Sensors & Transducers, Vol. 241, Issue 2, February 2020, pp. 13-28

www.sensorsportal.com

Sensors & Transducers

Published by IFSA Publishing, S. L., 2020
http://www.sensorsportal.com

Design of an Autonomous Greeting Robot

[1, 2] Yuze QIU, [1, 2] Ruoting LIAN, [1, *] Ishwar SINGH and [1, *] Zhen GAO

[1] W Booth School of Engineering Practice and Technology, McMaster University, Hamilton, ON, L8S 0A3, Canada

[2] School of Engineering, University of Science and Technology of China, No. 443 Huangshan Road, Hefei, Anhui Province 30027, China

E-mail: isingh@mcmaster.ca, gaozhen@mcmaster.ca

Received: 15 January 2020 /Accepted: 12 February 2020 /Published: 28 February 2020

Abstract: The paper focused on a portable selection which allows a robot at low cost to recognize guests before and guide them to their destinations. First, we begin with a brief description of the Greeting Robot and the methods introduction we used in our robot. After that, we focus on how to accomplish the movement control of the mobile robot platform. Specially, we discuss a lot of cases to find the better way to make the robot avoid crashing on its own. Then we also focus on the programing about how to complete the target of facial recognition and maintaining the database by itself and how to ensure the interaction between RaspberryPi and Arduino.

Keywords: Greeting robot, Mobility control, Machine Learning, Autonomous, Robot vision, Information acquisition.

1. Introduction

With development of technology in recent years, more and more companies and libraries choose robots as greeters and guides [1-5]. In many aspects, these robots can do a good job—they can not only let guests feel curious about the company or the library, but also show the strength of the company or the library. And several great works has been done in robots navigation, robot greeting, and human-robot communication based on face recognition. Most of existing systems assume unpractical or uneconomic structure for a robot greeter that a company can use, like using a big biped robot structure hanged in a room to ensure its stability; or don't have good enough system part to improve human-robot interaction, making robots just navigate guests to destinations without any kinds of communication; or even lack suitable mechanical structure.

Therefore, this paper is oriented towards providing an economic and simple robot greeter designing, which is not only able to navigate people to their destinations, but also acknowledge and greet every guest, who come to see it, on its own.

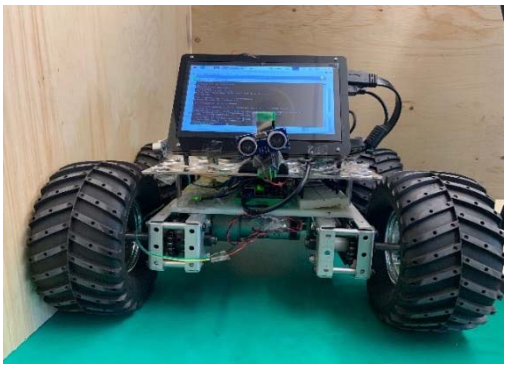
In this designing, we use a face recognition algorithm based on OpenCV library and python to trace face taken by camera and recognize it; use a four-wheel mobile car as a basic movement platform [6-9]. Furthermore, the face recognition program uses Cascade Classifier and LBPH Face-Recognizer to capture and output fasces' coordinate in the picture, and then using OpenCV Classifier to train faces and match faces with corresponding labels, and finally to predict and recognize faces' labels and names; and the four-wheel mobile car avoid crashing and turn right/left exactly with the help of ultrasonic sensors and gyroscope [10-12]. The main controller of the robot greeter contains RaspberryPi and Arduino----the former one is responsible for facial recognition and the later one controls movements of the robot.

This paper is organized as follows. In Section 2, the paper mainly talks about the hardware and mechanical structure of the robot; and in Section 3, we will explain the whole logic of the mobile car's motion control; and in Section 4, we will describe the way

how the mobile car plan the routine to its destination and avoid crash with obstacles in the roads; in Section 5, the facial recognition algorithm will be analyzed in detailed, include face detecting, face register and face recognition; in Section 6, how RaspberryPi and Arduino connect and collaborate with each other will be illustrated. Finally, in Section 7, the conclusion of the robot system designing is given.

2. Hardware and Mechanical Structure

In the article, the robot (Fig. 1) can lead the guest to the corresponding position in the map after the robot has recognized the face of the guest close to the robot and has been input a coordinate of the target position. In the process, the robot has an ability of automatic path planning and automatic obstacle avoidance.



(a)

(b)

Fig. 1. Mobile robot platform: (a) Front view, (b) Side view.

Robot is a wheeled robot whose structure can be roughly divided into two parts - the actuator part and the control part. The actuator part is at the bottom and consists of four wheels, two 12 V DC motors and a timing belt chain. The timing belt chain connects the front wheel with the rear wheel on the left side and the front wheel and the rear wheel on the right side to ensure that the front and rear wheels always have the same speeds. These two motors are respectively fixed to the wheels on both sides. The motions of the robot (move forwards, move backwards, left turn, right turn) changes with the motors' speeds. (Fig. 2 shows the bottom of the robot).

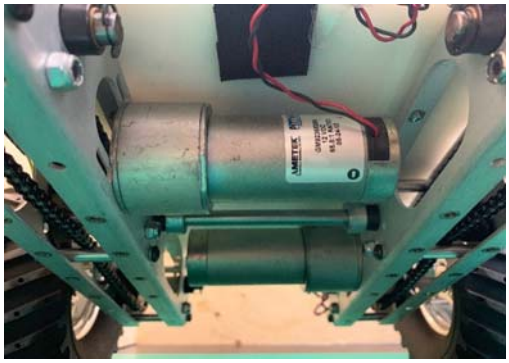


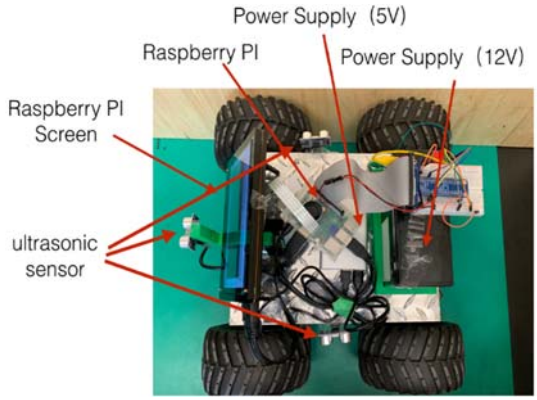
Fig. 2. The bottom of the robot.

The control element portion consists of an overhead layer and a top layer. An Arduino Uno board, a motor driver, a MPU6050 sensor, a breadboard and some lines connecting different components are placed on the overhead platform. A 12 V power supply, a 5 V power supply, a Raspberry pie board, a Raspberry Pi screen and a Raspberry Pi camera are placed on the top layer. The 12 V power supply supplies power to the motor driver to drive the motor. The 5 V power supply supplies power to the Raspberry Pi, Arduino and other sensors. The Raspberry Pi camera and the Raspberry Pi screen are connected to The Raspberry Pi for face recognition. The Raspberry Pi also receives the coordinate signals input by the computer. The remaining sensors are connected to the Arduino. The Raspberry Pi interacts with the Arduino via a USB cable. In the process, Arduino receives the feedback signal from each sensor and the coordinate signal sent by the Raspberry Pi. After planning and calculation, the corresponding PWM wave signal will be transmitted to the motor driver by Arduino. The motor driver then outputs the corresponding level signal to drive the motor to rotate at the corresponding speed to carry out the corresponding motion. (Fig. 3 shows the top view of the robot and the front view of the overhead layer).

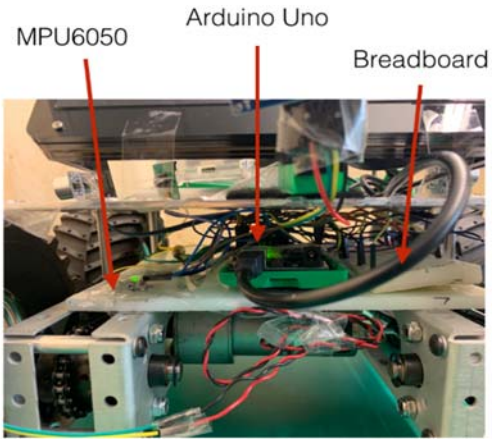
3. Motion Control

The basic motions of the robot includes moving forwards, moving backwards, left turn and right turn. Moving backward was not used in the test. In this article, the two motors respectively drive the left and right wheels to rotate at different speeds to carry out different motions. The speed of the motor depends on the duty ratio of the PWM wave transmitted by the Arduino board. The larger the duty ratio is, the higher the level, up to 12 V (supply voltage). In addition to

receiving the PWM wave signal transmitted by the Arduino, the motor driver also receives an enable signal for determining the rotating direction of the motor. There are two enable signals pin to control the steering of a motor. For example, when the two pin receive a signals as 1&0 (1 represents a high level, 0 means low level), the wheel is rotating clockwise, while the signal is 0&1, the steering is reversed.



