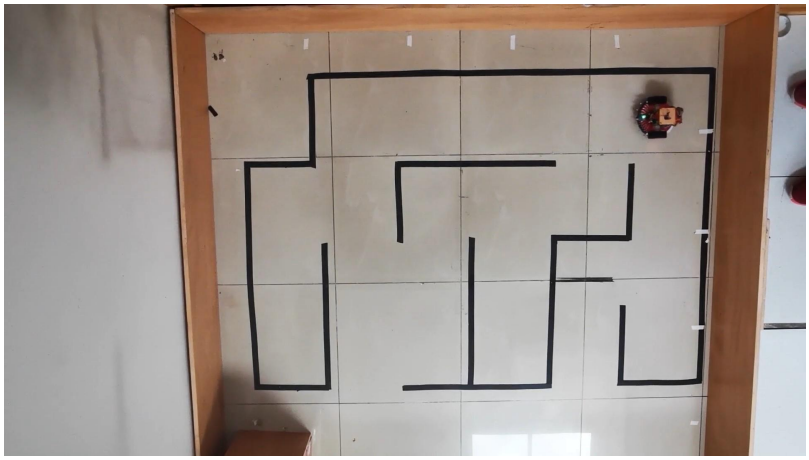
8.1 Trolley exhibition



8.2 Accessibility

Trolley set off

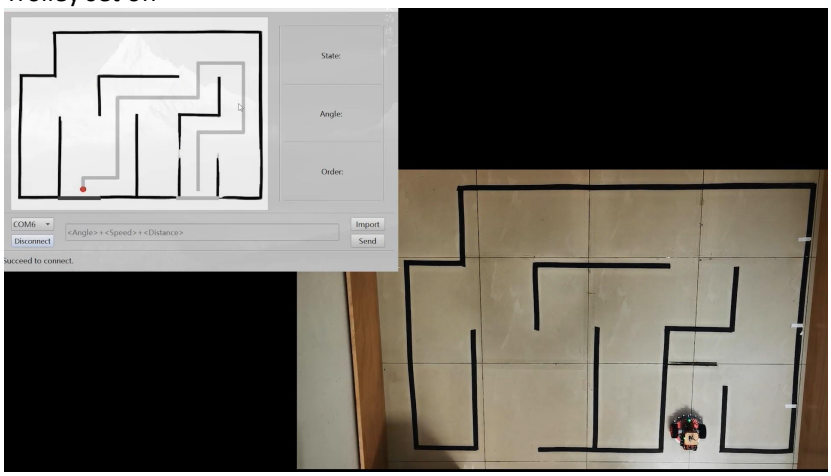


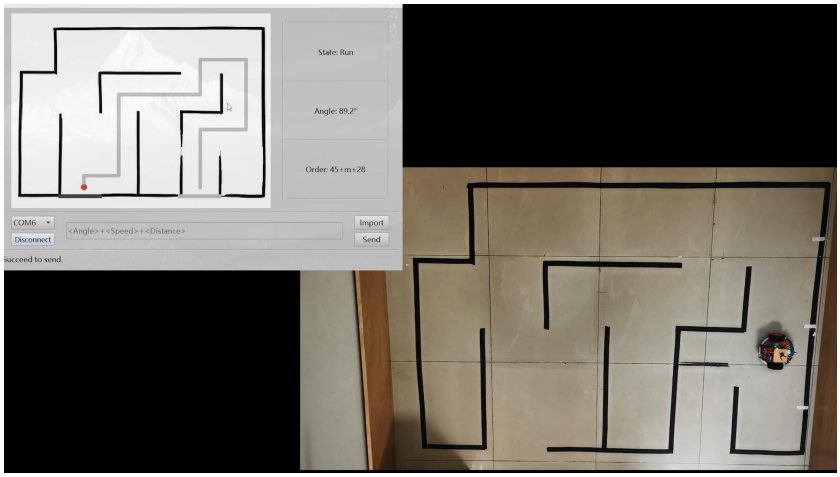


The car reaches the finish line perfectly.

