2016

Automated Car Guiding System Using Reinforcement Learning

Haoran Zhang
Lehigh University

AUTOMATED CAR GUIDING SYSTEM USING REINFORCEMENT LEARNING

by

HAORAN ZHANG

A Thesis

Presented to the Graduate and Research Committee

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial System Engineering

Lehigh University

May 2016

Thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science in Industrial System Engineering.

AUTOMATED CAR GUIDING SYSTEM USING
REINFORCEMENT LEARNING

HAORAN ZHANG

_____
Date Approved

_____
Prof. Martin Takáč

_____
Prof. Tamás Terlaky

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisor Professor Martin Takáč, who has offered me the support with his patience, motivation, enthusiasm, and immense knowledge. It would not be possible to finish this thesis without his help and guidance. I also would like to thank my parents for the support and encouragement.

CONTENTS

LIST OF TABLE

LIST OF FIGURES

ABSTRACT

The major objective of this project is to design and implement a car guiding system in a desk-size area, with a remote-controlled toy car.

The software, including the calculation, image processing, and movement control, was coded with Python and C++. Q-learning algorithm was selected to be the core of the calculation part, and OpenCV library was used for image processing.

The hardware consists of a webcam, a laptop, and a toy car with Bluetooth connection. Some wood boards were also used to build a frame for restraining the area for running the car.

The system is able to track the car, detect the obstacles, calculate the optimal path, and send signals to the car in order to control the movement.

CHAPTER 1

INTRODUCTION

Parking is becoming a serious problem nowadays, especially in big cities. Drivers spend too much time on finding a parking place, or walking between parking and destination if the cars have to be parked far away. Therefore, an automated car guiding system for parking is very necessary. This system should be able to find a parking place by itself, after the driver leaves the car at the destination.

The whole system is a combination of several functions. First, we need an algorithm that provides an optimal path from the current location of the car to the destination, which should be a vacant parking place. Second, we need a location detection function which not only can find out where the obstacles are, but also track the location of the car. Third, communication function is needed to send signal to the car, in order to make the car move along the optimal path calculated before.

CHAPTER 2

Q-LEARNING ALGORITHM

In this chapter, the core algorithm of this project is presented. A simple example of Q-learning algorithm will be introduced.

2.1 A Q-LEARNING SIMPLE EXAMPLE

Q-learning is a numerical optimization method of reinforcement learning, which is the core algorithm of this project.

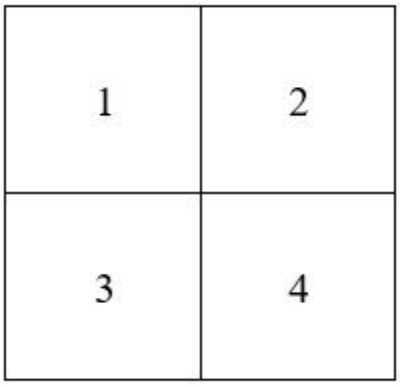

Figure 2.1: Four states of Q-learning algorithm example.

For instance, there are 4 squares in Figure 2.1, which represent 4 states. An object can move from one state to its neighbor state, such as moving from state 1 to state 3, moving from state 3 to state 4, or stay without any movement.

The destination is state 4, a path which has the fewest movements from a specific state to the destination should be optimal.

Now, let's assign a reward of 100 to the movement which get the object to the destination. For example, if the object is in state 3, then moving right to state 4 will have a reward of 100. However, if the object moves from state 2 to state 4, the reward will be assigned as -1 because the object can only go to the neighbor state. For all other movements, the reward will be assigned as 0. According these reward rules, we will have the R matrix shows as table 2.1.

		Actions			
		1	2	3	4
States	1	0	0	0	-1
	2	0	0	-1	100
	3	0	-1	0	100
	4	0	-1	0	100

Table 2.1: The R matrix of Q-learning algorithm example.

The R matrix stores the values of rewards for all the movement options in each states. With these values, we will have another matrix. The rows represent the current states, and the columns represent the next states. This is the Q matrix. All the values in the Q matrix are initialized to 0, as shown in table 2.2.

		States			
		1	2	3	4
States	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	0
	4	0	0	0	0

Table 2.2: The initialized Q matrix of Q-learning algorithm example.

Now we are using an equation to update the values in the Q matrix.

$$Q(s,a) = R(s,a) + Gamma * Max[Q(next\ s, all\ a)]. \quad (2,1)$$

In (2,1), $Q(s,a)$ is the value which will be updated after the calculation, s represents the current state, and a is the next state after taking the action. The gamma value in (2.1) is between 0 to 1. The higher value of gamma means we focus more on the choice of next step than the current. We let the gamma equals to 0.6. By using equation 1.1, the values in Q matrix will be updated continuously. For example, now the object is in state 3, and we randomly take the action to state 4. The Q value should be calculated as below.

$$Q(3,4) = R(3,4) + 0.6 * Max[Q(4,2), Q(4,3), Q(4,4)] = 100 + 0.6 * Max(0,0,0) = 100. \quad (2,2)$$

The value in Q matrix was updated shown in table 2.3.

		States			
		1	2	3	4
States	1	0	0	0	0
	2	0	0	0	0
	3	0	0	0	100
	4	0	0	0	0

Table 2.3: The Q matrix updated once.

Then we repeat the step, now the object is in state 1, and take the action to state 3 randomly.

$$Q(1,3) = R(1,3) + 0.6 * Max[Q(3,1), Q(3,3), Q(3,4)] = 0 + 0.6 * Max(0,0,100) = 60. \quad (2,3)$$

Again, the Q matrix is updated according to the result from (2,3). After repeating this calculation and updating the data several times, we will have a Q matrix shows as table 2.4.

		States			
		1	2	3	4
States	1	0	60	60	0
	2	36	60	0	0
	3	36	0	60	100
	4	0	60	60	100

Table 2.4: The calculated Q matrix of Q-learning algorithm example.

According to the values in the Q matrix, we will know what is the best movement option in each state. For example, if the object is in state 2, going to state 4 has the highest Q value, so we take this action. In this way, the object will always know where to go in any state, which is the optimal path.

2.2 Q-LEARNING APPLIED IN THE PROJECT

In this project, we use a webcam, with a resolution of 640 by 480, to detect the location and angle of the car. If we regard each pixel in the image taken by the webcam as a location, plus 360 degrees in each location, we will have 110,592,000 states. This will take an intolerably long time to calculate the Q matrix with a laptop.

After several times of tests, we found a balance between accuracy and efficiency, we decided to set up the locations with the size of 320 by 240, and 90 angles in this area. So, there are 6,912,000 states in this project.

As the introduction in Chapter 2, the car is able to have 5 different actions. In order to improve the efficiency of the program, all zero values are deleted from the R matrix and Q matrix, which means we only have 5 columns represent the 5 states we can get after doing the movement. Besides this, there are some other methods are used in the programming part, which will be introduced in Chapter 3.

CHAPTER 3

DEVICES AND HARDWARE

In this chapter, all the devices and hardware used in the project will be introduced, including the design which was abandoned because of some unsolvable problems.

3.1 CARS IN THE PROJECT

There were two designs of cars in this project. We started with the first design, but then it was found that there are some problems which were hard to be solved. A toy car with remote control was used in the first design. This car has 7 actions, besides the moving straight forward and backward, it was able to make turns like a real car, which were moving forward left and right, and reversing left and right. Staying was also counted as an action.

A Raspberry Pi computer was connected to the controller of the car, with pin headers and sockets. The Raspberry Pi computer controlled the movement of the car by turning the switches on and off in the remote controller. Also, a connection was built between the Raspberry Pi computer and the laptop via SSH. According the result calculated by the laptop, signals were sent to the Raspberry Pi, in order to make the car move.

However, we found some problems after several times of test. First, the accuracy is not tolerable. For a guiding system based on Q-learning algorithm, we need to know where the car will get if a specific action is executed, that is, if we do an action from a state, the next state after this movement should be fixed. But the car cannot move strictly as predicted sometimes. There are two reasons for this problem. First, this is a toy car, the mechanical parts are not reliable. Second, it takes too long to send a signal from the laptop to the Raspberry Pi, process the signal, and control the switch of the remote controller. In consequence, we abandon the first design.