(a)



(b)

Fig. 3. Hardware modules, (a) top view; (b) side view.

While the wheels on both sides rotate in the forward direction at the same speed, the robot goes straight. However, the performance of different pins on the motor driver or the performance of two motors is slightly different. Thus although the PWM signals of the two motors are controlled to be the same, the speeds of the two motors are inconsistent. We set two PWM signals the same first. It is found that as the robot moving forwards, the direction of the robot is always shifted to the left, which indicates that under the same input conditions, the speed of the right wheel is larger than that of the left wheel. In order for the robot to go straight, the PWM wave we set for the right wheel is slightly smaller than the left one. After many test adjustments, in the case that set the PWM signal with parameter 220 to the right wheel on the Arduino board, and set the signal with parameter 240 to the left one, the robot can go straight. (The parameter changing range is from 0 to 255.) In the case of moving back, it is only need to reverse the steering of the wheels in the forward direction.

Since the robot moving time is determined by the delay time given by the delay function in the Arduino program, the distance cannot be directly determined. Therefore, it is necessary to calibrate the relationship between the delay time and the robot's moving distance. In the study, we have given the delay time of 1000 ms, 2000 ms, 3000 ms, 4000 ms and 5000 ms, and measured five sets of data, each group repeats five times, taking the path average as the dependent variable and the delay time (in ms) as the independent variable. Perform a function fit. The fitting results show that the delay time and moving distance are linear.

$$s = a \cdot t + b \tag{1}$$

where t is the delay time (in ms) and s is the travel distance (in cm). a=0.0342, the uncertainty is 3.3547e-4. b=2.786, the uncertainty is 1.1126.

Considering that the same side wheel speed is the same, to turn left we should make the right wheel rotate faster than the left side. However, while testing, we have encountered a problem: when the rotation directions of the wheels on both sides are the same, due to the frictional force, the wheels with higher speed will be decelerated by the resistance, and the wheels with lower speed will be accelerated by the thrust. It turns out that the difference between the speeds of two side will be small, which means that the robot will require a large arc to turn left. Therefore, the approach we take is to make the left wheel stationary and the right wheel rotate forwards so that the robot can make a left turn in-situ. In order to do this, we need to give the left wheel a reverse force to counteract the thrust, so we reverse the left wheel. It should be a proper PWM wave signal to do this. If the values of the input PWM waves on the left and right sides are the same, it may cause the wheels on both sides to be locked, and the robot does not move. If the parameter value we set is too small, it will not change the phenomenon that the wheels on left side are still driven by the right wheel. After adjustment, it is finally found that while the PWM wave signal parameter value set to the left wheel is 40, and the right side is 240, the robot can turn left as we expected. In the case of turning right, you only need to reverse the left and right wheels.

Since we use the approach that one side of the wheels is stationary and the other side rotate to turn left/right, the wheels will inevitably slip during the rotation. Thus, it is unreliable to use the delay time to control the rotation angle of the robot. In actual measurement, it is found that the rotation angle will be greatly deviated under the same delay time, and occasionally the error value is even up to 40 degrees. Therefore, we apply the gyroscope MPU6050 sensor to ensure the accuracy of the robot rotation.

The MPU6050 sensor can measure the angular velocity of the robot during rotating. We can multiply

the angular velocity by the time to estimate the angle. However, the angular velocity is not fixed. To ensure the accuracy, we use the method of rectangular numerical integration. The angular velocity is measured every other short time, and the angular velocity is multiplied by the time interval and accumulated, and the sum will be compared with the target rotation angle (90 degrees). If the sum is smaller than 90 degrees, the rotation continues. When it reaches 90 degrees, the rotation is immediately terminated. In the study, we chose a time interval of 100 ms, which can guarantee the rotation accuracy. There is still a problem. Because the data got by the MPU6050 sensor is zero-biased, and the deviation value changes as the time goes, the solution we used to this problem in this study is to let the MPU6050 sensor measure a large number of angular velocity data and uses the average as the zero point every time the robot is stationary. The angular velocity used to calculate the corner is a relative value relative to this zero point. It should be noted that the use of the MPU6050 only guarantees the stability of each rotation angle and does not achieve accuracy. To achieve accuracy, we also convert the units measured by angular velocity. Since the zero elimination has been performed, we only need to multiply the value got by the MPU6050 by a constant. After many tests, when the constant is set to 0.0085, the angle the robot rotates can be kept at 90 degrees.

4. Path Planning and Obstacle Avoidance

Considering that the robot we designed only needs to work indoors with relatively simple functions, we expected to find a low cost and easy to implement method.

First, consider the most basic and simple case 1: When there is no obstacle around the robot, the robot is at the origin of the coordinate, and the target position is (a, b) in the first quadrant (Fig. 4).

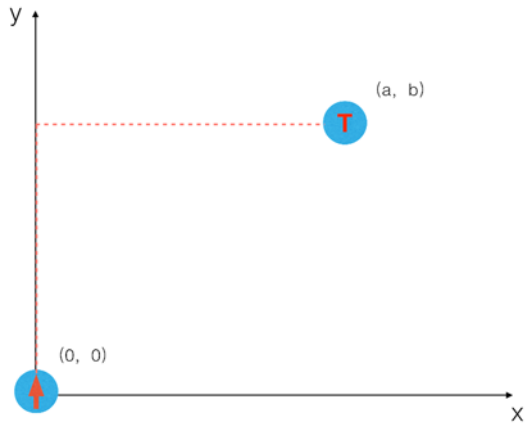


Fig. 4. Scenario 1.

The arrow in the blue circle represents the robot, and the direction of the arrow is the direction of the robot. The mark "T" in the blue circle represents the target position. The initial direction of the robot is toward the positive direction of the y-axis.

In this case, the robot only needs to move forwards y, then turn right and then move forwards x. The red dotted line indicates the driving path. When the initial direction is the positive direction of the x-axis, first move forwards x along the positive direction of x-axis and then turn left, and move forwards y. The coordinate of the robot's position is obtained by back-calculating the moving time into the linear function in the fourth part. In the actual experiment, we use the conversion of length to time. After defining the direction of the robot as move direction, the process can be represented by a block diagram shown in Fig. 5.

For the case that the robot is not at the origin or the end position is not in the first quadrant, as long as the end point coordinates and the robot position coordinates are subtracted, the relative coordinates are obtained. Then the quadrant of the relative coordinates is determined, and the judgment condition and the corresponding steering direction are changed according to the quadrant. Assuming that the initial position of the robot is (x0, y0). Add some obstacles in the range of $x < a$, $y < b$ based on case 1 (Fig. 6).

The blue box with the cross in the picture represents the obstacle. In this case, the ultrasonic sensor is used to detect the obstacle. When there is an obstacle in the path, the robot turns but always keeps moving toward the end point (x+ or y+). The red dotted line indicates the moving path. The process can be represented by a block diagram shown in Fig. 7.

However, the algorithm above has an obvious disadvantage that the moving region of the robot in the above algorithm never exceeds the range of $x <= a$, $y <= b$ and the robot does not move in the x- latter y-direction. Therefore when the obstacle is as follows: case 3, the robot will repeat turning left then turning right at the mark E but cannot avoid the obstacle (Fig. 8).

In order to avoid this, we set the robot to move forwards a short distance if there is no obstacle in front of it after each turn, and the judgement condition to stop and turn is that the side ultrasonic sensor does not detect obstacles besides. Then the robot will drive to the place marked with 1 in the figure below. In order to guarantee that the robot reach the target position finally, we only need to add a judgment after the robot reaches 1 to determine whether the robot is near the target position. If it is not, then repeat the previous steps until reach the target position. As shown in Fig. 9, the robot can reach the target position after repeating the steps before one more time after 1 is reached.

For the convenience of representation, we represent the calculation of the quadrant in the List2 Position Standards as the "direction calculation". The judgement of reaching the target position in x-axis direction or the y-axis direction is uniformly represented with "axis arrive", and the both arrival of the two directions is represented with "direction

arrive". The steering in List 3 How to Turn is uniformly recorded as turn.

Previously, our final position judgment can only guarantee that the robot finally reaches the area (in the first quadrant as an example) x >= a , y >= b, which means we can only guarantee that x, y are greater than a, b, but can not guarantee that the robot reach the target position. Therefore, it is necessary to add a judgment of |x-a|<= Δ and |y-b|<= Δ, which is represented with end arrive.

The process can be represented by a block diagram shown in Fig. 10.

