INTELLIGENT MOTION CONTROL

Submitted by
Fu Yalun

Beijing Jiaotong University
School of Electronic Information Engineering

The final year project work was carried out under the 3+1+1 Educational Framework at
The National University of Singapore (Suzhou) Research Institute

May 2021

ABSTRACT

As the first year of L3 level of autonomous driving technology, 2021 is an important point in the development of autonomous driving technology, and the research related to autonomous driving systems has attracted more and more attention. This project, Intelligent Motion Control, can be seen as a simple, small autopilot simulation system.

In this system, the images of the plate and the ball in the cricket system are acquired through the camera and transferred to the computer. After that, using Python programming, the image is binarized and the areas on the plate are distinguished according to the different colors, the plate and the obstacles, and the coordinates of the ball and the circular area on the plate can be obtained. In addition, after obtaining the image of the LEGO maze, the algorithm plans the route of the ball moving from the starting point to the end point and determines a series of target positions of the ball. The ball's position is compared with the set position, and the pulse width adjustment (PWM) and the encoder are used to make the motor reach different rotational speeds.

ACKNOWLEDGMENTS

First of all, I would like to thank my supervisor, Prof. Yung C. Liang from National University of Singapore, for his rigorous and sincere treatment. From the beginning of the topic selection, framework structure, technical details, to the writing process, he has provided meticulous guidance and many useful suggestions for improvement on the difficulties and doubts I encountered.

Secondly, I would like to thank all the teachers and staff of NUS Suzhou Research Institute for their hard work and dedication in providing a well-equipped and comfortable laboratory, which greatly enhanced my enthusiasm and efficiency in my studies. Also, I would like to thank my classmates who came to the institute from various universities. We learned from each other, helped each other, and spent a perfect and unforgettable time together. Of course, we would also like to thank my Alma Mater, Beijing Jiaotong University and NUSRI for providing this valuable graduation design opportunity.

In addition, I would like to thank the shares on the site. Through the sharing, I have gained a better understanding of the research topic. I would also like to thank my defending teachers for their conscientiousness and rigor in promoting my thesis Thank you for your hard work.

Finally, I would like to express my sincere thanks to my class teacher, family, friends and classmates, whose encouragement and support made it possible for me to successfully complete my thesis.

CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Background and significance

Autonomous driving, also known as driverless, computerized driving or wheeled mobile robots, is a cutting-edge technology that relies on computer and artificial intelligence technology to accomplish complete, safe and efficient driving without human manipulation.

In the 21st century, due to the increasing number of car users, road traffic is facing more and more serious problems such as congestion and safety accidents. With the support of Telematics and artificial intelligence, autonomous driving technology can coordinate travel routes and planning times, thus improving travel efficiency to a large extent and reducing energy consumption to a certain extent. Autonomous driving can also help avoid drunk driving, fatigue driving and other safety hazards, reduce driver error and improve safety. Automated driving has also become a focus of research and development in recent years.

Developments in autonomous driving

In 1999, the Naclab-V, a driverless car developed by Carnegie Mellon University, completed its first driverless test, and many laws and regulations for open road experiments with unmanned vehicles were introduced one after another. After development, autonomous driving was promoted in the following years, and in 2009, pictures of the prototype of self-driving cars were revealed, and autonomous driving began to receive attention.

There are six levels of autonomous driving, LO level is completely by the driver to carry out driving operations; L1 level means that the car can assist the driver to complete certain driving tasks under certain circumstances; to L2 level, autonomous driving can complete certain driving tasks, but the driver needs to monitor the changes in the surrounding environment at all times and be ready to take over when encountering dangerous situations, which is the current self-driving technology that many self-driving cars have done L3 level, the driver almost does not need to be ready to take over at all times, the car can complete all the actions independently; L4 and L5 level is fully autonomous driving technology, the car has been completely without the driver's control, the difference is that L4 level can only be fully independent under specific conditions such as highway, while L5 level is established under any conditions.

As a hot spot of much attention in AI applications, autonomous driving technology is also developing like wildfire at this stage. With deep learning, the progress of artificial intelligence

algorithms, driverless is becoming more and more practical and commercialized.

Features

(1) Safety: Through the intelligent operation of digitalization and information technology, self-driving cars can accurately sense the surrounding things and react in time, which greatly reduces the probability of safety accidents. Self-driving cars can detect the condition of human beings, and when human beings are tired of driving, self-driving cars will automatically take over the driving of human beings to avoid accidents.

(2) Convenience: Self-driving cars liberate human hands and can navigate themselves to their destinations without the need for driver control.

(3) Intelligence: After selecting a destination, the self-driving car will use the navigation system to plan the best route to reach the destination, and the car will select a parking space and park automatically.

System Components

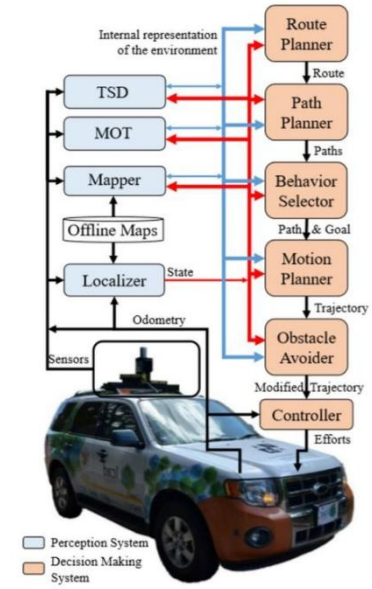


Figure 1.Overview of the typical hierarchical architecture of self-driving cars. TSD denotes Traffic Signalization Detection and MOT, Moving Objects Tracking

Fig 1. Automated Driving System Framework

The control computer autopilot technology consists of positioning and path planning, environment perception, behavior decision and control. That is, the determination of the route is carried out through the collaboration of CPS and computer technology, and the environment is

sensed by sensors, and the control computer handles specific events and overall navigation. In an autonomous vehicle, the master control computer is a device that collects information and makes behavioral decisions, and is the core device for autonomous driving. The control computer controls all the behaviors of the self-driving vehicle. After the information is provided by the sensing device, the computer processes this information and makes the appropriate decisions and actions based on the equipped software algorithms. All the information received by the self-driving car is centralized at the computer, which needs to synthesize and analyze this data before making a judgment.

When driving on the road, the computer's function is to identify the elements of the surrounding environment based on a large database and then make the appropriate countermeasures. The computer can thus issue instructions to speed up, slow down, and steer at the right time, just like a human driver, in order to avoid obstacles, stay in the lane, and identify traffic signals on the road such as speed limit sign instructions and red and green signals. Past autonomous driving technology breakthrough difficulties, an important factor is the slow development of artificial intelligence, people thought in the past is the backwardness of the algorithm, now artificial intelligence deep learning method has actually been proposed much earlier, such as now which Dijkstra's algorithm as a representative of the graph search method in a variety of optimization problems have been more widely used, and this algorithm is the global optimum . But the computer in the past due to the small capacity, slow running speed, this exhaustive algorithm and the need for a large amount of data operations, now the computer itself has been significantly improved performance, the development of big data technology, deep learning efficiency to be reflected. The importance of high-performance computers is not only reflected in the practical application stage, but also in the experimental stage to promote the progress of artificial intelligence technology.