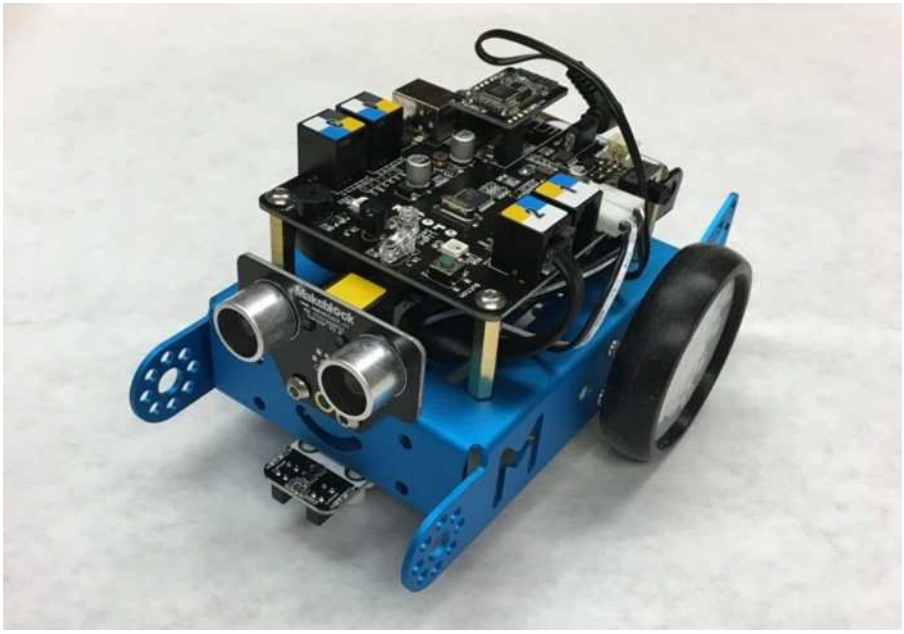

Figure 3.1: Makeblock mBot.

A Makeblock mBot car in Figure 3.1 is used in the second design. The mechanical parts of this car are more reliable, which means the movement is almost

predictable. The car is driven by a Bluetooth module, which can be connected to the laptop directly. The firmware of the car is rewritten to fix the length of moving forward or backward, and the angle for turning.

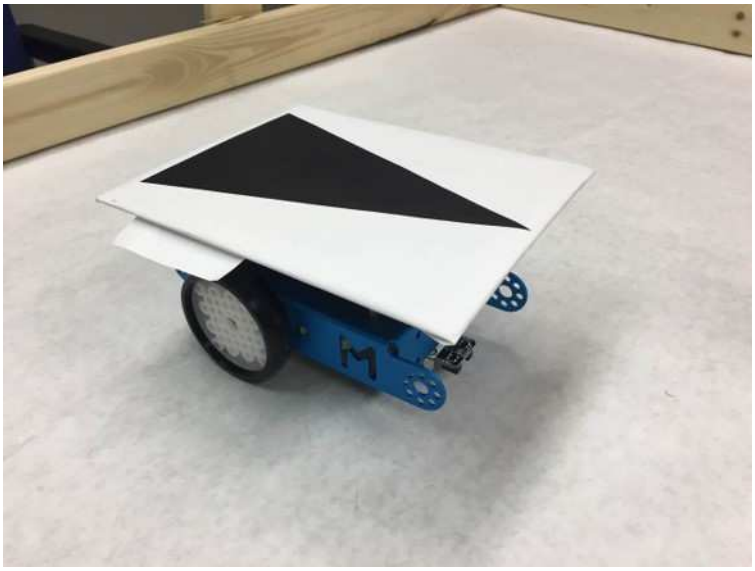

Figure 3.2: Isosceles triangle on the top of the car.

As the Figure 3.2 shows, an isosceles triangle is put on the top of the car, which is used for detecting the location and angle of the car. The detecting method will be introduced in Chapter 4.

3.2 LOCATION DETECTION WITH A WEBCAM

Figure 3.3: Webcam.

A webcam is used for detecting the object in this project, vertically hanging above the central point of the area. Images taken with the webcam are processed by the program coded with Python and OpenCV, and the information, such as coordinates and angles, will be used in controlling the movement of the car. The resolution of the webcam is 640 by 480, which is lower than normal webcams. A higher resolution webcam is not used in this project because a larger size image will lower the running speed of the whole system. We always need to track the location of the car before making each movement, a high efficiency of image processing will make the car running faster.

3.3 DESIGN OF TEST AREA

Figure 3.4: Design of test area.

Figure 3.4 shows the whole test area of this project. The webcam on the top is able to capture the image of the area. The boundary built up with wood board will

be regarded as obstacles in the image processing, which can make sure that the car will not hit the boundary.

CHAPTER 4

SOFTWARE

In this chapter, all the functions realized by coding will be introduced, including the calculation of Q matrix, the image process, and the movement control of the car.

4.1 CODING OF Q-LEARNING ALGORITHM

As mentioned in Chapter 2, there are 6,912,000 rows and columns of data in Q matrix. That is tremendously time consuming. The code of Q-learning algorithm is written with Python at first, but after several times of test running, it's realized that transforming the code from Python to C++ is necessary.

Python is 10 to 100 times slower than C++ on execution time because the code is interpreted at runtime instead of being complied as native code at compile phase. So we transfer the process of calculating Q matrix from Python to C++, and use python to do data preparation, image capture and processing, and car movement controlling.

The Q Matrix computing consists of two critical parts. The first one is the calculation of R Matrix. It is fundamental as well as the core of the Q-learning.

```
1. this->get_next_state(state,angle,action,&values[0]);
2. int next_state=values[0];
3. int next_angle=values[1];
4. int next_x=values[2];
5. int next_y=values[3];
6. if (next_x > this->x_end-
   5 and next_x < x_end+5 and next_y > this->y_end-
   5 and next_y < y_end+5 and next_angle == this->angle_end){
7. return 1000;
8. }
9. else if( next_x < 1 or next_x > (this->x_size-1)-
   1 or next_y < 1 or next_y > (this->y_size-1)-1)
10.{return -1000;}
11.else if(
12.this->obstacles[next_x*this->y_size+next_y]
13.){
14.return -1000;
15.}
16.else{
17.return 0;
18.}
```

Figure 4.1: Source code for R matrix computing.

This function initializes the R value for one specific combination of state, angle and action. For differentiating potential routes and the path that will lead to a dead end, we manually pick 1000 to be the reward value when the action leads to the destination. The value of 1000 means that kind of next state is the appropriate one while the value -1000 is the penalty value which prohibits the program to choose this action to next state. This function is invoked for 6,912,000 times to initialize

the R matrix. For efficiency and reality concern, we reduce next states from over 600 million columns to only 5 columns and each column is the corresponding action. As a result, we have over 30 million (multiply 6,912,000 states by 5 actions) elements in R matrix.

As explained in Chapter 2, one hundred iterations are deployed to get the Q matrix based on the values from R matrix. The number of iterations is based on the difference before and after each iteration. When the difference getting close to zero, the Q matrix has almost been fully calculated. Otherwise, a Q matrix which has not been fully calculated will not be able to provide the optimal path to the destination. On the contrary, the total computational time would ascend according to the scale of the iteration times. So, we decide to have 100 iterations for Q matrix calculation.

For each possible state, we update the values in Q matrix.

```
1. Q[direction1 * totalStates + currentGlobalId] =
2. 0.5* ( RVal[direction1*RvalLD+currentGlobalId]+
3. gamma * V[newStateId]- Q[direction1 * totalStates+
4. currentGlobalId]);
```

Figure 4.2: Source code for updating Q values.

In order to simplify the code, we use C++ vector instead of 2-dimension array to store the Q Matrix. As mentioned before, the program will iterate for 100 times, and each time it outputs the total changes amount all possible states. During the iteration, Q values will converge to a constant value and the optimal one among all five actions will be the optimal choice for the next action of the car.