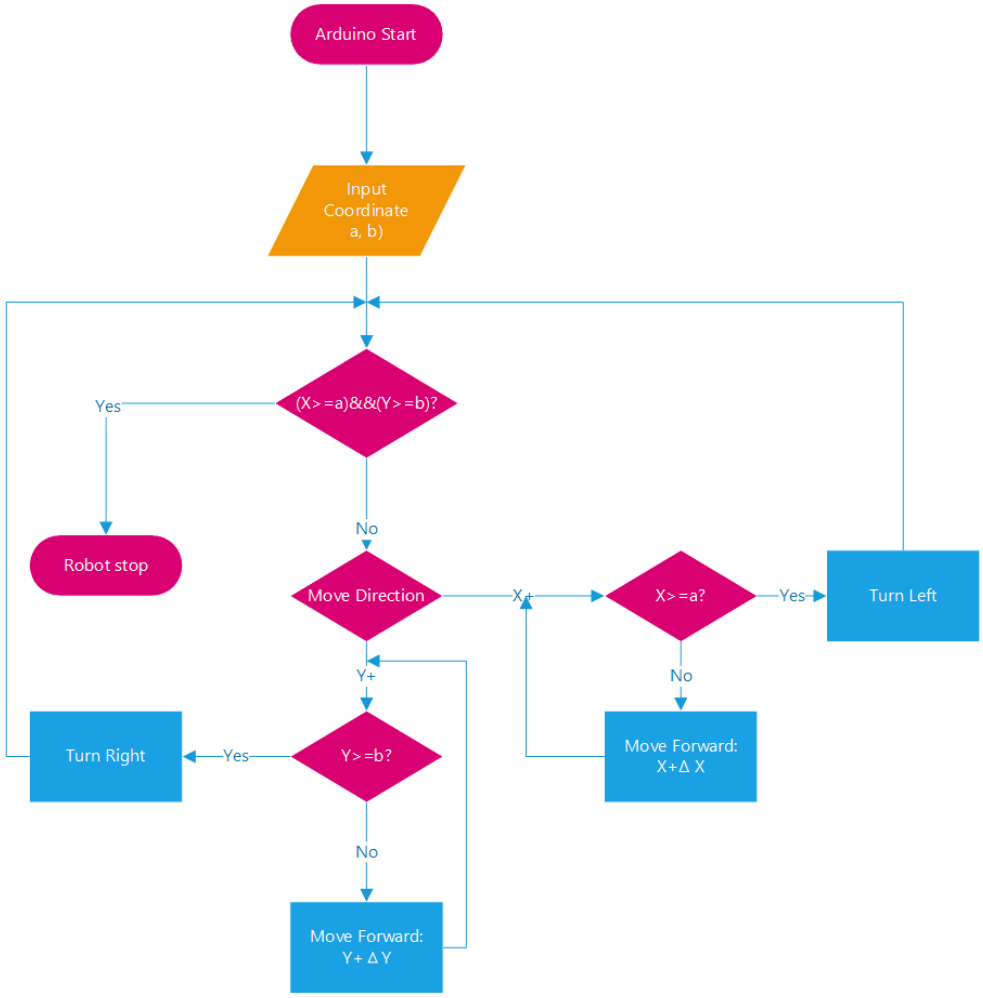


Fig. 5. Logical flow of movement control ver1. Δx = Δy is a unit length of the robot's advancement, corresponding to moving forwards for 100 ms.

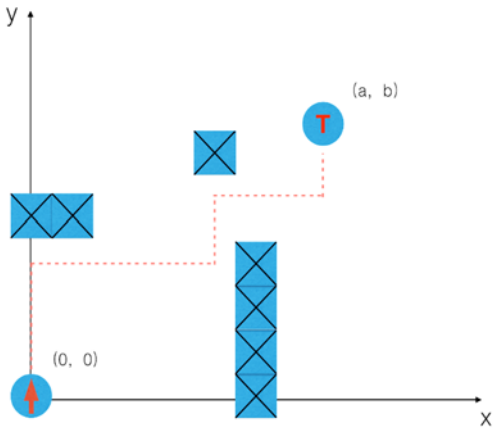


Fig. 6. Scenario 2.

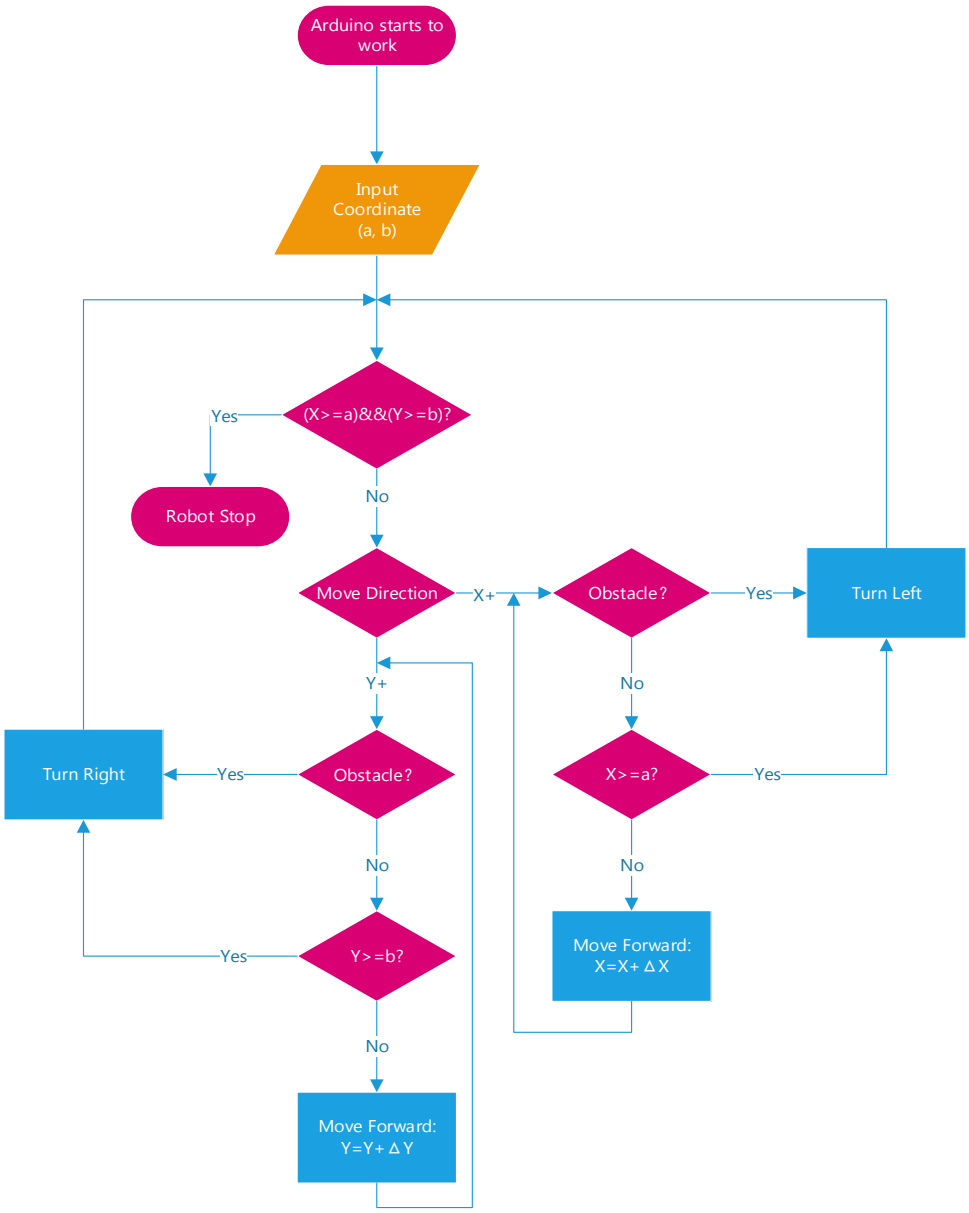


Fig. 7. Logical flow of movement control ver2.

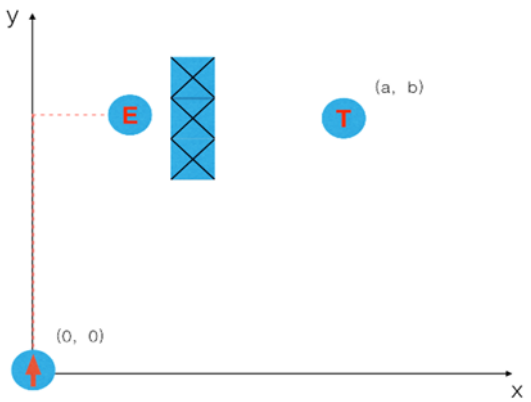


Fig. 8. Scenario 3.

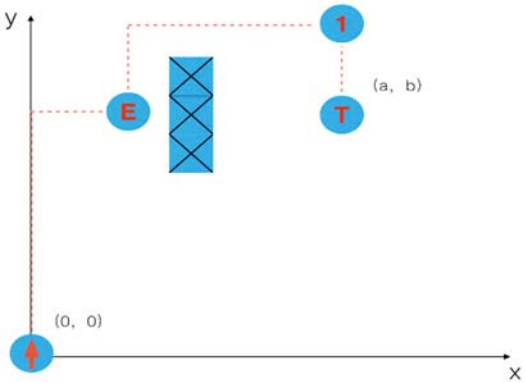


Fig. 9. Scenario 4.