Significance

This project, Intelligent Motion Control, can be seen as a simple, small autopilot simulation system.

Mapper: cameras instead of satellites, radar, etc. as image sensors to obtain road information: the path of the LEGO maze,
Route planner: the computer background planning the driving route: using python to plan the movement route of the ball,
Behavior selection: the behavior of the car to choose (acceleration, deceleration, left turn, right turn, turn around, parking, etc.). In this project, use the ball to simulate the car on the road,
Motion planner: because the ball itself does not have a power unit, only by controlling the board (adjust the tilt angles, the use of gravity to control the direction of motion and speed of the ball). Two motors are mounted on the wheels that control the tilt of the LEGO plates, and the control motor rotation can drive the plates to tilt.
Localizer: GPS real-time positioning of the car. In this project, the camera real-time access to the position of the ball, speed information.

Obstacle avoider: A complete self-driving system should include the function of avoiding obstacles, but this part of the function is currently missing because the driving method used to control the movement of the target in the project is different and cannot slow down or stop autonomously.

CHAPTER 2

STRUCTURE OF THE SYSTEM

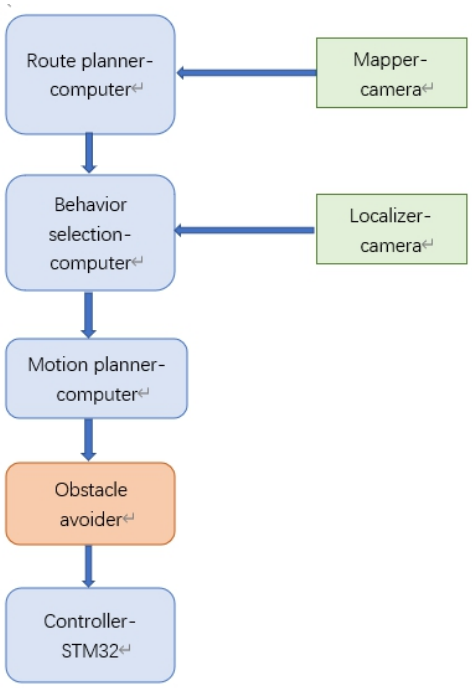


Fig 2. Framework of the System

LEGO Maze:

The ***LEGO Ideas Maze, 21305*** features a customizable maze system, wheel-operated tilt mechanism (incorporating LEGO Technique axles and lift arms), built-in removable ball container with four orange balls and a travel lock. Maze measures over 2" (7 cm) high, 10" (27

cm) wide and 10" (26 cm) deep.

Turn the wheels to control the tilt and avoid the traps to complete each maze challenge. Attach the ball container under the maze and apply the travel lock for safe transit.

Fig 3. The LEGO Maze

The LEGO maze is the basic device of this project, the board surface is equivalent to the road and obstacles in the actual autopilot system, and the orange ball is equivalent to the car driving on the road.

USB Camera:

The camera is placed directly above the LEGO maze and ensures that the entire LEGO set-up is always in the frame. First, the camera needs to be used to capture the panels of the LEGO maze and send them back to the computer. After the movement path of the ball is planned, the camera returns the image in real time to get the position and speed information of the ball to provide the basis for the subsequent judgment.

STM32 Micro-controller and Servo Motors:

Driving the motor rotation depends on the input PWM, different duty cycle PWM corresponds to different rotation direction and angle of the motor. The code to control the pin to output various duty cycle PWM waveforms is stored in the microcontroller, and the corresponding angle is set. When a command is received from the serial port from the computer, the microcontroller can quickly determine and output the corresponding PWM to drive the motor rotation.

CHAPTER 3

IMAGE PROCESSING

Photo preprocessing

RGB color model

The RGB color model is an additive color model in which red, green, and blue light are added together in various ways to reproduce a broad array of colors, which includes almost all colors that the human eye can perceive and is one of the most widely used color systems. The name of the model comes from the initials of the three additive primary colors, red, green, and blue.

The main purpose of the RGB color model is for the sensing, representation, and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography.

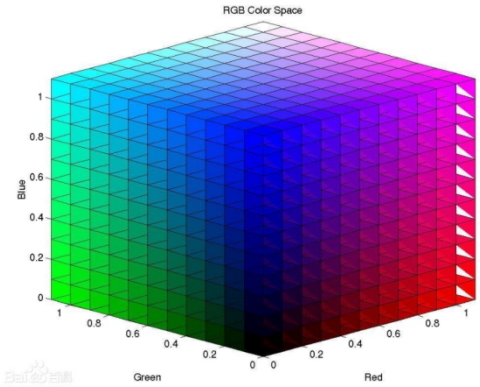


Fig 4. RGB color model

HSV color model

HSV (hue, saturation, value) is an alternative representation of the RGB color model, designed by computer graphics researchers in the 1970s to more closely relate to the way human vision

perceives color properties.In these models, the colors of each hue are arranged in radial slices around the central axis of the neutral color, which ranges from black at the bottom to white at the top.

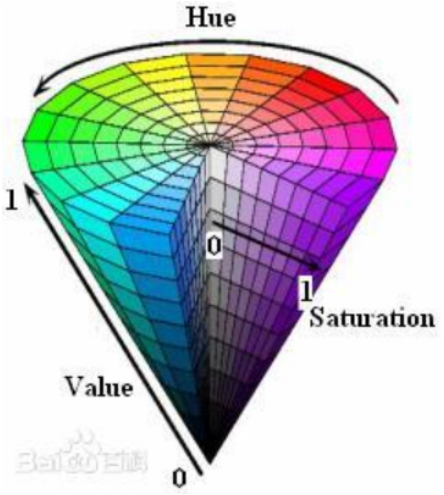


Fig 5. HSV color model

OpenCV usually captures images and videos in 8-bit, unsigned integer, BGR format. In other words, captured images can be considered as 3 matrices, BLUE,RED and GREEN with integer values ranges from 0 to 255.

How BGR image is formed In the above image, each small box represents a pixel of the image. In real images, these pixels are so small that human eye cannot differentiate. Usually, one can think that BGR color space is more suitable for color based segmentation. But HSV color space is the most suitable color space for color based image segmentation. So, in the above application, I have converted the color space of original image of the video from BGR to HSV image.

Therefore, the basic summary is that HSV is better for object detection.

HSV color space is consists of 3 matrices, 'hue', 'saturation' and 'value'. In OpenCV, value range for 'hue', 'saturation' and 'value' are respectively 0-179, 0-255 and 0-255. 'Hue' represents the color, 'saturation' represents the amount to which that respective color is mixed with white and 'value' represents the amount to which that respective color is mixed with black.

Step 1 Distinguish the starting point and end point

In order to plan the path of the ball on the board, it is necessary to distinguish the starting point, end point, pathway and obstacles from the board by different colors of different areas.
Identify the start and end points

The red and green blocks on the board are regarded as the start and end of the maze, and also belong to the passable area. These two area can be extracted by using the threshold of the RGB

values of the two colors.