We output the changes between two iterations to ensure the Q value will converge finally. The total running time of C++ program is about 318 seconds while the original python program takes around 4 hours. Rewriting of the code grants us immense convenience and more test opportunities. And the final accuracy has been improved greatly since in each test we could adjust and calibrate the parameters.

4.2 CODING OF LOCATION DETECTION

At the beginning, we would like to use the HAAR Classifier in OpenCV to track the moving car and detect the obstacles. This Classifier uses a lot of positive images to train and extract features. Each feature is a single value of subtracting sum of pixels under white rectangle from sum of pixels under black rectangle. It is a good classifier to recognize moving units but it required relatively high resolution. Another drawback of this classifier is, the car's position is not steady and vacillates upside down by several pixels because there will be aliasing phenomenon. This is fatal to our project because we need accurate position and action. Because of the limitation of camera, we have to abandon the HAAR method.

Afterwards, we select Pixel Comparing algorithm to locate the car.

Straightforwardly, it compares every pixel between the image before placing the car (blank image) and the image after placing the car somewhere. The following code represents the core idea.

```
1. diff = np.fabs(img1-img2)
2. diff[diff < 115] = 255
3. diff[diff < 220] = 1
4. diff[diff > 220] = 255
```

Figure 4.3: Source code for comparing images.

When difference of two pixels is larger than 220, that is anomaly because under the illumination of general light, the grey scale of one pixel cannot be over 220. So we add a simple noise detection step in our program and actually it is effective. It also helps to detect the exact positions of obstacles by comparing the blank image with the one in which there is obstacles.

To make the car prominent, we put a black isosceles triangle pictures on the top of the car. So after comparison, the triangle will be highlighted as complete black pixel (grey scale = 255) and we set all others to be pure white (grey scale = 1). Next step is extracting three apexes of this isosceles triangle. We use Harris corner detection in OpenCV to separate the corner points from edges and inner points. To get three exact apex positions from all corner points, we iterate among all of them and cluster them by the distance between each other. A threshold of 40 pixels' length is set for distinguishing corners. If the distance between two corner pixels is larger than that threshold, then they are from two distinct corner clusters. The following figure shows how to find the corner clusters.

```
img_c = cv2.cvtColor(img_g,cv2.COLOR_GRAY2RGB)
dst = cv2.cornerHarris(img_g,10,3,k=0.01)
corner_set = dst > 0.35*dst.max()
img_c[corner_set] = [0,0,255]
corners = []
    for y in range(len(dst)):
        for x in range(len(dst[0])):
            if corner_set[y,x] == True:
                corners.append([x,y])
```

Figure 4.4: Source code for finding the corner clusters.

The center of each cluster is the precise pixel that represents one of three apexes. Among these three apexes, the pair which inner distance is smaller than other pairs of apexes are the base apex pair and we define the other one as M. We find the middle point called N of the base. Then calculate the middle point of line MN, and that centroid is the location of the car in our project. Also, the angle of the car will be calculated from the line MN.

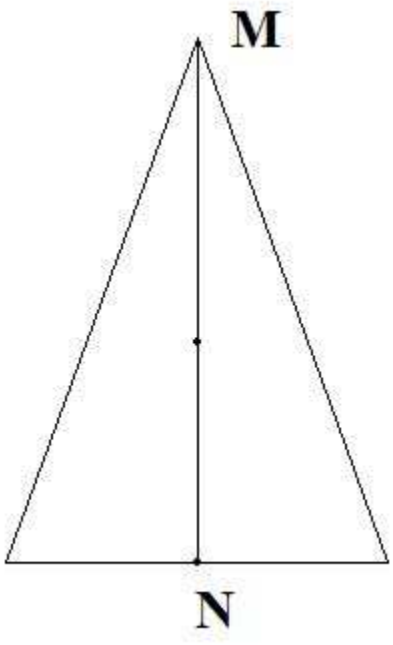

Figure 4.5: Isosceles triangle for car detection.

4.3 CODING OF MOVEMENT CONTROL

Although python has slow nature, it is still a good option when programming with Bluetooth API, data preparation or other functional operation where large amount of calculation is not included, because of its flexible and API-friendly instinct.

We define all parameters that need to be used and write it to local .txt files in python part. Than in the C++ part, it reads the parameter file, computes the Q-Matrix and then saves it to local file. One limitation here is the file operations of writing and reading Q Matrix. Q Matrix is around 350MB generally and with the

help of SSD (Solid – State - Drive) we are able to lessen the time of file operations to relatively low.

We use a Bluetooth connection kernel in Linux and send message to the car through it. There is an embedded chip processor in the car and it is programmable. We rewrite the code of it to connect it to the Python program. It has only four movements and we link each movement to a character instruction. So the car will react immediately after it receives an instruction. We decrease the length of message to one to make it more efficient.

CHAPTER 5

CONCLUSIONS

Several conclusions are reached throughout this project. Firstly, this model can be applied in some other situations, not only the parking lot. Such as doing heavy duty job in warehouses, dangerous works in nuclear reactors, or even in military. For all this applications, there is only one purpose, which is saving people from the works with risk. Because of the limit of time and budget, this project can only be performed on a desktop size area. But it should have a bright future with the development of technology.

A serious problem also appears when doing this project, which is the size of Q matrix. With such a low resolution of webcam and a very small test area, the size of Q matrix has reached million level. The data size will increase explosively. This will bring a large pressure on data storage and processing.

This project still has a space for doing some improvement, such as using neural network method to increase the efficiency of calculation. Future research should be focused on this area.

BIBLIOGRAPHY

- DuVander, Adam. "What Programming Language Is Most Popular with APIs?" *ProgrammableWeb*. 03 June 2013. Web. 05 May 2016. <http://www.programmableweb.com/news/what-programming-language-most-popular-apis/2013/06/03>.

- McCullock, John. "A Painless Q-Learning Tutorial." *Mnemosyne Studio*. Web. 05 May 2016. <http://mnemstudio.org/path-finding-q-learning-tutorial.htm>.

- Borenstein, Greg. "Adam Harvey Explains Viola-Jones Face Detection." *Makematics*. 2012. Web. 05 May 2016. <http://www.makematics.com/research/viola-jones/>.

- "OpenCV Tutorials." *OpenCV Tutorials — OpenCV 2.4.13.0 Documentation*. Opencv Dev Team, 5 May 2016. Web. 05 May 2016. <http://docs.opencv.org/2.4/doc/tutorials/tutorials.html>.

- "C Language Tutorial." *C Tutorials*. Cplusplus.com. Web. 05 May 2016. <http://www.cplusplus.com/doc/tutorial/>.

- Mordvintsev, Alexander, and Abid K. Revision. "Feature Matching Homography to Find Objects." *Feature Matching Homography to Find Objects*. 2013. Web. 05 May 2016. <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_feature_homography/py_feature_homography.html#feature-homography>.

- "Animate a Line with Different Colors." *Stack Overflow*. Web. 05 May 2016. <http://stackoverflow.com/questions/21077477/animate-a-line-with-different-colors>.