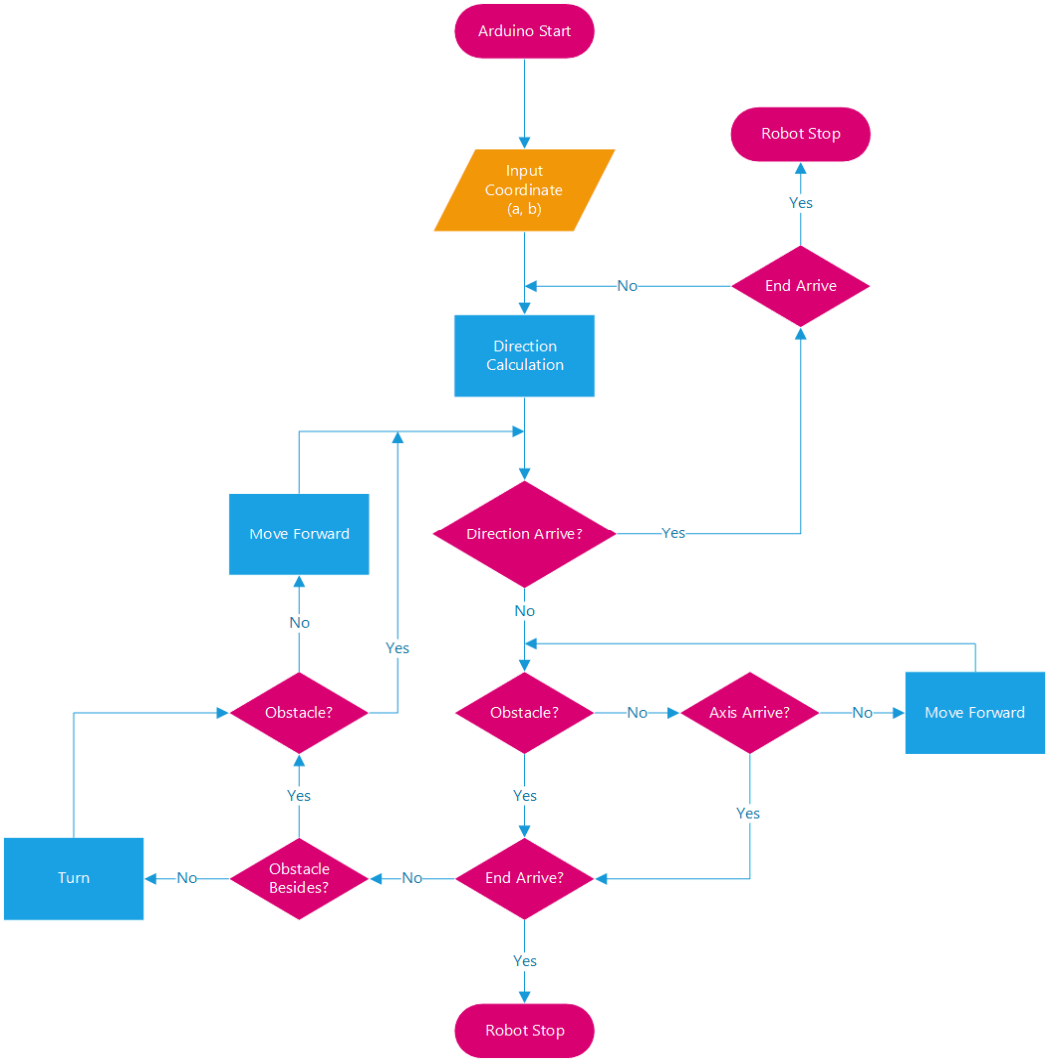


Fig. 10. Logical flow of movement control ver3.

"Move forward" means moving forwards a unit distance. Now, although the obstacle in case 3 can be avoided, but it always moves in one direction. Taking Case 3 as an example, the robot will only move in the y+ direction to avoid the obstacle, not in the y- direction. When the obstacle is as follows in case 4, the robot will repeatedly turn at E and cannot avoid the obstacle (Fig. 11).

For this we need an alternative that robot needs to move in the y- direction when the obstacle cannot be avoided in the y+ direction. When the ultrasonic sensor located at the front continuously detects obstacles twice while the robot has not moved, it can be known that the robot has reached a dead corner like E. Using the variable k to count, the value of k can be used for judgement. When a dead corner is encountered, the direction calculation will be performed again. The robot can be follow the dotted line trajectory in the following figure. Firstly, the robot reaches 1 point following the red dotted line, and then follows the green dotted line and avoids the obstacle to reach 2 points. Finally arrived at the target position. (Fig. 12)

At the same time, considering that the point on the x positive half axis is taken as a first quadrant point, when the dead corner is exactly has the same x ordinate as the target position, the robot still cannot avoid the obstacle because the recalculated target direction is still in First quadrant. Therefore, while the front ultrasonic sensor has detected obstacles four times continuously, which means the robot was stuck, it is necessary to change the judgement condition of the quadrant of target position. The final algorithm can be represented by a block diagram shown in Fig. 13.

This is the final algorithm used in this study, and path planning and obstacle avoidance can be achieved in most cases. However, an obvious problem still exists. When entering a dead end like a "U" shaped area, the robot is possibly to be stuck.

In order to verify the feasibility of the algorithm in the above section, we chose to experiment in a more regular indoor room. The structure of the room can be simplified by the Fig. 14.

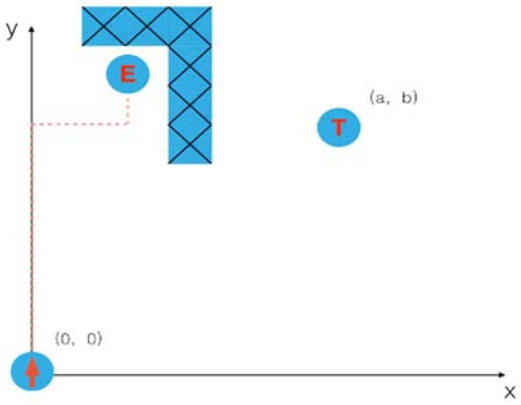


Fig. 11. Scenario 5.

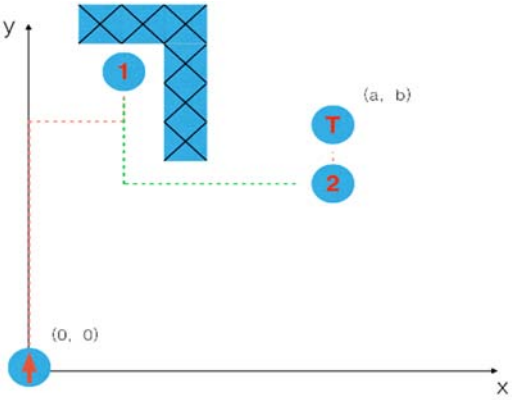


Fig. 12. Scenario 6.