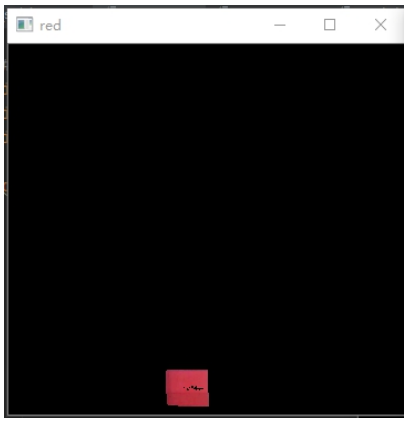


Fig 6. end point

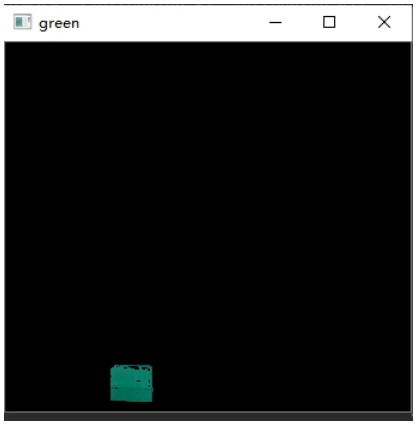


Fig 7. start point

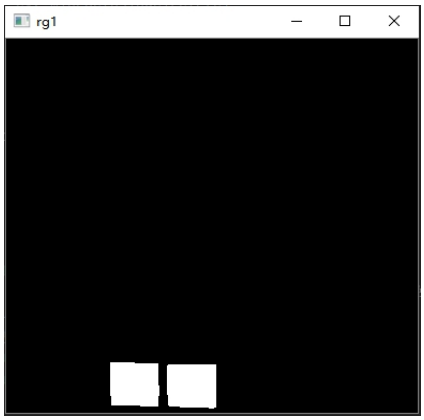


Fig 8. start & end point

Step 2 Identify the pathway

Since the path area is relatively large and the color parameters are more likely to be affected by external factors such as light, it is better to use the HSV color model. The pathway area can be extracted by using the threshold of the HSV values of the two colors.

Fig 9. pathway

‘Add’ start and end points and the pathway part to get the final passable area on the board.

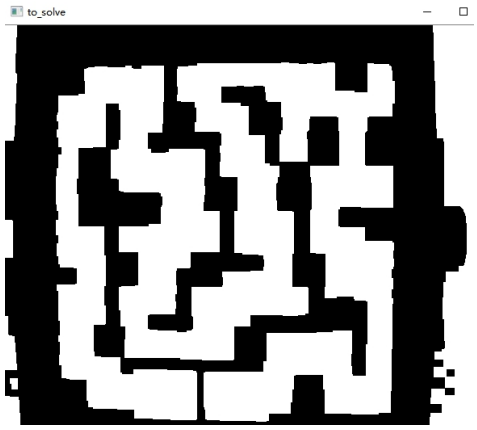


Fig 10. passable area

Step 3 Extract the Center Line

Center line extraction, Skeletonize, is implemented using Skimage library. Skeletonization reduces binary objects to 1 pixel wide representations. This can be useful for feature extraction, and/or representing an object’s topology.

Skeletonize works by making successive passes of the image. On each pass, border pixels are identified and removed on the condition that they do not break the connectivity of the corresponding object.

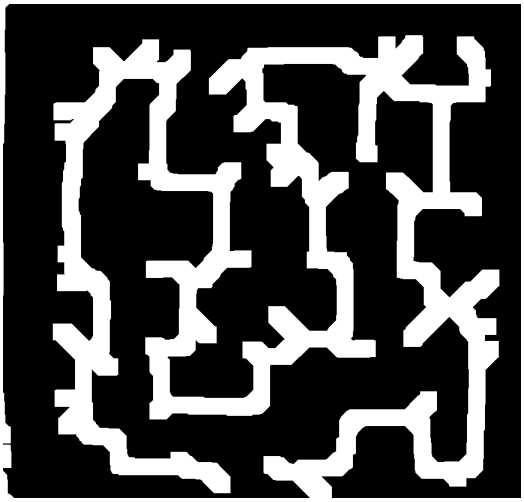


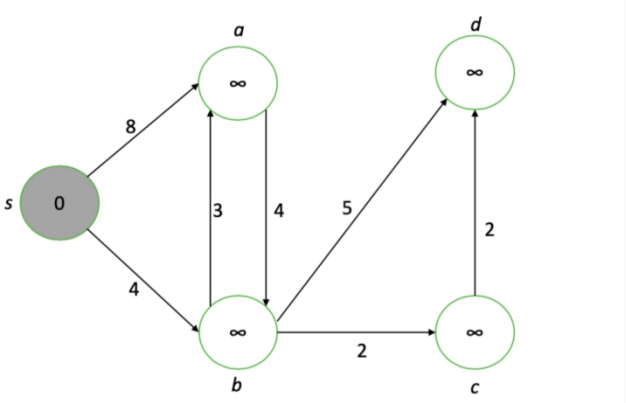
Fig 11. center line of the pathway

Maze solving

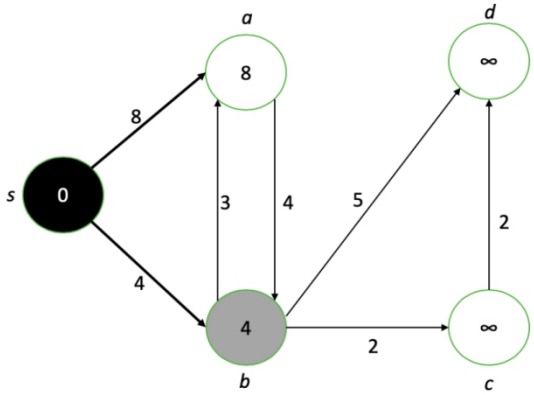
Introduction of Maze-solve Algorithm

Dijkstra's Algorithm is one of the more popular basic graph theory algorithms. It is used to find the shortest path between nodes on a directed graph. We start with a source node and known edge lengths between nodes.

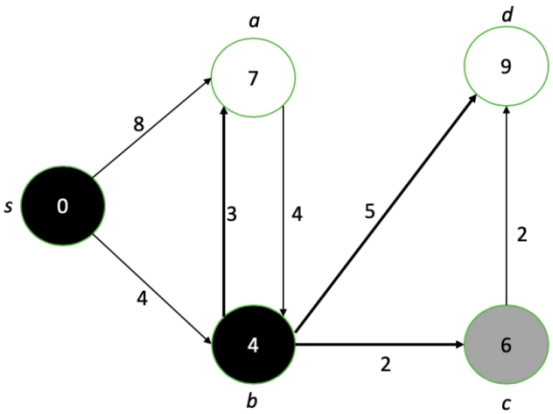
We first assign a distance-from-source value to all the nodes. Node s receives a 0 value because it is the source; the rest receive values of infinity to start.