APPENDIX

This appendix presents the sources code for this project.

```
1. import cv2
2. import numpy as np
3. import math
4. import timeit
5.
6.
7. def get_image():
8.     camera = cv2.VideoCapture(0)
9.     retval,img = camera.read()
10.    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
11.    camera.release()
12.    return img
13.
14.
15.
16.
17.def compare(img1,img2):
18.
19.    x = len(img1[0])
20.    y = len(img1)
21.    #img_d = np.ones((y,x),dtype = 'uint8')
22.    #img_d = img_d*255
23.
24.
25.
26.
27.
28.    diff = np.fabs(img1-img2)
29.    diff[diff < 115] = 255
30.    diff[diff < 220] = 1
31.    diff[diff > 220] = 255
32.    #img_d[diff < 115] = 255
33.
34.    for i in range(y):
35.        for j in range(x):
36.            pass
```

```
    return diff.astype( 'uint8')



def get_distance(p1,p2):
    lenth = np.sqrt(((p1[0]-p2[0])**2)+((p1[1]-p2[1])**2))
    return lenth

def get_center(cset):
    x_total = 0
    y_total = 0
    n = len(cset)
    for i in range(n):
        x_total = x_total + cset[i][0]
        y_total = y_total + cset[i][1]
    x_center = int(x_total/float(n))
    y_center = int(y_total/float(n))
    return [x_center,y_center]

def get_middle_point(p1,p2):
    x = int((p1[0]+p2[0])/2.0)
    y = int((p1[1]+p2[1])/2.0)
    return [x,y]

def find_corner(img_g):
    img_c = cv2.cvtColor(img_g,cv2.COLOR_GRAY2RGB)
    dst = cv2.cornerHarris(img_g,10,3,k=0.01)
    corner_set = dst > 0.35*dst.max()
    img_c[corner_set] = [0,0,255]
    #cv2.imshow('test',img_c)

    corners = []
    for y in range(len(dst)):
        for x in range(len(dst[0])):
            if corner_set[y,x] == True:
                corners.append([x,y])

        #print(corners)



    corner = []
    ini_point = []
    for i in range(len(corners)):
        current_point = corners[i]
        if i == 0:
            corner.append([])
            corner[0].append(current_point)
            ini_point.append(current_point)

```

```
        else:
            distances = []
            for m in range(len(ini_point)):
                distances.append(get_distance(current_point,ini_point[m]))

            min_distance_idx = np.argmin(distances)
            if distances[min_distance_idx] > 40:
                ini_point.append(current_point)
                cor_idx = len(corner)
                corner.append([])
                corner[cor_idx].append(current_point)
            else:
                        corner[min_distance_idx].append(current_point)



        pointvol = []
        for m in range(len(corner)):
            pointvol.append(len(corner[m]))


        corner_temp = []
        for n in range(3):
            current_largest = np.argmax(pointvol)
            pointvol[current_largest] = 0
            corner_temp.append(corner[current_largest])
        corner = corner_temp


        corner3 = []
        for k in range(3):
            tem = get_center(corner[k])
            cv2.circle(img_c,(tem[0],tem[1]),5,(0,0,255),-1)
            corner3.append(tem)
            #print('Three corners of the triangle is '+ str(corner3)+'.')
        return corner3

    def get_info(corners):

        x,y = get_center(corners)
        lenth1 = get_distance(corners[0],corners[1])
        lenth2 = get_distance(corners[1],corners[2])
        lenth3 = get_distance(corners[0],corners[2])
        middle_point1 = get_middle_point(corners[0],corners[1])
```

```
133.           middle_point2 = get_middle_point(corners[1],corners[2
    ])
134.           middle_point3 = get_middle_point(corners[0],corners[2
    ])
135.           middle_line = []
136.           angle = 0
137.           x_val = 0
138.           y_val = 0
139.           if abs(lenth1-lenth2) <= 10:
140.               middle_line = [middle_point3,corners[1]]
141.           if abs(lenth2-lenth3) <= 10:
142.               middle_line = [middle_point1,corners[2]]
143.           if abs(lenth1-lenth3) <= 10:
144.               middle_line = [middle_point2,corners[0]]
145.
146.           x_val = float(middle_line[1][0] - middle_line[0][0])

147.           y_val = float(middle_line[0][1] - middle_line[1][1])

148.
149.           if x_val == 0 and y_val > 0:
150.               angle = 360
151.           if x_val == 0 and y_val < 0:
152.               angle = 180
153.           if x_val > 0 and y_val == 0:
154.               angle = 90
155.           if x_val < 0 and y_val == 0:
156.               angle = 270
157.           if x_val > 0 and y_val > 0:
158.               angle = math.atan(x_val/y_val)*180/math.pi
159.           if x_val > 0 and y_val < 0:
160.               angle = 180+math.atan(x_val/y_val)*180/math.pi
161.           if x_val < 0 and y_val > 0:
162.               angle = 360+math.atan(x_val/y_val)*180/math.pi
163.           if x_val < 0 and y_val < 0:
164.               angle = 180+math.atan(x_val/y_val)*180/math.pi
165.
166.           angle = int(angle)
167.           print('Current coordinate is ('+str(x)+','+str(y)+'),
    angle is '+str(angle)+'.')
168.           return x,y,angle
169.
170.
171.       def get_obs(img_blank,xGrid,yGrid):
172.           img_current = get_image()
173.           height = len(img_current)
174.           width = len(img_current[0])
175.           xGrid_size = width/float(xGrid)
176.           yGrid_size = height/float(yGrid)
177.
178.           img_ob = compare(img_blank,img_current)
```

```
179.
180.          img_t = 255 - img_ob
181.          obs = []
182.          for ii in range(yGrid):
183.              y1 = int(ii*yGrid_size)
184.              y2 = int((ii+1)*yGrid_size)
185.              for jj in range(xGrid):
186.                  x1 = int(jj*xGrid_size)
187.                  x2 = int((jj+1)*xGrid_size)
188.                  s = sum(sum(img_t[y1:y2,x1:x2]))
189.                  if s >= 100:
190.                      obs.append([jj,ii])
191.          return img_ob,obs
192.
193.      def get_car_info(img_ob):
194.
195.          img_current = get_image()
196.          img_car = compare(img_ob,img_current)
197.
198.          corner = find_corner(img_car)
199.          x,y,angle = get_info(corner)
200.
201.          return x,y,angle
```