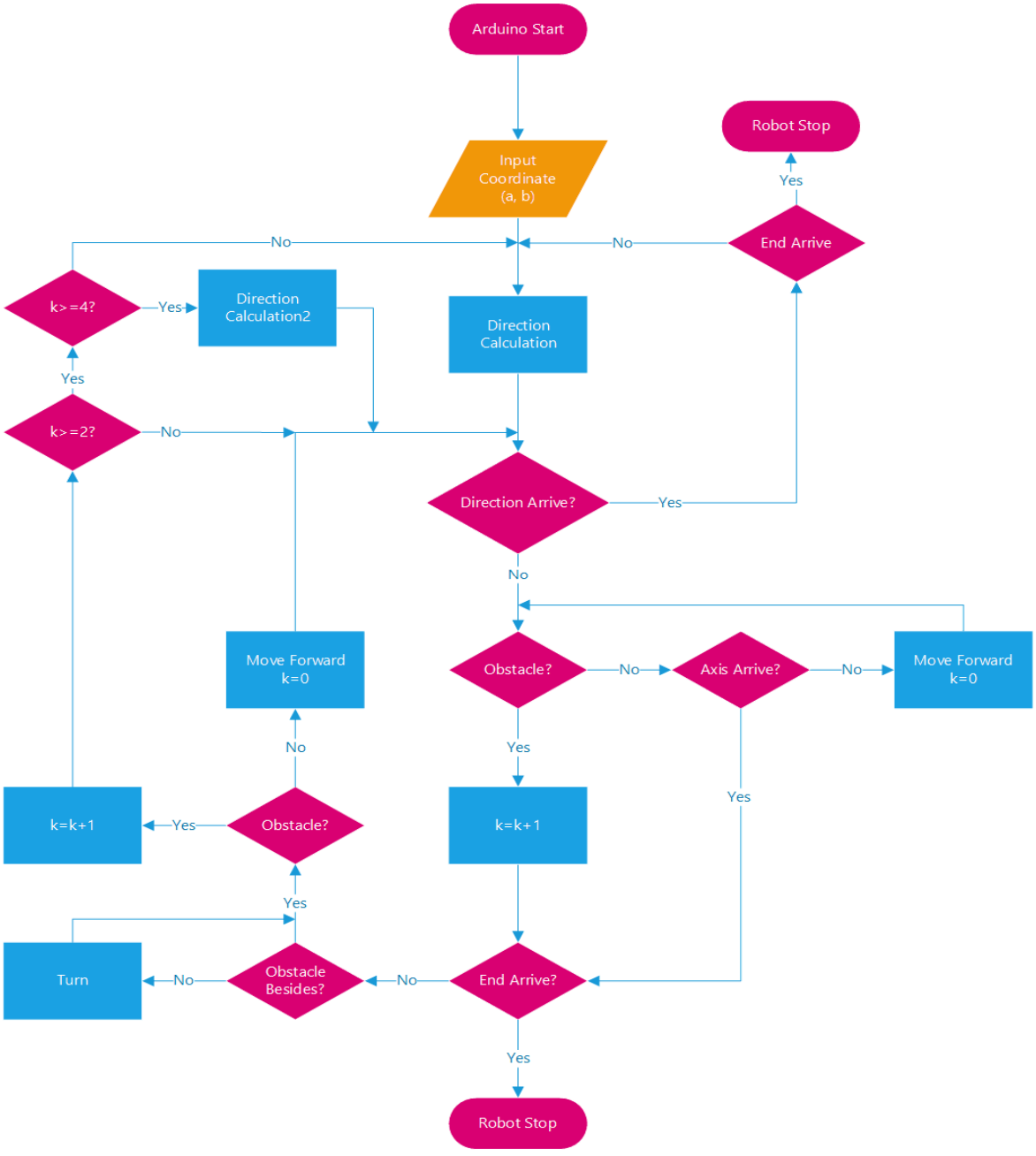


Fig. 13. Logical flow of movement control ver4.

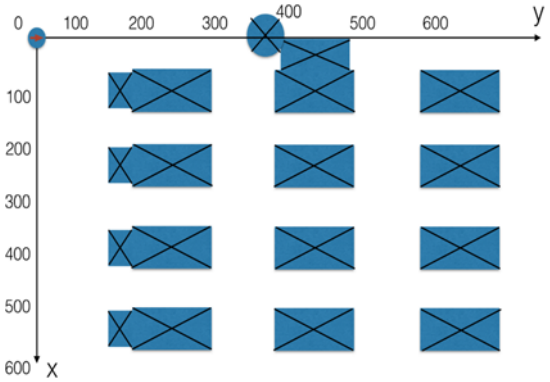


Fig. 14. Room map.

The gray box indicates the boundary of the room, and the blue cross is the fixed obstacle in the room. At the beginning, the robot is at the origin of the absolute coordinate system, facing to the y+ direction. The dimensions of the coordinate system are measured and the unit is cm.

We did the following experiments:

1) Experiment 1 (Fig. 15): Straight line test. Set the target points to (0,200) and (300, 0) for the robot starting from the origin.
   Experimental results: The robot reaches the target point along the red dotted line.
2) Experiment 2 (Fig. 16): Target point in the first quadrant (no need to avoid obstacles). Starting from the origin, the target point is set to (150,330).
   Experimental results: The robot reaches the target point along the red dotted line.
3) Experiment 3 (Fig. 17): Simple obstacle avoidance test. Starting from the origin, the target point is set to (150, 450).
   Experimental results: The robot reaches the target point along the red dotted line.
4) Experiment 4 (Fig. 18): Do not start from the origin. After the end of Experiment 3, the target point is set to (300, 550) without returning to the origin.

Experimental results: The robot reaches the target point along the red dotted line.

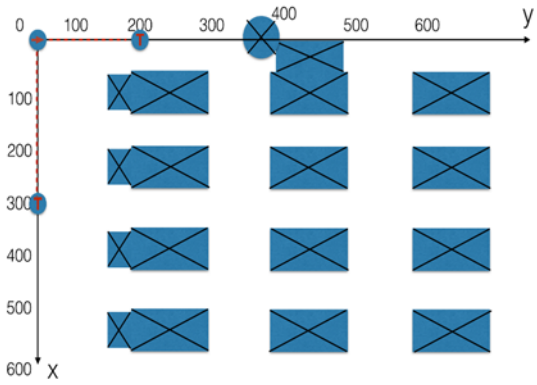


Fig. 15. First experiment's map.

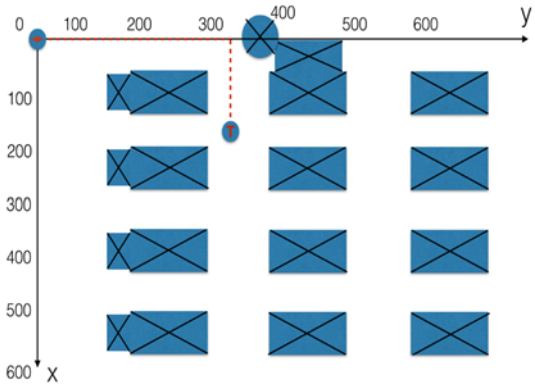


Fig. 16. Second experiment's map.

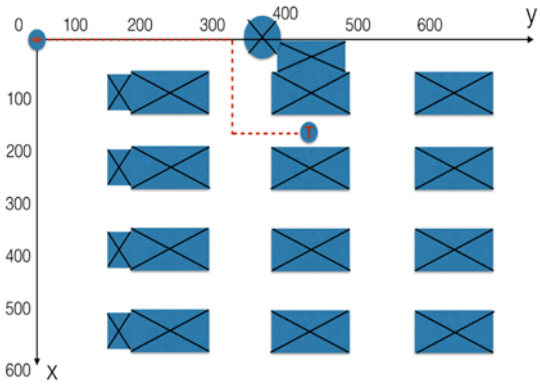


Fig. 17. Third experiment's map.

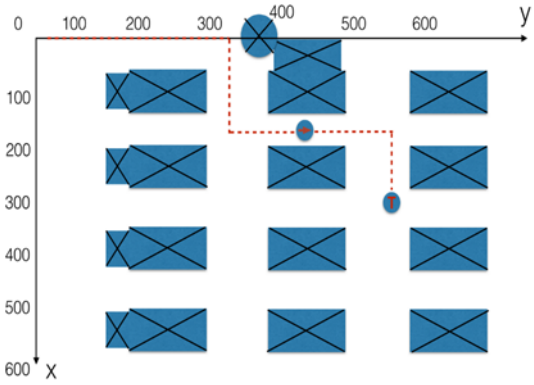


Fig. 18. Fourth experiment's map.

5) Experiment 5 (Figs. 19-22): In order to verify the ability of the robot to successfully plan a feasible path under different obstacle conditions. We experimented with changing the obstacles with the origin as the starting point and (150,550) as the end point.

Case 5-1 (Fig. 19): Basic situation, no more obstacles were added.

Experimental results: The robot reaches the target point along the red dotted line.

Case 5-2 (Fig. 20): Add a plate to block the original path. The yellow part is the added board.

Experimental results: The robot reaches the target point along the red dotted line.

Case 5-3 (Fig. 21): Add a horizontal plate to block the original path based on Case 2.

Experimental results: The robot reaches the target point along the red dotted line.

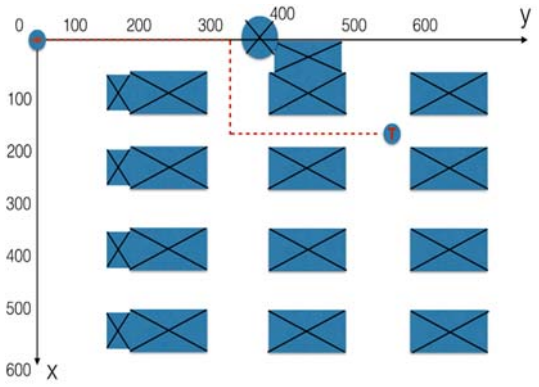


Fig. 19. Fifth experiment's map.

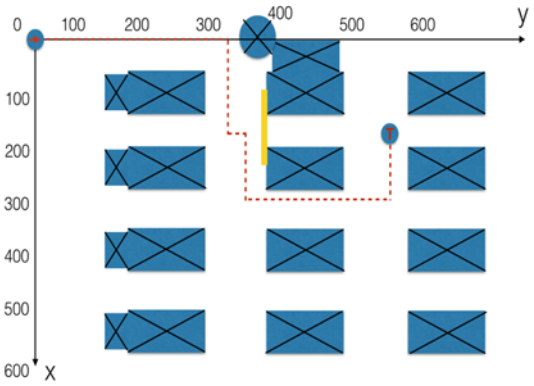


Fig. 20. Sixth experiment's map.