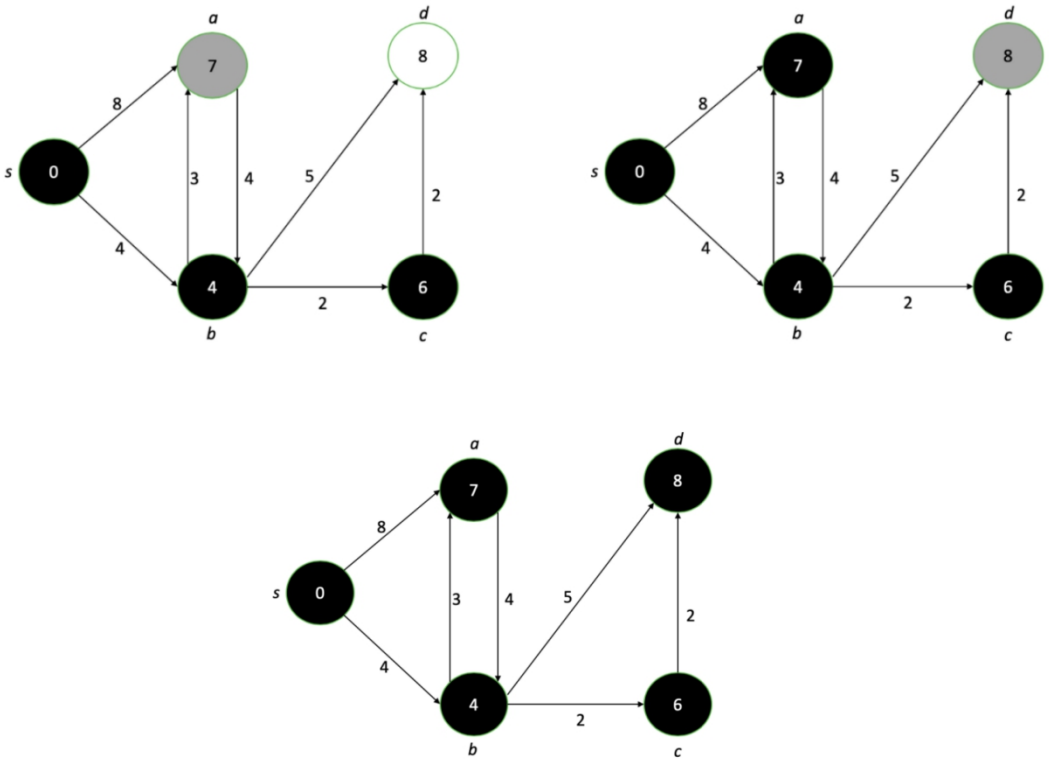
Our node of interest is the smallest-value unprocessed node (shown in grey), which is s. First, we "relax" each adjacent vertex to our node of interest, updating their values to the minimum of their current value or the node of interest's value plus the connecting edge length.



Node s is now finalized (black) and its neighbors a and b have taken on new values. The new node of interest is b, so we repeat the process of “relaxing” b’s adjacent nodes and finalizing the shortest-path value of b.



After going through each node, we eventually end up with a graph showing the shortest path length from the source to every node.



Our final diagram after running Dijkstra's algorithm. The numbers within each node represent the shortest possible distance from the source node.

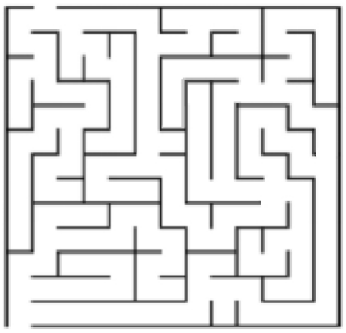


Fig 12. maze for testing

We can consider a maze as a matrix of pixels. Each pixel (for simplicity's sake) has an RGB value of (0,0,0) for black or (255,255,255) for white. Our goal is to create a shortest path which starts in the white and does not cross into the black boundaries. To represent this goal we can treat each pixel as a node and draw edges between neighboring pixels with edge lengths based on RGB value differences. We will use the Euclidean squared distance formula and add 0.1 to ensure no 0-distance path lengths (a requirement for Dijkstra's algorithm):

$$distance = 0.1 + (R_2 - R_1)^2 + (G_2 - G_1)^2 + (B_2 - B_1)^2$$

This formula makes the distance of crossing through the maze boundary prohibitively large. As we can see, the shortest path from source to destination will clearly be around the barrier, not through it.

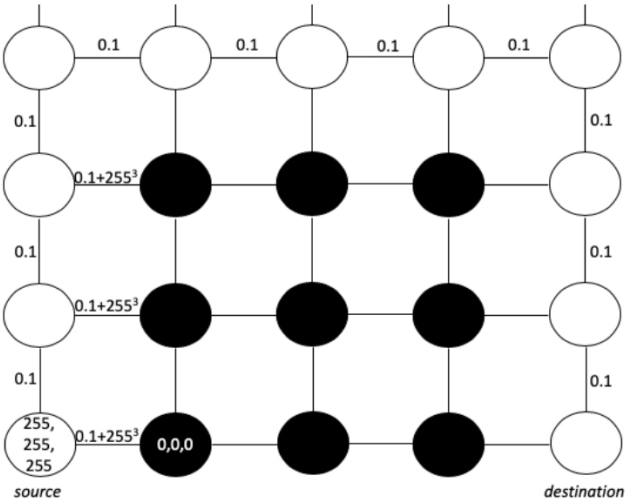


Fig 13. distance between pixels

Test results of maze solving algorithm:

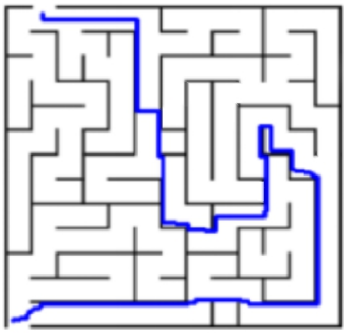


Fig 14. testing result

Solve result

To make sure the planned route not be close to walls or traps, use the maze after extracting the center line as the solving object.

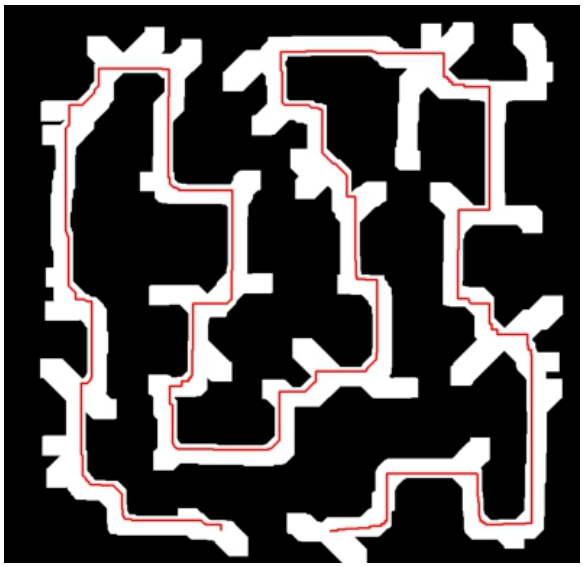


Fig 15. solve result

The planned path is composed of a number of points (4500 or so) in python. A certain number of points are extracted at equal intervals (the extraction interval is about 45) as the target points for the ball movement and stored in the list in order.