Figure 5.1: Source code for image process.

```
1.  #parameters initialization#
2.  xGrid = x_size =  320
3.  yGrid = y_size =  240
4.  windows = False
5.  useC = True
6.  numberOfAngles = 90
7.  turning_angle = 360/numberOfAngles
8.  linear_distance =  8
9.  #length of 4 pixel
10. margin=35
11. numberofActions = 5
12. iteration = 1200
13.
14.
15. DEBUG=False
16. combo_move = False
17.
18. import cv2
19. from bluetooth import *
20.
21.
```

```
22.
23.
24. import os
25. import numpy as np
26. import socket
27. import math
28. from Qlearning import Qlearning
29. from subprocess import call
30. import sys
31. import time
32. import matplotlib.pyplot as plt
33. from image_process import get_image,get_obs,get_car_info,compare
    ,find_corner,get_info
34.
35.
36. port = 1
37. name = "MakeBlock"
38. host =  "00:1B:10:0E:0B:F4"
39. sock=BluetoothSocket( RFCOMM )
40. sock.connect((host, port))
41. camera = cv2.VideoCapture(0)
42.
43. def coord2index(x,y):
44.     index = y*x_size+x
45.     return index
46.
47. def index2coordx(ind):
48.     x_c = ind%x_size
49.     return x_c
50.
51. def index2coordy(ind):
52.     y_c = ind/x_size
53.     return y_c
54.
55. def pixel2index(x,y):
56.     x_ind = x/xyratio
57.     y_ind = y/xyratio
58.     index = coord2index(x_ind,y_ind)
59.     return index
60.
61. def movement(state,angle):
62.     Q_idx = state*numberOfAngles+(angle/turning_angle)
63.     Q_current = Q[:,Q_idx]
64.
65.     Q_current[2] = Q_current[2]*1.0005
66.     Q_current[3] = Q_current[3]*1.0005
67.     #print('Q_current is '+ str(Q_current))
68.     action = np.argmax(Q_current)
69.     return action
70.
71. def movement_rep(state,angle):
```

```
72.     action_queue = []
73.     Q_idx = state*numberOfAngles+(angle/turning_angle)
74.     Q_current = Q[:,Q_idx]
75.     Q_current[2] = Q_current[2]*1.00003
76.     Q_current[3] = Q_current[3]*1.00003
77.     first = np.argmax(Q_current)
78.     action_queue.append(first)
79.     Q_current[first] = 0
80.     second = np.argmax(Q_current)
81.     action_queue.append(second)
82.     Q_current[second] = 0
83.     third = np.argmax(Q_current)
84.     action_queue.append(third)
85.     Q_current[third] = 0
86.     forth = np.argmax(Q_current)
87.     action_queue.append(forth)
88.     return action_queue
89.
90.
91.
92. def movement_des(state,angle):
93.     Q_idx = state*numberOfAngles+(angle/turning_angle)
94.     Q_current = Q[:,Q_idx]
95.     Q_current[0] = 0
96.     Q_current[1] = 0
97.     action = np.argmax(Q_current)
98.     return action
99.
100.
101.
102.       #img1 blank space
103.       #img2 space with obstacles
104.       #img3 space with car and obstacles
105.
106.
107.       if raw_input('Please leave the area empty, then press ent
    er:\n'):
108.           pass
109.
110.
111.       for i in xrange(10):
112.           retval,img0 = camera.read()
113.       img1 = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)
114.
115.
116.
117.       height = len(img1)
118.       width = len(img1[0])
119.       obstacles = []
120.       if raw_input('Please place the obstacles, then press ente
    r:\n'):
```

```
121.           pass
122.
123.
124.       for i in xrange(10):
125.           retval, img_ob0 = camera.read()
126.       img_ob1 = cv2.cvtColor(img_ob0, cv2.COLOR_BGR2GRAY)
127.
128.
129.       img_ob2 = compare(img1,img_ob1)
130.       img_ob = 255 - img_ob2
131.       xGrid_size = width/float(xGrid)
132.       yGrid_size = height/float(yGrid)
133.
134.
135.
136.       for ii in xrange(yGrid):
137.           y1 = int(ii*yGrid_size)
138.           y2 = int((ii+1)*yGrid_size)
139.           for jj in xrange(xGrid):
140.               x1 = int(jj*xGrid_size)
141.               x2 = int((jj+1)*xGrid_size)
142.               s = sum(sum(img_ob[y1:y2,x1:x2]))
143.               if s >= 100:
144.                   obstacles.append([jj,ii])
145.                   for xi in xrange((jj-margin),(jj+margin)):
146.                       for yi in xrange((ii-
    margin),(ii+margin)):
147.                           if xi >= 0 and xi <= (x_size-
    1) and yi >= 0 and yi <= (y_size-1):
148.                               obstacles.append([xi,yi])
149.
150.
151.
152.       while True:
153.           x_end = input('Please input the x coordinate of desti
    nation(range:50~270):\n')
154.           if x_end >= 0 and x_end <= x_size:
155.               break
156.
157.       while True:
158.           y_end = input('Please input the y coordinate of desti
    nation(range(50~190):\n')
159.           if y_end >= 0 and y_end <= y_size:
160.               break
161.
162.       while True:
163.           angle_end = input('Please input the angle of destinat
    ion:\n')
164.           if angle_end >= 0 and angle_end <= 358 and angle_end
    turning_angle == 0:
165.               break
```

```
166.
167.      index_end = coord2index(x_end,y_end)
168.
169.      obsDic={}
170.      for e in obstacles:
171.          obsDic['_'.join(str(e))]=e
172.
173.      for i in xrange(x_size):
174.          for j in xrange(y_size):
175.              if i <= margin or i >= (x_size-
    margin) or j <= margin or j >= (y_size-margin):
176.                  e = [i,j]
177.                  obsDic[ '_'.join(str(e)) ] = e
178.
179.      obstacles = obsDic.values()
180.
181.
182.
183.
184.
185.
186.      xyratio = height/y_size
187.      xGrid_size = width/float(xGrid)
188.      yGrid_size = height/float(yGrid)
189.
190.      angleDisc = numberOfAngles
191.      angleratio = 360/angleDisc
192.
193.
194.
195.
196.
197.
198.
199.
200.
201.
202.
203.
204.      if index_end in obstacles:
205.          print('ERROR! Destination is in obstacles!')
206.
207.
208.
209.      if windows:
210.
211.          ft1 = os.path.isfile('D:/temp/obstacles.in')
212.          if ft1 == True:
213.              os.remove('D:/temp/obstacles.in')
214.
215.          ft2 = os.path.isfile('D:/temp/q.out')
```

```
216.          if ft2 == True:
217.              os.remove('D:/temp/q.out')
218.
219.          ft3 = os.path.isfile('D:/temp/parameters.in')
220.          if ft3 == True:
221.              os.remove('D:/temp/parameters.in')
222.
223.
224.
225.
226.      else:
227.
228.          ft4 = os.path.isfile('./obstacles.in')
229.          #if ft4 == True:
230.          #    os.remove('./obstacles.in')
231.
232.          ft5 = os.path.isfile('./q.out')
233.          #if ft5 == True:
234.          #    os.remove('./q.out')
235.
236.          ft6 = os.path.isfile('./parameters.in')
237.          #if ft6 == True:
238.          #    os.remove('./parameters.in')
239.
240.
241.      obstacles_trans = np.zeros((2,len(obstacles)))
242.
243.
244.
245.      parms = []
246.      ob_lenth = len(obstacles)
247.      parms.append(ob_lenth)
248.      parms.append(x_size)
249.      parms.append(y_size)
250.      parms.append(angleDisc)
251.      parms.append(x_end)
252.      parms.append(y_end)
253.      parms.append(index_end)
254.      parms.append(angle_end)
255.      parms.append(linear_distance)
256.      parms.append(numberofActions)
257.      parms.append(iteration)
258.      parms.append(windows)
259.      parms.append(useC)
260.
261.      print(parms)
262.      if useC:
263.          if windows:
264.              np.savetxt('D:/temp/obstacles.in',obstacles, deli
   miter=' ')
```

```
265.                 np.savetxt('D:/temp/parameters.in', parms, delimi
    ter=' ')