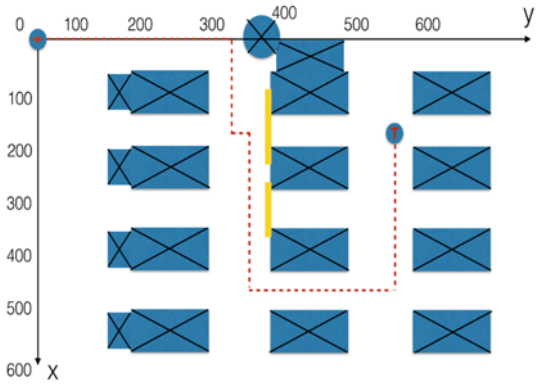


Fig. 21. Seventh experiment's map.

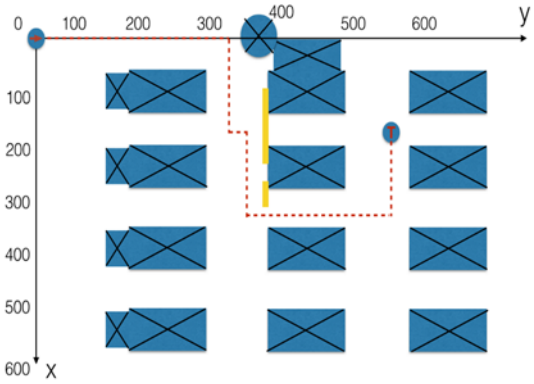


Fig. 22. Eighth experiment's map.

(6) Experiment 6 (Figs. 23-25): change the obstacle with the origin as the starting point and (300,550) as the target point.

Situation 6-1 (Fig. 23): Basic situation, no more obstacles are added.

Experimental results: The robot reaches the target point along the red dotted line.

Case 6-2 (Fig. 24): Add a broad to block the path.

Experimental results: The robot reaches the target point along the red dotted line.

Case 6-3 (Fig. 25): On the basis of the 6-2 case, add a plate that blocks the x direction.

Experimental results: The robot firstly reaches 1 along the red dotted line and finally reach target point along the dotted line. The two red dotted lines at 1 are actually coincident, and are drawn separately in the order for convenience.

Experiments 1, 2, 3, and 4 test some basic functions for auto-navigation. The experimental results show that the robot can achieve these basic functions well. Experiments 5 and 6 test the auto-navigation ability of the robot in more complicated situations. The experimental results show that the robot can plan the reasonable path and achieve auto-navigation well in the case given by the experiment.

Case 5-4 (Fig. 22): Reduce the length of the board in case 3, which means the path between the obstacles is partly but not completely blocked.

Experimental results: The robot reaches the target point along the red dotted line.

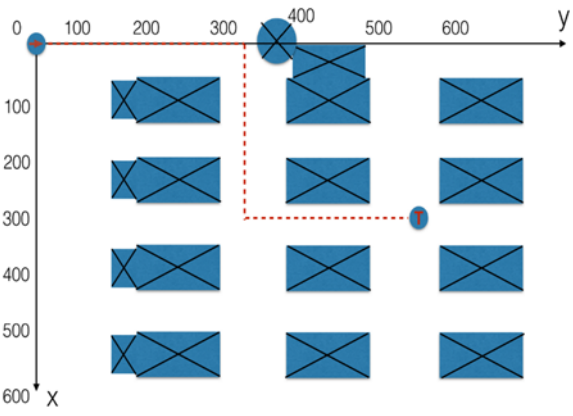


Fig. 23. Ninth experiment's map.

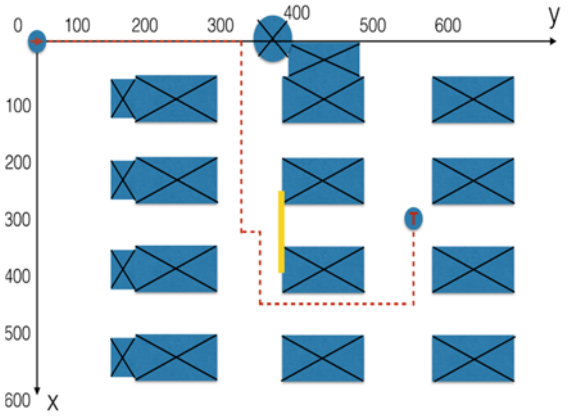


Fig. 24. Tenth experiment's map.

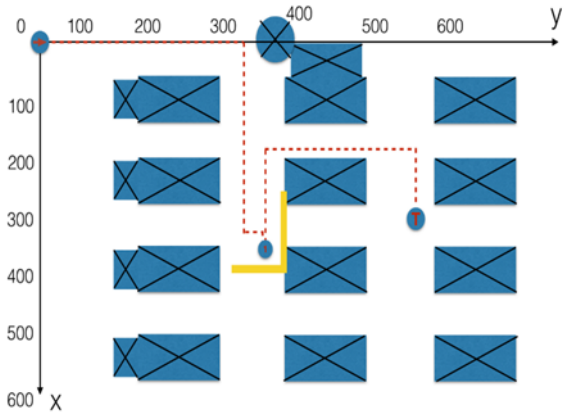


Fig. 25. Eleventh experiment's map.

5. Identity Recognition

Because the greeting robot has the need for face recognition, in order to better improve the greeting robot's performance in human-robot interaction, a facial recognition system with quick response, high accuracy and good human-machine interface is essential. After investigation, it is found that there are two main ways of face recognition now: first, training a deep learning model to achieve the goal of face recognition through a deep learning model trained by ourselves. When based on a premise of large database, this method has high accuracy and a better response speed. But in return, this method requires huge computation and the equipment requirements are also strict. The need of better GPU for model training and the completed deep-learning model is often large, whose requirement is slightly insufficient to meet for the computing power of RaspberryPi. So it is not appropriate to use deep the deep-learning model method in RaspberryPi. Second, adopting OpenCV into writing the Facial Recognition Program. OpenCV - an open source Computer Vision Library in C++ developed by Intel, which has a better Python interface, and can be also called directly in Python, in addition that the function of its face recognition section is relatively mature and has a high degree of practicality. Taking into account that the Raspberry Pi has a better performance in Python usage and OpenCV operating environment. Therefore, the robot in the paper adopts this method, that is to say, using Python's OpenCV library to achieve the face recognition on the RaspberryPi. The following systems are programmed on Python by default.

The basic idea of the face recognition system is: the Picamera captures the image → captures faces in each frame → recognizes the face and determines whether the face in the database is in line with the captured face → record the new face or ask for the help if needed → generates a new face-recognition-training file → reload the training file → start the next round of face recognition, based on the new database and the new training file.

The logical flow chart of the overall system is shown in (Fig. 26).

In the Facial recognition system, the recognition of the face is based on the Haarcascade Classifier in the OpenCV library, and the corresponding operations of face recognition is carried out by using the internal functions of OpenCV's LBPH Face Recognizer such as save, load, predict and other functions to realize the memory of face-recognition-training file, face-recognition-training file loading, face accuracy evaluation and other functions.

The system also adds a function that will only capture the largest face in the picture, a section to check the size of faces in the database and delete the data with abnormal size, a function to permit a certain number of prediction errors and other measures to prevent system from abnormal errors/data, which may lead to anomalous termination of the system. These sections is aimed at maximizing the practicality of the facial recognition system and the performance in human-robot interaction.

5.1. Pretreatment of Images

RaspberryPi uses OpenCV's command "VideoCapture" to read the video stream, transmitting each frame of footage read by the Picamera to the Raspberry Pi for processing. In order to facilitate the user to view, the frame is mirrored to get the picture "img", which is displayed to the screen. And then through the "cvtColor" function, the picture is converted from the BRG color space to the gray color space, generating the picture "gray" to facilitate subsequent processing. The picture "gray" is then probed for human faces, and system will place into the list "face" the length and width and diagonal points' coordinates of the rectangular area in which all the faces in "gray" are located. The LPBH operator in OpenCV's cascade classifier is used here to detect faces.

This operator locates the exact areas with the characteristics of human faces, mainly by using the binary pattern processing to get the feature graph of the picture. Since there will be more than one customer in the screen/picture when the robot works in the actual workplace, which means that it is necessary to

assess the detected faces remove no-target faces and maintain the target faces. In this paper, it is assumed that in the process of facial recognition because there is only one target of service in one service there will be only one target face in the screen that occupies the largest area in the frame, that is, the length and width value of the target face is the maximum value in the list "face", the output of each face-detection round. The image pre-processing function "get_faces" filters out the maximum value using the bubbling sorting method, and then use the target face, picture "gray", and picture "img" as its return value.