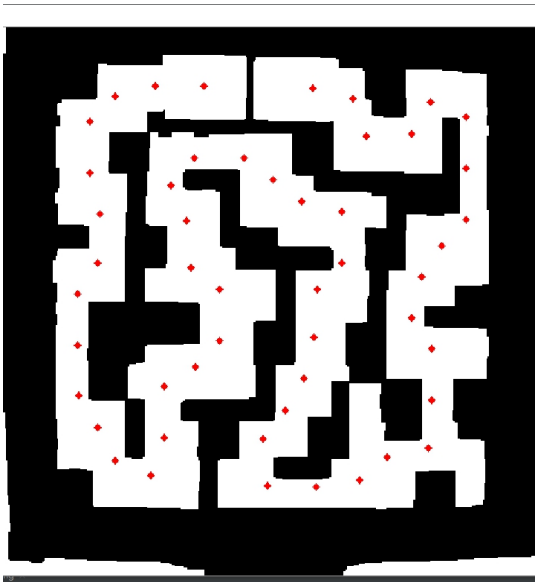


Fig 16. Set target points

CHAPTER 4

BALL TRACKING

Identify the Ball

First, use the color threshold of the ball to extract the ball only, and convert the RGB image to a grayscale image.

Use the edge detection function in the Opencv library to find the contour of the ball. The principle of image edge detection is to detect all the points in the image with large changes in gray value, and these points are connected to form several lines, and these lines can be called the edges of the image.

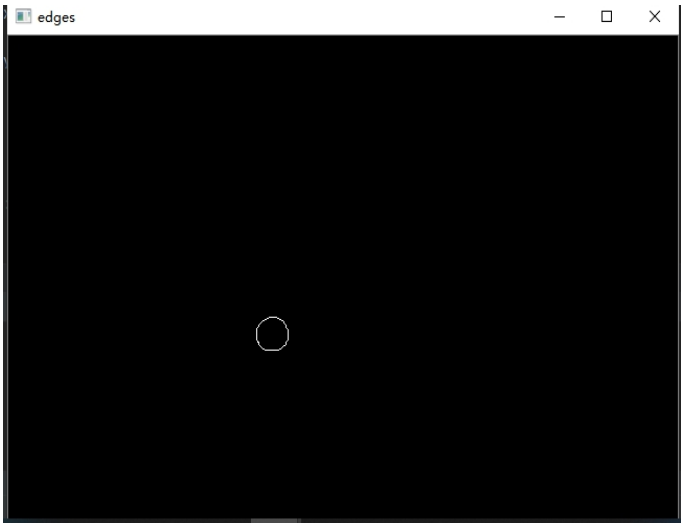


Fig 17. edge of the ball

Then, use the Hough transform to find out the center point coordinates and radius of the ball, and draw the edge and center of the ball.

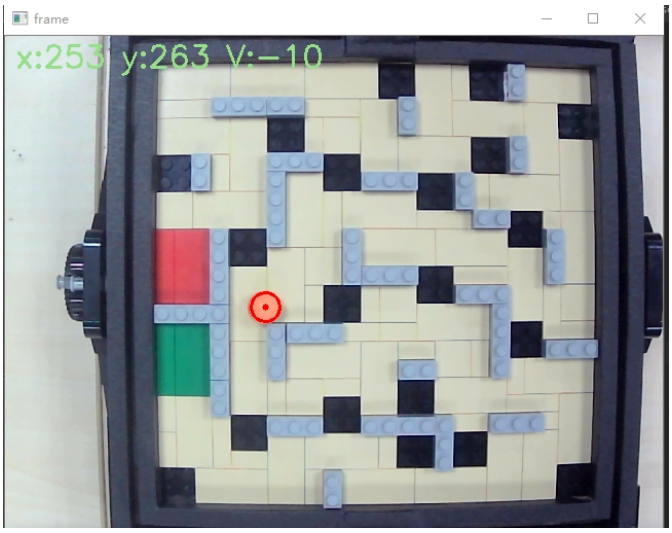


Fig 18. track the ball

Calculate the velocity(obtained from the difference between the two coordinate values) of the ball's motion, to achieve the tracking of the ball.

```
P: 199 89 V: -4 -7
P: 200 84 V: 1 -5
P: 205 85 V: 5 1
P: 219 89 V: 14 4
P: 236 95 V: 17 6
P: 258 103 V: 22 8
P: 278 115 V: 20 12
P: 298 127 V: 20 12
P: 314 137 V: 16 10
P: 324 146 V: 10 9
P: 326 150 V: 2 4
```

Fig 19. print the coordinates and velocity

Ball Tracking Error Analysis

Place small balls at equal distances (3cm) on the edge of the frame and record the coordinates of the small balls. The y-coordinate remains the same, and the difference in the x-coordinate value is approximately equal to 58.

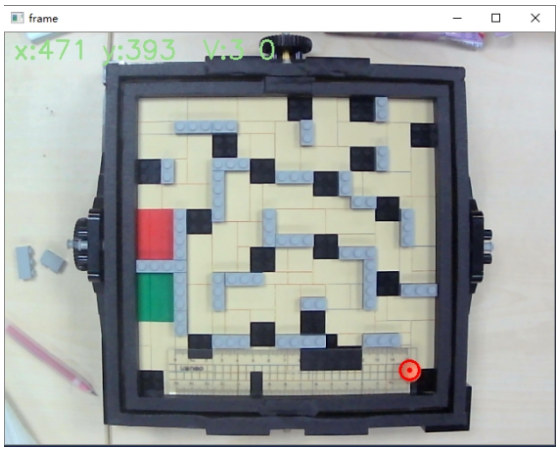


Fig 20. edge of the frame

Place balls at equal distances in the middle area of the frame and record the coordinates of the balls. The difference between the coordinate values of the two points in the middle of the frame is obviously greater than the difference between the coordinate values of the two points closer to the edge.

The camera has a slight barrel distortion. The distortion in the center area of the image is more obvious than the edge area. Since the points on the planned pathway correspond to the actual position of the ball, I think this deviation should not affect the control of the ball along the pathway.

Description of Mechanical Unit

This system from bottom to top are:

1. 40x40 cm wooden board (including two wooden blocks padded motors):

A wooden board large enough to hold the LEGO maze and the two motors, so that the maze board surface tilts with it when the servos turn. At the same time, fixing the LEGO device is convenient for the camera to position the ball.