266.
267.             else:
268.                 np.savetxt('obstacles.in', obstacles, delimiter='
     ')
269.                 np.savetxt('parameters.in', parms, delimiter=' ')

270.
271.
272.
273.             if windows:
274.                 call('D:/temp/qlearning.exe')
275.                 Q = np.loadtxt("D:/temp/q.out", delimiter=' ' )
276.             else:
277.                 call('./qlearning')
278.                 Q = np.loadtxt("./q.out", delimiter=' ' )
279.
280.         else:
281.             parms[-1]= 0
282.             if windows:
283.                 np.savetxt('D:/temp/parameters.in', parms, delimi
    ter=' ')
284.
285.             else:
286.                 np.savetxt('parameters.in', parms, delimiter=' ')

287.             qlearn = Qlearning(obstacles,x_size,y_size,angleDisc,
    index_end,angle_end,linear_distance,numberofActions,iteration,wi
    ndows)
288.
289.
290.             Q = qlearn.get_Q()
291.
292.         def move(x,y,angle,action):
293.
294.             if action == 0:
295.                 newAngle = angle
296.                 newX = x + linear_distance * math.sin(float(newAn
    gle)/float(360)*2*math.pi)
297.                 newY = y - linear_distance * math.cos(float(newAn
    gle)/float(360)*2*math.pi)
298.             if action == 1:
299.                 newAngle = angle;
300.                 newX = x - linear_distance * math.sin(float(newAn
    gle)/float(360)*2*math.pi)
301.                 newY = y + linear_distance * math.cos(float(newAn
    gle)/float(360)*2*math.pi)
302.             if action == 2:
303.                 newAngle = (angle - turning_angle)%360
304.                 newX = x
```

```
305.                 newY = y
306.             if action == 3:
307.                 newAngle = (angle + turning_angle)%360
308.                 newX = x
309.                 newY = y
310.             if action == 4:
311.                 newAngle = angle
312.                 newX = x
313.                 newY = y
314.             return newX,newY,newAngle
315.
316.
317.
318.         def path_finder(xLoc,yLoc,angleLoc):
319.             x_idxLoc = round(xLoc/(xyratio+0.0))
320.             y_idxLoc = round(yLoc/(xyratio+0.0))
321.             path = []
322.             actions = []
323.             path.append([x_idxLoc,y_idxLoc])
324.             it = 0
325.             while True:
326.                 it = it + 1
327.                 if it>800:
328.                     break
329.                 tval = destination_test(xLoc,yLoc,angleLoc)
330.                 if tval == 1:
331.                     path.append([x_end,y_end])
332.                     break
333.                 stateLoc = coord2index(x_idxLoc,y_idxLoc)
334.                 actionLoc = movement(stateLoc,angleLoc)
335.                 nx,ny,nangle = move(x_idxLoc*xyratio+0.0,y_idxLoc
    *xyratio+0.0,angleLoc,actionLoc)
336.                 xLoc = nx
337.                 yLoc = ny
338.                 angleLoc = nangle
339.                 x_idxLoc = round(xLoc/(xyratio+0.0))
340.                 y_idxLoc = round(yLoc/(xyratio+0.0))
341.                 path.append([x_idxLoc,y_idxLoc])
342.                 actions.append(actionLoc)
343.             return path,actions
344.
345.         def draw_line(path):
346.             x = []
347.             y = []
348.             for i in xrange(len(path)):
349.                 x.append(path[i][0])
350.                 y.append(path[i][1])
351.             plt.plot(x,y)
352.             plt.show()
353.
354.         ###Plot initialization####
```

```
355.
356.      ob_x = []
357.      ob_y = []
358.      for index_o in xrange(len(obstacles)):
359.          ob_x.append(obstacles[index_o][0])
360.          ob_y.append(obstacles[index_o][1])
361.
362.      des_x = []
363.      des_y = []
364.
365.      for i in range(x_end-3,x_end+4):
366.          for j in range(y_end-3,y_end+4):
367.              des_x.append(i)
368.              des_y.append(j)
369.
370.
371.      plt.ion()
372.      plt.show()
373.
374.      def draw_route(path):
375.          x = []
376.          y = []
377.          for i in xrange(len(path)):
378.              x.append(path[i][0])
379.              y.append(path[i][1])
380.          plt.clf()
381.          plt.xlim(0,x_size)
382.          plt.ylim(y_size,0)
383.          plt.scatter(x[0],y[0],s=1,color='red')
384.          plt.scatter(des_x,des_y,s = 5,color='green')
385.          plt.scatter(ob_x,ob_y,s = 1.7,color='black')
386.          plt.plot(x,y)
387.          plt.draw()
388.
389.
390.
391.
392.
393.
394.      def angle_test(angle):
395.          val1 = (angle - angle_end)%360
396.          val2 = (angle_end-angle)%360
397.          if val1 >= val2:
398.              val = val2
399.          else:
400.              val = val1
401.          return val
402.
403.      def destination_test(x,y,angle):
404.          xt = 0
405.          yt = 0
```

```
406.           anglet = 0
407.           x_idx = x/xyratio
408.           y_idx = y/xyratio
409.           angle_tval = angle_test(angle)
410.           if abs(x_idx - x_end) <= 7:
411.               xt = 1
412.           if abs(y_idx - y_end) <= 7:
413.               yt = 1
414.           if  angle_tval <= 6:
415.               anglet = 1
416.           t_val = xt*yt*anglet
417.           if t_val == 1:
418.               return 1
419.           else:
420.               return 0
421.
422.
423.
424.
425.
426.       if raw_input('Please place the car, then press enter:\n')
    :
427.           pass
428.       while True:
429.           for i in xrange(10):
430.               retval,img2 = camera.read()
431.           img_t = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
432.           cv2.imshow('realtime',img_t)
433.           cv2.waitKey(1)
434.           img_d = compare(img_ob1,img_t)
435.           try:
436.               corner = find_corner(img_d)
437.               x,y,angle = get_info(corner)
438.               t_val1 = destination_test(x,y,angle)
439.               if t_val1 == 1:
440.                   print('The car has reached the destination.')

441.                   break
442.               else:
443.                   x_idx = round(x/xyratio)
444.                   y_idx = round(y/xyratio)
445.                   angle_idx =round(angle/angleratio)
446.                   angle_c = angle_idx*angleratio
447.                   x_c = x_idx*xyratio
448.                   y_c = y_idx*xyratio
449.                   state = coord2index(x_idx,y_idx)
450.                   path, actions = path_finder(x_c,y_c,angle_c)

451.                   draw_route(path)
452.                   action = actions[0]
453.
```

```
                    if action == 0:
                        sock.send('F')
                        print('The car is moving forward.')
                    if action == 1:
                        sock.send('B')
                        print('The car is moving backward.')
                    if action == 2:
                        sock.send('L')
                        print('The car is turing left.')
                    if action == 3:
                        sock.send('R')
                        print('The car is turing right.')
                except:
                    print "Unexpected error:", sys.exc_info()[0]

                    pass
        camera.release()
        time.sleep(10)
        plt.close()
```