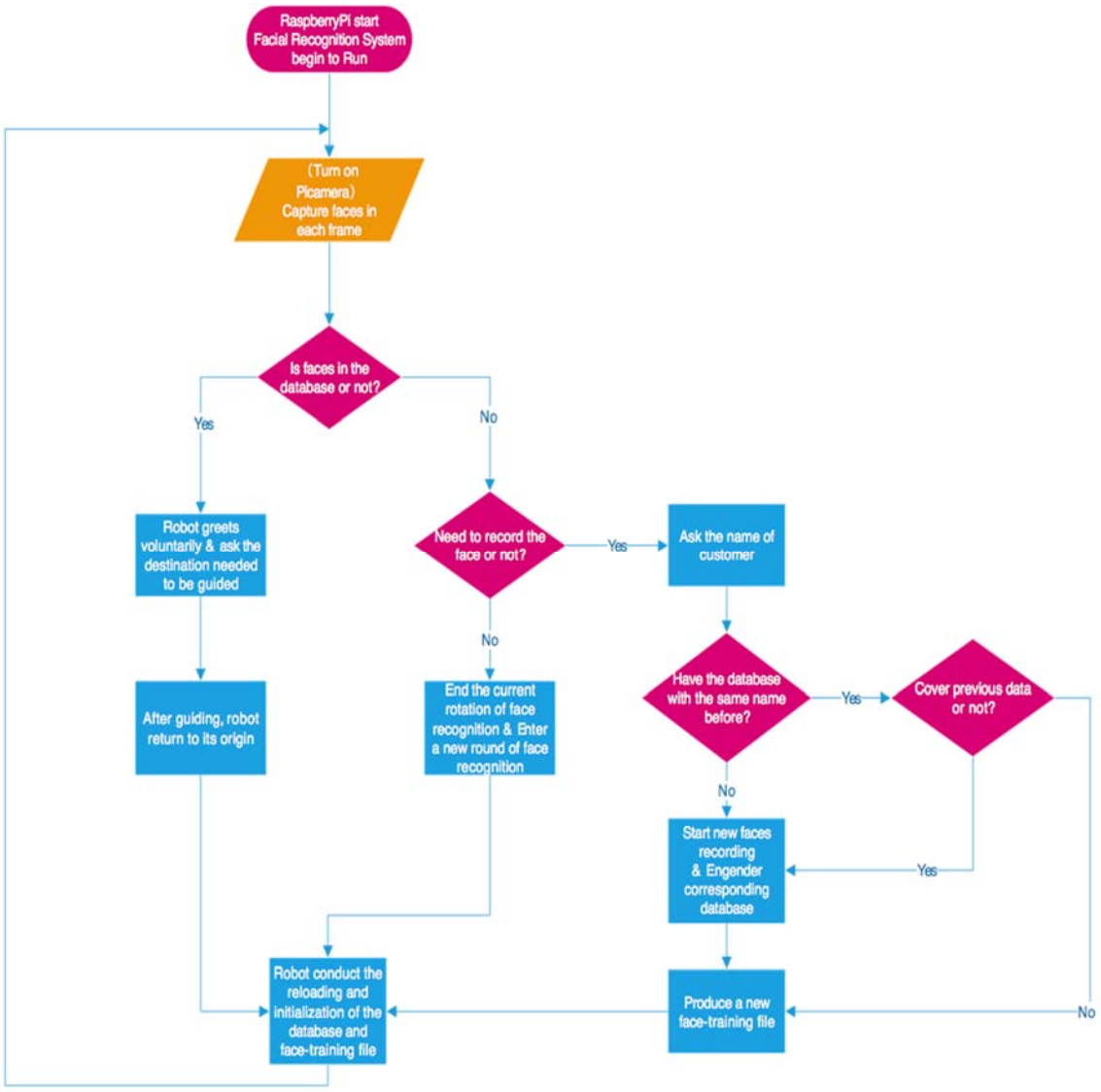


Fig. 26. Logic of Facial Recognition System.

It should be noted here that due to the limit to pixels of the camera and complex ambient lighting conditions although LPBH operators can theoretically greatly reduce or even eliminate the impact of ambient lighting on face detection, the system would still meet errors when capturing faces due to the effects of surrounding lighting condition. For example, the target face cannot be captured or the system registers others' faces as target faces (other people's faces around the target face, are usually smaller than the target face in size). So the images' size in newly established database need to be checked after building a new database and then the system should delete all the inappropriate data.

5.2. Prediction

The core of this section is prediction of faces' corresponding names and confidence probability of the detected face using the OpenCV's module "createLBPHPhARecognizer". Before making the prediction, the pre-trained face-recognition file "trainer.yml" is loaded by the function "load" of this module to obtain the characteristic matrix, which contains important reference between faces and labels which their corresponding names refer to. And then the captured faces are predicted using the function "predict" of the module. The function "predict" makes predictions based on the loaded feature matrix,

returning the predicted label and the expected distance value, which is the associated confidence value between the input picture and the picture corresponding to the predicted label. Finally, the predicted name and confidence rate will be displayed on the rectangle surrounding the face in video, and the confidence rate is taken as the desired distance minus the difference of 100, because the confidence rate fluctuates up and down on a 100 basis.

5.3. Adding a New Object

The function of adding a new face is mainly divided into the following parts to achieve: entering the user's name, determining whether the user's database has already existed, asking whether to overwrite if previous database exists, re-recording the face/ skipping the recording step, checking the size of data and database.

When creating a guest's database, the system will create a corresponding directory under the database's designated directory using guest's name as the folder name and store into it the pictures of the face obtained in the pre-processing section of the image, "get_faces". When the program restarts, system can get the existing users' names by traversing the subdirectories under the specified directory. Each user's name is assigned a corresponding id (label), i.e. the label used in training, and the picture data is stored in the form of using its corresponding id as the prefix filename, so that the user's corresponding id/label can be easily obtained by reading the picture's filename under each user's directory, his or her database. After the current user's data is stored, the program begins to examine the database, removes too small images, and continues to re-record face data until the target number of data is met. This checking function is done by the function "check_size": the function determines the current database size by the number of sub-files in the user's directory, and obtains the stored face-characteristic data (diagonal points' coordinates, length, width) again through the LPBH operator in OpenCV's Cascade Classifier. The maximum values of the length and width in the face feature data are obtained by the bubbling sorting method. Under normal circumstances, because the distance between the user and the Picamera during recording human faces (taking photos) is basically unchangeable and stable, the target face's size will not change too much, that is, the face data whose size has small difference with the maximum length and width in list "face" can be regarded as normal data to preserve. In addition, due to the limitations of the actual picture (how large area Picamera can take), through trial and error, we can also fix a difference value x to confirm and exclude the abnormal data in a database, our difference value is about 50, which is an empirical value and could be modified with the specific experimental conditions. And face data whose difference between its size and the maximum length/width (absolute) is less than the difference value can be considered as the abnormal data to be deleted.

5.4. Generate a New Trainer File

The core of this section is the use of OpenCV's createLBPHFaceRecognizer module for face recognition training and the output of the corresponding face-training files. First, by the loading function "gettheImageAndLabels" to get the faces data in the existing database and their corresponding ids (labels) the same as the function "check_size", the loading function get all the guests' names, labels and faces data by traversing the subdirectory under the database's designated directory, that is, users' directories and traversing the picture files under the user directories. And then program transforms the face data (face pictures) into a form of unit8, a kind of numpy format, to satisfy the limitation in input of the function "train", a function of createLBPHPhFaceRecognizer module. And the transformed faces data and corresponding labels will be stored into two lists "facesamples" and "ids", which are used as the return value of the function "gettheImageAndLabels". Then the "facesamples", "ids" will be inputted into function "train" of createLBPHFaceRecognizer module as parameters. The completed face-training file is then saved to the specified path through the function "save" of the createLBPHFaceRecognizer module.

5.5. System Optimization

Considering that in the actual process of use, the robot's facial recognition may meet some situations of identification error due to environmental factors. So it is necessary to identify whether the confidence rate is too small or there is too much prejudgment error, so that the robot can independently decide in these cases whether it needs to re-identify. The system in this paper sets the maximum times of errors to four, and the lowest confidence rate to 20 % once the prediction confidence rate is less than 20 %, the system will prepare for the next round of face recognition, and reset the number of errors to zero when the times of prediction error reaches four times.

5.6. Experiment

The experimental environment adopts the normal indoor lighting, outdoor sunlight as two conditions, and the target number of nine people, that is, the robot will be based on the original five-people database to add and identify the new nine different faces data; the number of faces appearing in the screen increases from one to three (only one target/maximum face); and each face was recognized ten times to measure the system's accuracy in facial recognition.

In the first experiment, we test the system's ability to adding a new face indoors, including the speed to react to the new adding face, how long to train a database and the accuracy of the recognition in the new face, for the purpose of simulating the actual work environment, which is usually in a large and bright room.

In the second experiment, the system is tested in both indoor and outdoor environment, with the number of faces in screen increasing from one to three (only one target face), so as to test its accuracy in finding the target face and facial recognition, and the resistance to light conditions.

The experimental data are shown in (Table 1):

Table 1. Experimental data.