2. LEGO maze and motors:

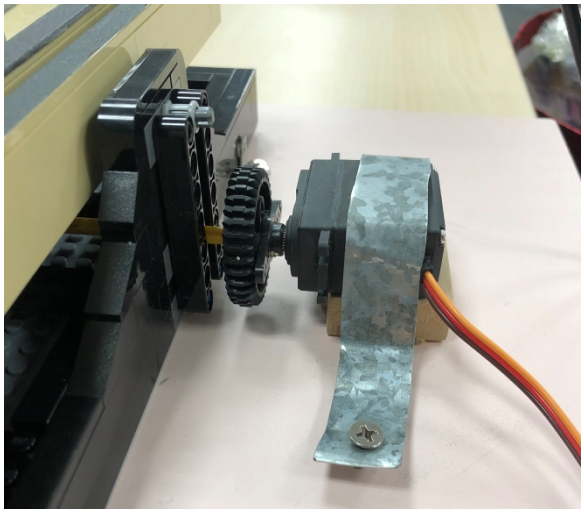


Fig 21. Connection between LEGO set and motor

Both the LEGO maze and the motor are the core devices of the system. When both devices are fixed on the wooden board and the motor is placed at the right height with wooden blocks, the motor can be installed on the two wheels of the LEGO maze.

3. Stand and camera:

Obtaining the position and velocity information of the ball in real time is essential for the proper operation of the system. Therefore, it is necessary to keep the camera directly above the LEGO device at all times during the system operation. It is troublesome and time consuming to customize the metal bracket, so here I used a common cell phone stand (the base can be fixed on the edge of the table or on the baffle in front of the workstation) to fix the camera.

4. Parts:

Several self-tapping screws: used to fix parts, such as L-shaped brackets, etc.

8 L-shaped brackets: 8 L-shaped brackets are fixed to the board with screws, allowing the LEGO maze to be held in place.

2 pieces of iron sheets: Cut the iron piece into a suitable shape, think of a band like across the motor, and fix the ends to the board with nails so that the motor can stay in place while rotating.

6 DuPont wires: Used to connect the motors and the STM32 micro-controller.

MG996R Servo Motor

Stepper motor is an open-loop control element stepper motor that converts electrical pulse signals into angular or linear displacements. Under non-overload conditions, the motor's speed and stop position depend only on the frequency of the pulse signal and the number of pulses, and are not affected by changes in load. When the stepper driver receives a pulse signal, it drives the stepper motor to rotate by a fixed angle in a set direction, called "step angle", and its rotation is run step by step at a fixed angle. The number of pulses can be controlled to control the amount of angular displacement, so as to achieve accurate positioning, while the pulse frequency can be controlled to control the speed and acceleration of motor rotation, so as to achieve high speed

Servo motor, also known as actuator motor. The rotor inside the servo motor is a permanent magnet, and the three-phase U/V/W electricity controlled by the driver forms an electromagnetic field under the action of which the rotor rotates. In automatic control systems, it is used as an actuating element to convert the received electrical signal into an angular displacement or angular velocity output on the motor shaft. The rotor inside the servo motor is a permanent magnet, and the three-phase U/V/W electricity controlled by the drive forms an electromagnetic field under which the rotor rotates, while the encoder that comes with the motor feeds signals to the drive, which adjusts the angle of rotor rotation based on the feedback value compared to the target value. The accuracy of the servo motor is determined by the accuracy of the encoder (number of lines) that is to say, the servo motor itself has the function of sending out pulses, it will send out the corresponding number of pulses for every angle of rotation, so that the servo drive and the servo motor encoder pulses form an echo, so it is closed-loop control, and the stepper motor is open-loop control.

The difference between stepper motor and servo motor mainly lies in:1. The control accuracy is different. Stepper motor phase and the number of beats, the higher its accuracy, servo motor from the block with the encoder, the more the encoder scale, the higher the accuracy. 2. low-frequency characteristics are different; stepper motor at low speeds prone to low-frequency vibration phenomenon, when it works at low speeds generally use damping technology or subdivision technology to overcome the phenomenon of low-frequency vibration, servo motors run very smoothly, even at low speeds will not vibration phenomenon even at low speed. Therefore, here I choose servo motor.

Fig 22. Motor MG996R

The **MG996R** is a metal gear servo motor with a maximum stall torque of 11 kg/cm. Like other RC servos the motor rotates from 0 to 180 degree based on the duty cycle of the PWM wave supplied to its signal pin.

MG996R Servo Motor Features

Operating Voltage	+5V
Current	2.5A (6V)
Stall Torque	9.4 kg/cm (at 4.8V)
Maximum Stall Torque	11 kg/cm (6V)
Operating speed	0.17 s/60°
Gear Type	Metal
Rotation	0°-180°
Weight of motor	55gm

Table 1. Features of motor

As we know there are three wires coming out of this motor. The description of the same is given on top of this page. To make this motor rotate, we have to power the motor with +5V using the Red and Brown wire and send PWM signals to the Orange colour wire. Hence we need something that could generate PWM signals to make this motor work, this something could be anything like a 555 Timer or other Microcontroller platforms like Arduino, PIC, ARM or even a microprocessor like Raspberry Pi. Now, how to control the direction of the motor? To understand that let us a look at the picture given in the datasheet.

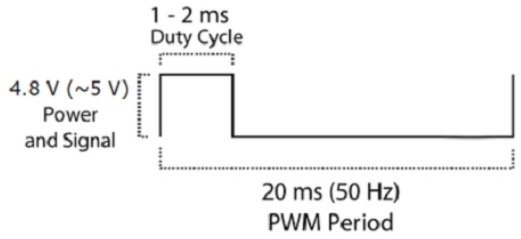


Fig 23. PWM Period

From the picture we can understand that the PWM signal produced should have a frequency of 50Hz that is the PWM period should be 20ms. Out of which the On-Time can vary from 1ms to 2ms. So when the on-time is 1ms the motor will be in 0° and when 1.5ms the motor will be 90°, similarly when it is 2ms it will be 180°. So, by varying the on-time from 1ms to 2ms the motor

can be controlled from 0° to 180°.

Connection

A **Python+STM32** approach was taken to control the servo because an STM32F407 micro-controller was already available, although it is possible to control the servo directly with a micro-controller that supports python compilation, such as Raspberry Pi.

Basically, I use the micro-controller (to output PWM) to control the rotation angle of the motors. Computer determines when the MCU should output the PWM and through serial communication, connecting the computer and MCU, the motors can rotate properly.

A. Motor control via STM32

PWM, short for Pulse Width Modulation, is the digital encoding of analog signal levels by modulating the width of a series of pulses to produce the desired waveform (including shape and amplitude), i.e. by adjusting the change in duty cycle to regulate the change in signal, energy, etc. The duty cycle is the percentage of time the signal is at a high level in a cycle, e.g. 50% for a square wave.

Stm32 steering gear control, the angle range is 0-180 degrees, through the key to change the PWM duty cycle, and then control the steering gear rotation angle. The rotation angle can also be set as needed. All STM32 timers except TIM6 and 7 can be used to generate PWM outputs. The advanced timers TIM1 and TIM8 can generate up to 7 PWM outputs simultaneously.

B. Communication between python and STM32

The communication between Python and STM32 microcontroller is implemented through serial communication (Pyserial) in Python.