Figure 5.2: Source code for data preparation and movement control.

```
#include <iostream>
#include <fstream>

using namespace std;
#include <vector>
#include <string.h>
#include <sys/time.h>
#include <math.h>
#define PI 3.14159265
int A_FORWARD = 0;
int A_BACKWARD = 1;
int A_LEFT = 2;
int A_RIGHT = 3;
int A_STAY = 4;
int gridPoints;
int anlgleDisc;

double gettime_(void) {
    struct timeval timer;
    if (gettimeofday(&timer, NULL))
        return -1.0;
    return timer.tv_sec + 1.0e-6 * timer.tv_usec;
}

 int getGLobalStateId(int state, int angleId) {
```

```
26.     return angleId + state * anlgleDisc;
27. 
28. }
29. 
30. class Qlearning{
31.     public:
32.     std::vector<bool> &obstacles;
33.     int x_size,y_size,angleratio,x_end,y_end,index_end,angle_end
    ,
34.     numberofActions,size_obstacles,angleDisc;
35.     double linear_distance;
36.     bool windows;
37. 
38. 
39. 
40.     Qlearning(std::vector<bool> &,int,int,int,int,int,int,int,in
    t,double,int,bool);
41.     int coord2index(int x, int y);
42.     int index2coordx(int ind);
43.     int index2coordy(int ind);
44. 
45.     int pixel2index(int x, int y);
46.     void get_next_state(int state,int angle,int action,int*ret_v
    alues);
47.     void get_Q();
48.     int get_R_value(int state,int angle,int action);
49. 
50.     int getGLobalStateId(int,int);
51.     ~Qlearning() {
52. 
53.     }
54. };
55. 
56. 
57. Qlearning::Qlearning(std::vector<bool> & _obstacles,int _size_ob
    stacles,
58. int _x_size,int _y_size,int _angleDisc,int _x_end,int _y_end,int
     _index_end,
59. int _angle_end,double _linear_distance,int _numberofActions,bool
     _windows)
60. :
61. obstacles(_obstacles)
62. {
63.     size_obstacles=_size_obstacles;
64. 
65. 
66.     x_size=_x_size;
67.     y_size=_y_size;
68.     angleratio=360/_angleDisc;
69.     angle_end=_angle_end;
70.     x_end = _x_end;
```

```
71.     y_end = _y_end;
72.     index_end=_index_end;
73.     linear_distance=_linear_distance;
74.     numberofActions=_numberofActions;
75.     windows=_windows;
76.     angleDisc=_angleDisc;
77. }
78.
79.
80. int Qlearning::coord2index(int x, int y){
81.     int index = (y*this->x_size+x);
82.     return index;
83.
84. }
85.
86.
87. int Qlearning::index2coordx(int ind){
88.     int x_c = (ind%this->x_size);
89.     return x_c;
90. }
91.
92.
93. int Qlearning::index2coordy(int ind){
94.     double y_c = double(ind/(this->x_size));
95.     return int(y_c);
96. }
97.
98.
99. int  Qlearning::pixel2index(int x, int y){
100.          int x_ind = x/2.0;
101.          int y_ind = y/2.0;
102.          int index = this->coord2index((x_ind),(y_ind));
103.          return index;
104.      }
105.
106.
107.      void Qlearning::get_next_state(int state,int angle,int ac
    tion,int* ret_values){
108.
109.          int currentX =  this->index2coordx(state);
110.          int currentY =  this->index2coordy(state);
111.          int realX = currentX*4;
112.          int realY = currentY*4;
113.          double linear_distance = this->linear_distance;
114.          int turning_angle = this->angleratio;
115.
116.
117.
118.          double real_newX=0;
119.          double real_newY=0;
120.          double newAngle=0;
```

```
121.
122.
123.          if ((action) == A_FORWARD){
124.           newAngle = angle;
125.
126.           real_newX = realX + double(linear_distance * sin(dou
   ble(newAngle)/double(360)*2*PI));
127.           real_newY = realY - double(linear_distance * cos(dou
   ble(newAngle)/double(360)*2*PI));
128.
129.          }
130.
131.          if ((action) == A_BACKWARD){
132.           newAngle = angle;
133.           real_newX = realX - double(linear_distance * sin(dou
   ble(newAngle)/double(360)*2*PI));
134.           real_newY = realY + double(linear_distance * cos(dou
   ble(newAngle)/double(360)*2*PI));
135.          }
136.
137.          if (action == A_LEFT){
138.           newAngle = (angle - turning_angle)%360;
139.
140.
141.           real_newX = realX;
142.           real_newY = realY;
143.          }
144.
145.          if (action == A_RIGHT){
146.           newAngle = (angle + turning_angle)%360;
147.
148.
149.
150.           real_newX = realX;
151.           real_newY = realY;
152.
153.          }
154.
155.          if (action == A_STAY){
156.           newAngle = angle;
157.           real_newX = realX;
158.           real_newY = realY;
159.          }
160.          int ret_newAngle = int(newAngle);
161.          int ret_newState = (this->coord2index(int(round(real_
   newX/4)), int(round(real_newY/4))));
162.
163.               if(ret_newAngle <0)
164.               {
165.                   ret_newAngle = 360+ret_newAngle;
166.               }
```

```
167.          ret_values[0]=ret_newState;
168.          ret_values[1]=ret_newAngle;
169.          ret_values[2]=int(round(real_newX/4));
170.
171.          ret_values[3]=int(round(real_newY/4));
172.
173.
174.
175.      }
176.
177.
178.
179.      int Qlearning::get_R_value(int state,int angle,int action
    ){
180.
181.
182.
183.
184.
185.          std::vector<int> values(4);
186.
187.
188.
189.
190.          this->get_next_state(state,angle,action,&values[0]);

191.          int next_state=values[0];
192.          int next_angle=values[1];
193.
194.          int next_x=values[2];
195.          int next_y=values[3];
196.
197.          if (next_x > this->x_end-
    5 and next_x < x_end+5 and next_y > this->y_end-
    5 and next_y < y_end+5 and next_angle == this->angle_end){
198.              return 1000;
199.          }
200.
201.          else if( next_x < 1 or next_x > (this->x_size-1)-
    1 or next_y < 1 or next_y > (this->y_size-1)-1)
202.          {return -1000;}
203.          else if(
204.              this->obstacles[next_x*this->y_size+next_y]
205.          ){
206.              return -1000;
207.          }
208.          else{
209.              return 0;
210.          }
211.
212.          }
```

```
213.
214.      int  Qlearning::getGLobalStateId(int state, int angleId){

215.      return angleId+state*this->angleDisc;
216.      }
217.
218.
219.
220.
221.      int main() {
222.
223.
224.
225.          double gamma = 0.95;
226.          int iterations;
227.
228.          std::ifstream ifs2("parameters.in", std::ifstream::in
    );
229.          double size_obstacles_double=0;
230.
231.          ifs2>>size_obstacles_double;
232.          double x_size_double,y_size_double,angleDisc_double,x
    _end_double,y_end_double,  index_end_double,angle_end_double,num
    berofActions_double,iteration_double;
233.          double linear_distance;
234.          double useC_double=0;
235.          double windows_double;
236.          ifs2>>x_size_double;
237.          ifs2>>y_size_double;
238.          ifs2>>angleDisc_double;
239.          ifs2>>x_end_double;
240.          ifs2>>y_end_double;
241.          ifs2>>index_end_double;
242.          ifs2>>angle_end_double;
243.          ifs2>>linear_distance;
244.          ifs2>>numberofActions_double;
245.          ifs2>>iteration_double;
246.          ifs2>>windows_double;
247.          ifs2>>useC_double;
248.          bool useC=bool(useC_double);
249.
250.          if(useC){
251.
252.          int size_obstacles=int(size_obstacles_double);
253.          int x_size,y_size,angleDisc,x_end,y_end,index_end,ang
    le_end,numberofActions;
254.          bool windows;
255.          x_size=int(x_size_double);
256.          y_size=int(y_size_double);
257.          angleDisc=int(angleDisc_double);
258.          x_end = int(x_end_double);
```

```
259.          y_end=int(y_end_double);
260.          index_end=int(index_end_double);
261.          angle_end=int(angle_end_double);
262.          numberofActions=int(numberofActions_double);
263.          iterations=int(iteration_double);
264.          windows=bool(windows_double);
265.
266.
267.           std::ifstream ifs("obstacles.in", std::ifstream::in)
    ;
268.
269.
270.          std::vector<bool> obstacles(x_size*y_size,0);
271.
272.          for(int i=0 ; i<size_obstacles;i++)
273.          {
274.              double temp1;
275.              double temp2;
276.                   ifs>>temp1;
277.                   ifs>>temp2;
278.                   obstacles[int(temp1) *y_size + int(temp2) ]=
    1;
279.
280.          }
281.
282.
283.          ifs.close();
284.          ifs2.close();
285.
286.          Qlearning qlearn=Qlearning(obstacles,size_obstacles,x
    _size,y_size,angleDisc,x_end,y_end,index_end,angle_end, linear_d
    istance,numberofActions,windows);
287.
288.          int size = qlearn.x_size * qlearn.y_size;
289.
290.         int RvalLD = size*(qlearn.angleDisc);
291.         std::vector<int> RVal (qlearn.numberofActions*RvalLD,0
    );
292.
293.
294.
295.          int**transition =new int*[qlearn.numberofActions];
296.          for (int i =0; i<qlearn.numberofActions; i++) {
297.              transition[i]=new int[size*(qlearn.angleDisc)];
298.
299.          }
300.            int**transition1 =new int*[qlearn.numberofActions];

301.          for (int i =0; i<qlearn.numberofActions; i++) {
302.              transition1[i]=new int[size*(qlearn.angleDisc)];
```

```
303.
304.           }
305.           std::ofstream os3;
306.           std::ifstream is3;
307.           is3.open("transition.in");
308.           for (int i =0;i<qlearn.numberofActions;i++)
309.               {
310.                   for(int j=0; j<size*qlearn.angleDisc;j++){
311.                       double temp1=0;
312.                       is3>>temp1;
313.                       transition1[i][j]=(int)temp1;
314.                   }
315.               }
316.               is3.close();
317.
318.           int angleScale = 360/qlearn.angleDisc;
319.           int *ret_values=new int [4];
320.
321.
322.           for (int state =0; state<size;state++){
323.               for (int angl =0;angl<qlearn.angleDisc;angl++){
324.                   for (int ac=0;ac<qlearn.numberofActions;ac++)
    {
325.                       qlearn.get_next_state(state, angl*angleSc
    ale, ac,ret_values);
326.                       int newState=ret_values[0];
327.                       int newAngle=ret_values[1];
328.
329.                       int newX=ret_values[2];
330.                       int newY=ret_values[3];
331.
332.           RVal[ac*RvalLD+qlearn.getGLobalStateId(state, angl)]

333.       =
334.                        qlearn.get_R_value(state, angl*angleScal
    e, ac);
335.
336.
337.                       transition[ac][qlearn.getGLobalStateId(st
    ate, angl)] =
338.                     qlearn.getGLobalStateId(newState, (newAngle
    /(angleScale)));
339.
340.
341.                       }
342.
343.               }
344.
345.           }
346.
347.
```

```
348.           delete[] ret_values;
349.
350.           int*V1 =new int[size*qlearn.angleDisc];
351.
352.           int *stateIdxForLearningTemp =new int[size];
353.           int statesForLearning = 0;
354.           int count=0;
355.
356.
357.            for (int i=0;i<size;i++){
358.
359.
360.                   int xx = qlearn.index2coordx(i);
361.                   int yy = qlearn.index2coordy(i);
362.
363.               if (   qlearn.obstacles[xx* qlearn.y_size+yy]   )
   {
364.
365.               }
366.
367.               else{
368.
369.                       stateIdxForLearningTemp[count]=i;
370.                       count++;
371.                       statesForLearning=statesForLearning+1;
372.                   }
373.           }
374.
375.
376.                   int *stateIdxForLearning=new int[count];
377.                   for(int i=0;i<count;i++)
378.                   {
379.                       stateIdxForLearning[i]=stateIdxForLearnin
   gTemp[i];
380.
381.                   }
382.                   delete[] stateIdxForLearningTemp;
383.
384.
385.           int actions;
386.
387.
388.           int totalStates;
389.                   actions=qlearn.numberofActions;
390.
391.            anlgleDisc = qlearn.angleDisc;
392.
393.
394.
395.           totalStates = size*qlearn.angleDisc;
396.           gridPoints = totalStates / anlgleDisc;
```

```
397.
398.         std::vector<double> V(totalStates, 0);
399.               std::vector<double> Q(totalStates * actions, 0);

400.           float tmp;
401.           double startTime = gettime_();
402.
403.           for (int it = 0; it < iterations; it++) {
404.
405.
406.
407.               for (int direction1 = 0; direction1 < actions; di
    rection1++) {
408.
409.       #pragma omp parallel for
410.                   for (int currentGlobalId = 0; currentGlobalId
     < totalStates;
411.                           currentGlobalId++) {
412.                       if (direction1 == 0) {
413.                           V[currentGlobalId] = Q[direction1 * t
    otalStates
414.                                   + currentGlobalId];
415.                       } else {
416.                           tmp = Q[direction1 * totalStates + cu
    rrentGlobalId];
417.                           if (V[currentGlobalId] < tmp) {
418.                               V[currentGlobalId] = tmp;
419.                           }
420.                       }
421.
422.                   }
423.
424.               }
425.
426.
427.
428.               double totalChange=0;
429.               for (int direction1 = 0; direction1 <  actions; d
    irection1++) {
430.
431.
432.         #pragma omp parallel for
433.                   for (int is = 0; is < statesForLearning; is++
    ) {
434.
435.                       int index_ep_start = stateIdxForLearning[
    is];
436.                       for (int ca = 0; ca < anlgleDisc; ca++) {

437.                          int currentGlobalId = getGLobalStateId
    (index_ep_start, ca);
```

```
                        int newStateId = transition[direction1][currentGlobalId];
                        double change = 0.5
                                * (
                                  RVal[direction1*RvalLD+currentGlobalId]
                                          + gamma * V[newStateId]
                                          - Q[direction1 * totalStates
                                                  + currentGlobalId]);


                         Q[direction1 * totalStates + currentGlobalId] += change;

                         totalChange+=change*change;
                    }

                }

            }

            cout << "iteration "<<it<<" Total change "<<totalChange<<endl;
        }
        cout << iterations << " iterations  took " << gettime_() - startTime<< 's'<<endl;
        std::ofstream os;









        os.open("q.out");
        for (int ac = 0; ac < actions; ac++) {
            for (int i = 0; i < totalStates; i++) {
                if (i == 0) {
                    os << std::fixed << Q[ac * totalStates + i];
                } else {
                    os << " " << std::fixed << Q[ac * totalStates + i];
                }
            }
            os << endl;
        }