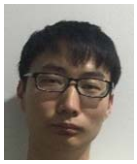
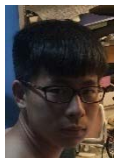
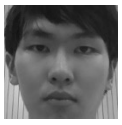
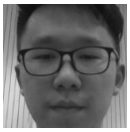
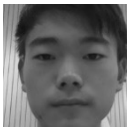
Number	Volunteer photos	Recognition times before recording	Recorded face examples (indoor)	Training time	Confidence rate in First time
1		4		70 s	62.0 %
2		1		40 s	57 %
3		4		70 s	65.0 %
4		4		40 s	43 %
5		4		60 s	76 %
6		4		50 s	54 %
7		4		90 s	71.0 %
8		4		40 s	70 %

Table 2. Multiple Faces Test (one target face at a time).

How many faces in the screen at the same time (only 1 target face)	1		2		3	
Environment light	Light indoor	Sunlight outdoor	Light indoor	Sunlight outdoor	Light indoor	Sunlight outdoor
Capture accuracy			7/10	8/10	5/10	6/10
Recognition accuracy	10/10	10/10	7/7	8 / 8	5/5	6 / 6

As can be seen from the above experimental data, in the actual operation, the system shows high accuracy and speed of reaction. For the recorded faces or newly recorded faces has good accuracy of recognition and the recognition confidence rate of newly recorded face in first time is higher than 50 % in most tests, the accuracy of recognition after recording is also close to 100 %. The time required from recording a new face to the generation of new face-training file is also distributed at about 1 min, which shows good practicality and speed of reaction.

However, the system, when working in the uneven ambient light condition or meet some people's faces, is easy to identify the face slowly, or even difficult to recognize the face. The later improvement in this system can focus on rewriting the program by C++, which is the original programming language of OpenCV and would have better speed of reaction; or ameliorating source code of OpenCV's CascadeClassifier for further stability and denoise, so as to improve the performance of the system.

6. Complete Test of the Greeting Robot

6.1. Interaction between RaspberryPi and Arduino

After building the RaspberryPi and Arduino's respective systems, it is now necessary to add the interactive ports to both systems so that these two microcontrollers can exchange with each other commands and information. Through investigation, it's found that RaspberryPi and Arduino could be connected in two ways: one is through serial communication, but this method needs conversion of voltage level, otherwise would result in the burning of both microcontrollers, since voltage level of output of RaspberryPi and Arduino is different. And the other is connection through USB. Since the new RaspberryPi has added a USB interface to the serial port, it is possible for Arduino and RaspberryPi to communicate and connect more stably and conveniently via USB. So, this method is chosen to connect RaspberryPi and Arduino in this article.

Because of the difference in hardware and speed between serial communication of Raspberry Pi and Arduino, the way that RaspberryPi and Arduino exchange information requires a special logical structure and code form when the serial information is transited via USB. Logically, we use a continuous loop approach to ensure the accuracy of information interaction between RaspberryPi and Arduino, i.e. the current loop is ended or specific work instructions start only when both the RaspberryPi and Arduino receive the corresponding instruction characters transmitted by themselves individually. With a pre-established instruction-to-motion list, RaspberryPi can easily direct Arduino to perform the expected action. And subsequent addition of new instructions can be achieved easily through adding a new round of sending & reading specific instructions to both RaspberryPi and Arduino on the original basis, which shows great exploitability.

The logic flow chart of interaction system between RaspberryPi and Arduino is shown in Fig. 27.

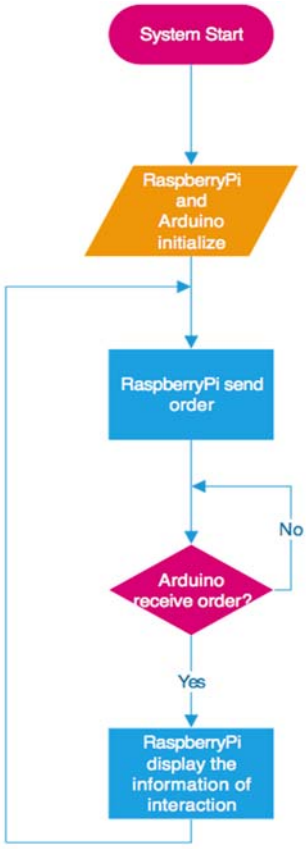


Fig. 27. Interaction between RaspberryPi and Arduino.

6.2. The Overall Test

Add the corresponding interfaces of the interaction system designed in the previous section for RaspberryPi and Arduino to their respective systems, and then connect the two microcontrollers with a USB cable. Connect the two microcontrollers to the electricity supply at the same time and run the code on the terminal of RaspberryPi.

Throughout the robot's operation, the RaspberryPi's facial recognition system and Arduino's motion control system run well to achieve the desired goal, and the interaction system can communicate the information between Raspberry Pi and Arduino accurately.

7. Conclusions and Future Work

Although the Greeting Robot was able to recognize faces and navigate safely and effectively in our experiment under ideal localization, our experiments do not guarantee that our Greeting Robot will perform the same way under real conditions. Additionally, the simulated environment did not contain other obstacles, such as chairs and trash bins, which could have deterred the robot's performance. And the experiment's light conditions couldn't reflect real light conditions in working place, which may influence the performance of the Facial Recognition System. However, since our simulated environment was the actual conditions in the lab, the Greeting Robot will probably perform the same under real conditions when it works.

Further work consists of figuring out a method that allows RaspberryPi to acknowledge the voice and transform the voice into corresponding characters, testing the robot under real conditions and adding a function that could check whether it is followed by guests. We also plan to test the entire Greeting Robot project, complete with face and voice interaction, after we are satisfied with the robot's recognition and navigation around the lab.

References

[1]. Zeynep Barlas, When robots tell you what to do: Sense of agency in human- and robot-guided actions, *Consciousness and Cognition*, Volume 75, October 2019, Article 102819.

[2]. Marlena R. Fraune, Benjamin C. Oisted, Catherine E. Sembrowski, Kathryn A. Gates, Selma Šabanović, Effects of robot-human versus robot-robot behavior and entitativity on anthropomorphism and willingness to interact, *Computers in Human Behavior*, Vol. 105, April 2020, Article 106220.

[3]. Ravi Teja Chadalavada, Henrik Andreasson, Maike Schindler, Rainer Palm, Achim J. Lilienthal, Bi-directional navigation intent communication using spatial augmented reality and eye-tracking glasses for improved safety in human–robot interaction, *Robotics and Computer-Integrated Manufacturing*, Vol. 61, February 2020, Article 101830.

[4]. Christof Mahieu, Femke Ongenae, Femke De Backere, Pieter Bonte, Pieter Simoens, Semantics-based platform for context-aware and personalized robot interaction in the internet of robotic things, *Journal of Systems and Software*, Vol. 149, March 2019, pp. 138-157.

[5]. Sangseok You, Jeong-Hwan Kim, Sang Hyun Lee, Vineet Kamat, Lionel P. Robert, Enhancing perceived safety in human–robot collaborative construction using immersive virtual environments, *Automation in Construction*, Vol. 96, December 2018, pp. 161-170.

[6]. Hiroki Sugano, Ryusuke Miyamoto, Highly optimized implementation of OpenCV for the Cell Broadband Engine, *Computer Vision and Image Understanding*, Vol. 114, Issue 11, November 2010, pp. 1273-1281.

[7]. Abdillah Komarudin, Ahmad Teguh Satria, Wiedjaja Atmadja, Designing License Plate Identification through Digital Images with OpenCV, *Procedia Computer Science*, Vol. 59, 2015, pp. 468-472.

[8]. Kehua Guo, Yan He, Xiaoyan Kui, Paramjit Sehdev, Jialun Li, LLTO: Towards efficient lesion localization based on template occlusion strategy in intelligent diagnosis, *Pattern Recognition Letters*, Vol. 1161, December 2018, pp. 225-232.

[9]. Ippei Torii, Kaoruko Ohtani, Naohiro Ishii, Study and Application of Detecting Blinks for Communication Assistant Tool, *Procedia Computer Science*, Vol. 35, 2014, pp. 1672-1681.

[10]. U. Rajendra Acharya, Wei Lin Ng, Kartini Rahmat, Vidya K. Sudarshan, Kwan Hoong Ng, Shear wave elastography for characterization of breast lesions: Shearlet transform and local binary pattern histogram techniques, *Computers in Biology and Medicine*, Vol. 911, December 2017, pp. 13-20.

[11]. Mahmood Sotoodeh, Mohammad Reza Moosavi, Reza Boostani, A novel adaptive LBP-based descriptor for color image retrieval, *Expert Systems with Applications,* Vol. 1271, August 2019, pp. 342-352.

[12]. Xuan Wang, Junhua Liang, Fangxia Guo, Feature extraction algorithm based on dual-scale decomposition and local binary descriptors for plant leaf recognition, *Digital Signal Processing*, Vol. 34, November 2014, pp. 101-107.