Serial communication is a communication method that transfers data by bit between peripherals and computers through data signal lines, ground lines, control lines, etc. This communication method uses fewer data lines and can save communication costs in long-distance communication, but its transmission speed is lower than parallel transmission. Serial is a very common device communication protocol on computers. Pyserial module encapsulates python's access to the serial port, providing a unified interface for multi-platform use. The following figure shows the results of the serial communication test.

```python
import serial

# 连接串口
serial = serial.Serial('COM2', 115200, timeout=2)
if serial.isOpen():
    print('串口已打开')
```

Fig 24. Test result of Pyserial

Control the Movement

The points are taken at equal intervals from the planned route and obtained set target position points as in Fig. 25. The ball moves along the points. Take the coordinates of the center of the sphere as the position of the sphere at this time.

The position of the ball is compared with the horizontal and vertical coordinates of the target position point, and when the coordinate difference is less than a certain value (usually set to the diameter of the ball), it is judged that the ball has passed this target point. At this point, the target position point will become the next point in the list. When the difference between the x coordinate of the ball position and the target position point exceeds this set value, adjust the rotation angle of the motor controlling the horizontal axis. When the difference between the ball position and the y coordinate of the target position point exceeds this set value, adjust the rotation angle of the motor controlling the longitudinal axis.

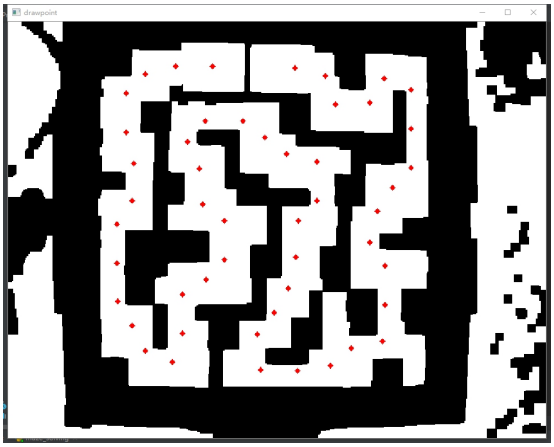


Fig 25. set target points

CHAPTER 5

Conclusion and Future Work

Conclusion

In this system, the images of the LEGO maze and the ball are acquired through a camera and transmitted to a computer. After that, the images are binarized using Python programming to distinguish the areas on the plates based on different colors, plates and obstacles, and to obtain the coordinates of the maze start and end points. Furthermore, after obtaining the image of the LEGO maze, the algorithm plans the movement route of the ball from the starting point to the end point and determines a series of target positions for the ball. The position of the ball is compared with the set position, and the pulse width regulation (PWM) and microcontroller are used to turn the motor, control the tilt direction of the board, and control the movement of the ball along the route.

Improvement

(1) There is still no way for the ball to bypass all the traps, so in order for the ball to move smoothly from the start to the end, it still needs to block the traps.

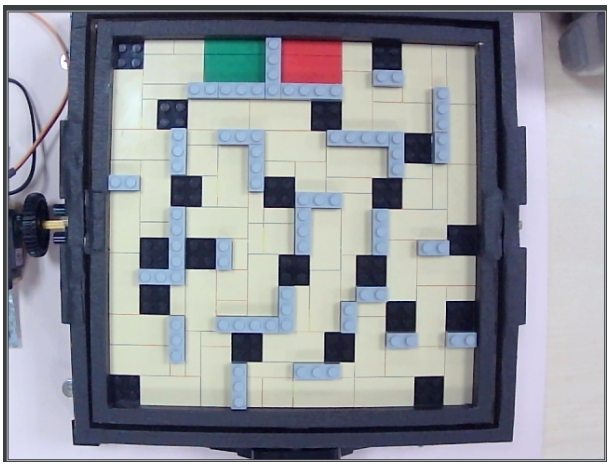


Fig 26. The initial maze

(2) We can try to build different mazes to compare the performance of the system in different mazes. For example, when there are many tight turns, can the ball pass smoothly, at what speed and in what time?

REFERENCE

[1] Huijuan Wang, Yuan Yu and Quanbo Yuan, "Application of Dijkstra algorithm in robot path-planning," 2011 Second International Conference on Mechanic Automation and Control Engineering, 2011, pp. 1067-1069, doi: 10.1109/MACE.2011.5987118.
[2] M. Noto and H. Sato, "A method for the shortest path search by extended Dijkstra algorithm," Smc 2000 conference proceedings. 2000 ieee international conference on systems, man and cybernetics. 'cybernetics evolving to systems, humans, organizations, and their complex interactions' (cat. no.0, 2000, pp. 2316-2320 vol.3, doi: 10.1109/ICSMC.2000.886462.
[3] Druzhkov, P.N., Erukhimov, V.L., Zolotykh, N.Y. et al. New object detection features in the OpenCV library. Pattern Recognit. Image Anal. 21, 384 (2011). https://doi.org/10.1134/S1054661811020271
[4] G. Chandan, A. Jain, H. Jain and Mohana, "Real Time Object Detection and Tracking Using Deep Learning and OpenCV," 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), 2018, pp. 1305-1308, doi: 10.1109/ICIRCA.2018.8597266.
[5] Cui Lin, Zhu Lei, Bai Lu. STM32-based parameter tunable PWM waveform generator design[J]. Information Communication, 2018, 000(001):129-131.
[6] Zhao Yanhua, Shao Hongxiang. Vision-based cricket ball control system research[J].

Automation Technology and Applications, 2011, 030(010):12-15.
[7] Zhang He-Yan, Shao Tian-Yang, Wang Qian, et al. An STM32-based cricket ball control system:, CN111510581A[P]. 2020.
[8] Wang Y. Comparison of stepper motors and servo motors [J]. Small and medium-sized enterprise management and technology, 2010.
[9] Xiao Medallion. Principles and considerations of servo motor selection [J]. Digital World, 2018.
[10] Lian Zozheng, Wang Hazhen. Experimental design of PWM output based on STM32[J]. Experimental Technology and Management, 2017, 34(008):137-140.