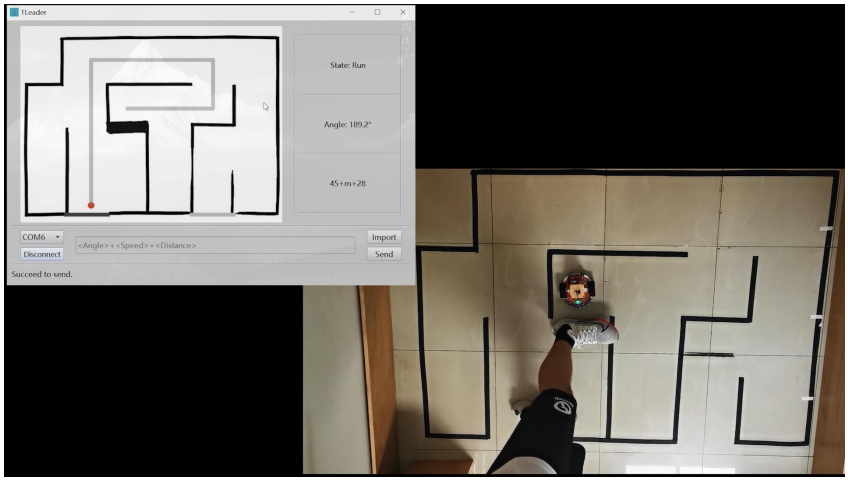
8.3 Handicap presentation

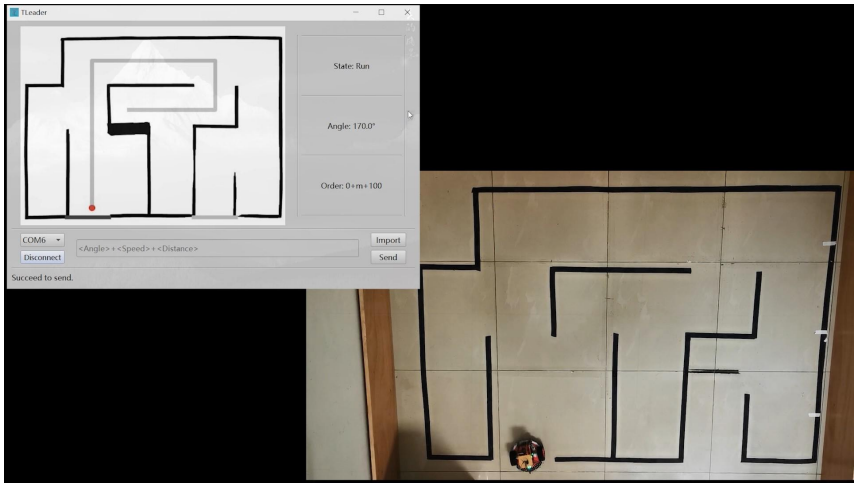
Trolley set off



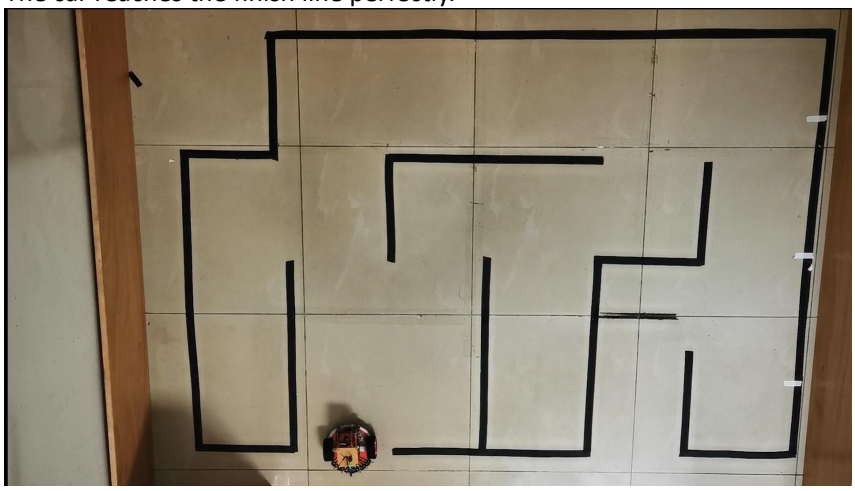


Obstacles appear, re-route planning.





The car reaches the finish line perfectly.



9. Thoughts and conclusions

In this experience, I learned the complete process of the design of the trolley device: from the thinking, discussion, and research before the design; to the less easy entry in the design process, serious thinking and correcting mistakes, and integrating the overall software and hardware logic; It was a very unforgettable process to finally complete the task and shoot the video. During this period, we completed a lot of our own ideas, modified and improved our own logic, and went through a difficult debugging process. I was also confused and annoyed because I couldn't debug it, But in the end I persevered and achieved my goal.