```

```
479.          os.close();
480.
481.
482.
483.      delete[] V1;
484.
485.                  for(int i=0 ; i<2;i++)
486.                  {
487.
488.
489.                      delete[] transition[i];
490.                  }
491.
492.
493.                  delete[] transition;
494.                  delete[] stateIdxForLearning;
495.          return 0;
496.      }
497.
498.      else{
499.              std::ifstream ifs("parms.in", std::ifstream::in);

500.          float tmp;
501.          int actions;
502.
503.          int statesForLearningC;
504.          int totalStates;
505.          ifs >> tmp;
506.          actions = tmp;
507.          ifs >> tmp;
508.          anlgleDisc = tmp;
509.          ifs >> tmp;
510.          statesForLearningC = tmp;
511.          ifs >> tmp;
512.          totalStates = tmp;
513.          gridPoints = totalStates / anlgleDisc;
514.
515.          float gamma;
516.          ifs >> gamma;
517.          int iterations;
518.          ifs >> tmp;
519.          iterations = tmp;
520.          ifs.close();
521.
522.          cout << statesForLearningC << " " << anlgleDisc << "
    " << totalStates
523.                  << endl;
524.
525.          std::vector<int> stateIdxForLearning(statesForLearnin
    gC, 0);
```

```
526.          std::ifstream ifStateIdxForLearning("stateIdxForLearn
   ing.in",
527.                   std::ifstream::in);
528.          for (int i = 0; i < statesForLearningC; i++) {
529.              ifStateIdxForLearning >> tmp;
530.              stateIdxForLearning[i] = tmp;
531.          }
532.          ifStateIdxForLearning.close();
533.
534.          std::vector<int> RVal(totalStates * actions, 0);
535.          std::ifstream ifRVal("RVal.in", std::ifstream::in);
536.          for (int i = 0; i < totalStates * actions; i++) {
537.              ifRVal >> tmp;
538.              RVal[i] = tmp;
539.          }
540.          ifRVal.close();
541.
542.          std::vector<int> transition(totalStates * actions, 0)
   ;
543.          std::ifstream iftransition("transition.in", std::ifst
   ream::in);
544.          for (int i = 0; i < totalStates * actions; i++) {
545.              iftransition >> tmp;
546.              transition[i] = tmp;
547.          }
548.          iftransition.close();
549.
550.          std::vector<double> V(totalStates, 0);
551.          std::vector<double> Q(totalStates * actions, 0);
552.          double startTime = gettime_();
553.          for (int it = 0; it < iterations; it++) {
554.
555.              for (int direction1 = 0; direction1 < actions; di
   rection1++) {
556.                              currentGlobalId++) {
557.                      if (direction1 == 0) {
558.                          V[currentGlobalId] = Q[direction1 * t
   otalStates
559.                                  + currentGlobalId];
560.                      } else {
561.                          tmp = Q[direction1 * totalStates + cu
   rrentGlobalId];
562.                          if (V[currentGlobalId] < tmp) {
563.                              V[currentGlobalId] = tmp;
564.                          }
565.                      }
566.
567.              double totalChange=0;
568.              for (int direction1 = 0; direction1 < actions; di
   rection1++) {
569.
```

```
570.                 for (int is = 0; is < statesForLearningC; is+
    +) {
571.                     int index_ep_start = stateIdxForLearning[
    is];
572.                     for (int ca = 0; ca < anlgleDisc; ca++) {

573.                         int currentGlobalId = getGLobalStateI
    d(index_ep_start, ca);
574.                         int newStateId = transition[direction
    1 * totalStates
575.                                 + currentGlobalId];
576.                         double change = 0.5
577.                                 * (RVal[direction1 * totalSta
    tes + currentGlobalId]
578.                                         + gamma * V[newStateI
    d]
579.                                         - Q[direction1 * tota
    lStates
580.                                                 + currentGlob
    alId]);
581.                         Q[direction1 * totalStates + currentG
    lobalId] += change;
582.                         totalChange+=change*change;
583.                     }
584.                 }
585.             }
586.             cout << "iteration "<<it<<" Total change "<<total
    Change<<endl;
587.         }
588.         cout << iterations << " iterations  took " << gettime
    _() - startTime
589.                 << endl;
590.
591.
592.         std::ofstream os;
593.         os.open("q.out");
594.         for (int ac = 0; ac < actions; ac++) {
595.             for (int i = 0; i < totalStates; i++) {
596.                 if (i == 0) {
597.                     os << std::fixed << Q[ac * totalStates +
    i];
598.                 } else {
599.                     os << " " << std::fixed << Q[ac * totalSt
    ates + i];
600.                 }
601.             }
602.             os << endl;
603.         }
604.
605.         os.close();
606.
```

```
607.            return 0;
608.
609.        }}
```

Figure 5.3: Source code for Q matrix calculation.

BIOGRAPHY

The author, Haoran Zhang, was born in Fuyang, China on April 15, 1990. In July 2013, he obtained a Bachelor of Engineering degree in Mineral Processing Engineering from North China Institute of Science and Technology. Then, he joined the master program of Industrial System Engineering at Lehigh University.