APPENDIX A Code for solving mazes

```
class Vertex:
    def __init__(self, x_coord, y_coord):
        self.x = x_coord
        self.y = y_coord
        self.d = float('inf')   # distance from source
        self.parent_x = None
        self.parent_y = None
        self.processed = False
        self.index_in_queue = None


# Return neighbor directly above, below, right, and left
def get_neighbors(mat, r, c):
    shape = mat.shape
    neighbors = []
    # ensure neighbors are within image boundaries
    if r > 0 and not mat[r - 1][c].processed:
        neighbors.append(mat[r - 1][c])
    if r < shape[0] - 1 and not mat[r + 1][c].processed:
        neighbors.append(mat[r + 1][c])
    if c > 0 and not mat[r][c - 1].processed:
        neighbors.append(mat[r][c - 1])
    if c < shape[1] - 1 and not mat[r][c + 1].processed:
        neighbors.append(mat[r][c + 1])
    return neighbors
```

```
def bubble_up(queue, index):
    if index <= 0:
        return queue
    p_index = (index - 1) // 2
    if queue[index].d < queue[p_index].d:
        queue[index], queue[p_index] = queue[p_index], queue[index]
        queue[index].index_in_queue = index
        queue[p_index].index_in_queue = p_index
        quque = bubble_up(queue, p_index)
    return queue


def bubble_down(queue, index):
    length = len(queue)
    lc_index = 2 * index + 1
    rc_index = lc_index + 1
    if lc_index >= length:
        return queue
    if lc_index < length and rc_index >= length:   # just left child
        if queue[index].d > queue[lc_index].d:
            queue[index], queue[lc_index] = queue[lc_index], queue[index]
            queue[index].index_in_queue = index
            queue[lc_index].index_in_queue = lc_index
            queue = bubble_down(queue, lc_index)
    else:
        small = lc_index
        if queue[lc_index].d > queue[rc_index].d:
            small = rc_index
        if queue[small].d < queue[index].d:
            queue[index], queue[small] = queue[small], queue[index]
            queue[index].index_in_queue = index
            queue[small].index_in_queue = small
            queue = bubble_down(queue, small)
    return queue


def get_distance(img, u, v):
    return 0.1 + (float(img[v][0]) - float(img[u][0])) ** 2 + (float(img[v][1]) - float(img[u][1])) ** 2 + (
                    float(img[v][2]) - float(img[u][2])) ** 2


def drawPath(img, path, thickness=2):
    '''path is a list of (x,y) tuples'''
    x0, y0 = path[0]
```

```
    for vertex in path[1:]:
        x1, y1 = vertex
        cv2.line(img, (x0, y0), (x1, y1), (255, 0, 0), thickness)
        x0, y0 = vertex


def find_shortest_path(img, src, dst):
    pq = []  # min-heap priority queue
    source_x = src[0]
    source_y = src[1]
    dest_x = dst[0]
    dest_y = dst[1]
    imagerows, imagecols = img.shape[0], img.shape[1]
    matrix = np.full((imagerows, imagecols), None)  # access by matrix[row][col]
    for r in range(imagerows):
        for c in range(imagecols):
            matrix[r][c] = Vertex(c, r)
            matrix[r][c].index_in_queue = len(pq)
            pq.append(matrix[r][c])
    matrix[source_y][source_x].d = 0
    pq = bubble_up(pq, matrix[source_y][source_x].index_in_queue)
    while len(pq) > 0:
        u = pq[0]
        u.processed = True
        pq[0] = pq[-1]
        pq[0].index_in_queue = 0
        pq.pop()
        pq = bubble_down(pq, 0)
        neighbors = get_neighbors(matrix, u.y, u.x)
        for v in neighbors:
            dist = get_distance(img, (u.y, u.x), (v.y, v.x))
            if u.d + dist < v.d:
                v.d = u.d + dist
                v.parent_x = u.x
                v.parent_y = u.y
                idx = v.index_in_queue
                pq = bubble_down(pq, idx)
                pq = bubble_up(pq, idx)

    path = []
    iter_v = matrix[dest_y][dest_x]
    path.append((dest_x, dest_y))
    while (iter_v.y != source_y or iter_v.x != source_x):
        path.append((iter_v.x, iter_v.y))
```

```
            iter_v = matrix[iter_v.parent_y][iter_v.parent_x]

    path.append((source_x, source_y))
    return path


# img = cv2.imread('C:\\Users\Administrator\Downloads\Maze-master\maze.png') # read image
img = cv2.imread('E:\\maze3.jpg') # input image
cv2.imwrite('maze-initial.png', img)
p = find_shortest_path(img, (150, 40), (300, 30)) # Manually set the starting and ending points.
drawPath(img, p)
cv2.imwrite('maze-solution.png', img)
plt.figure(figsize=(7, 7))
plt.imshow(img)   # show the image on the screen
plt.show()
```

APPENDIX B Code for ball tracking

```
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
#ret = cap.set(4, 400)
#ret = cap.set(4, 400)
font = cv2.FONT_HERSHEY_SIMPLEX
kernel = np.ones((5, 5), np.uint8)
x_before = 0
y_before = 0
lower_ball = np.array([0, 100, 200])
upper_ball = np.array([40, 200, 255])
data_x = b'rreset\r\n'
data_y = b'rreset\r\n'
# 连接串口
ser = serial.Serial('COM3', 115200, timeout=2)


if cap.isOpened() is True:
    while (True):
        ret, frame = cap.read()
        # Start time
```

```
        start = time.time()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
        #   消除噪声
        mask = cv2.inRange(hsv, lower_ball, upper_ball)
        opening = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
        # bila = cv2.bilateralFilter(opening, 10, 200, 200)
        edges = cv2.Canny(opening, 50, 100)
        #  识别圆形
        circles = cv2.HoughCircles(
            edges, cv2.HOUGH_GRADIENT, 1, 100, param1=100, param2=10, minRadius=10, maxRadius=30)
        if circles is not None:
            for circle in circles[0]:

                x = int(circle[0])
                y = int(circle[1])
                r = int(circle[2])

                cv2.circle(frame, (x, y), r, (0, 0, 255), 2)
                cv2.circle(frame, (x, y), 3, (0, 0, 255), -1)
                x_velocity = x - x_before
                x_before = x
                y_velocity = y - y_before
                y_before = y
                text = 'x:' + str(x) + ' y:' + str(y) + '    V:' + str(x_velocity) + ' ' + str(y_velocity)
                cv2.putText(frame, text, (10, 30), font, 1, (138, 223, 151), 2, cv2.LINE_4, 0)
                # print('P:', str(x), str(y), 'V:', str(x_velocity), str(y_velocity))
        else:
            cv2.putText(frame, 'x:None y:None V:None', (10, 30), font, 1, (138, 223, 151), 2, cv2.LINE_AA, 0)
        cv2.imshow('frame', frame)
        #cv2.imshow('gray', gray)
        cv2.imshow('edges', edges)
        #控制舵机
        if ser.isOpen():
            print('串口已打开')
            #  判断与目标点偏差
            if y != 320:
                if y < 320:
                    data_y = b'right\r\n'
                elif y > 320:
                    data_y = b'zuo\r\n'

            if x != 240:
                if x < 240:
```

```
                        data_x = b'up\r\n'
                    elif x > 240:
                        data_x = b'down\r\n'
                ser.write(data_x)
                print('x:', data_x)
                time.sleep(0.02)
                #data = ser.read(20)
                #print('receive data is :', data)
                ser.write(data_y)
                print('y:', data_y)
                #data = ser.read(20)
                #print('receive data is :', data)
            else:
                print('串口未打开')
            #if ser.isOpen():
            #    print('串口未关闭')
            # else:
            #    print('串口已关闭')

            # ser.close()

            k = cv2.waitKey(5) & 0xFF
            if k == 27:
                break
            # end time
            end = time.time()
            # Time elapsed
            seconds = end - start
            # print("Time taken : {0} seconds".format(seconds))
            # Calculate frames per second
            fps = 1 / seconds
            # print("Estimated frames per second : {0}".format(fps))
    data_0 = b'rreset\r\n'
    ser.write(data_0)
    print('reset')

    ser.close()
    print('串口已关闭')
    cap.release()
    cv2.destroyAllWindows()
else:
    print('cap is not opened!')
```