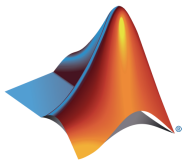
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY



*Shanghai Jiao Tong University*

*Department of Electronic Engineering*

# Face Detection and Human Tracking Robot

*Author:*
Xiaoheng Xia
Jianglong Li
Jialin Yang

*Supervisor:*
Yuhong Yang
Yueyi Xu

An Assignment submitted for the SJTU EI345:

*MathWorks Excellence in Innovation Projects*

May 31, 2022

# Contents

# 1 Introduction

## 1.1 Problem Background

Human-robot interaction is essential in many computer vision applications, including activity recognition, automotive safety, intelligent home security applications, and surveillance. Robots' task is to be assistants that help people do their jobs. Robots must be able to interact and communicate well with humans.

For this condition, face tracking systems become the main requirement for vision systems for this type of robot. Face detection can improve the human-robot interaction of a robot. Automatically detecting and tracking humans with sensors or algorithms is a challenging problem because of the diversity of locations, the complexity of the system, and finally, a complex optimization problem. The process includes extraction, selecting the best features, and then tracking.

## 1.2 Application Scenario

The application scope of facial recognition car is defined as follows: in a properly sized confined space, people are randomly distributed, and then point-to-point contact is required. This definition includes the following three aspects: considering the camera range of a single car, resolution and storage space, we believe that closed places are the main application point of face recognition car; Because the car has the function of face recognition, so we can reduce the distribution constraints of the crowd, can be randomized distribution; The car can be moved, for the larger car can also transport related materials. Then there is an opportunity for point-to-point contact.

Theoretically, we believe that all the application scenarios that meet the above identification range can be applied to face recognition cars. But to make the example concrete, we built an ideal restaurant model. We assume that the car can carry a certain amount of common restaurant tools, such as new chopsticks, disinfection towels, etc. There are traces in the restaurant, which are suitable for the indoor layout of the restaurant and take care of the whole range as far as possible. The car moved slowly along the track. When a face appears in the lens, and its distance to the car is less than a certain threshold value, the car will stop tracking, and through the face detection function to ensure that its own face. Then the car will wait for 10 seconds to listen to the customer's voice commands, such as "chopsticks" and "towels", the car can provide the customer with the corresponding items. If there is no command within 10 seconds, or if the face leaves a certain range, the car will automatically travel along the planned path.

The smart car we design has the following advantages:

Undertake work as a substitute Labour force. At present, due to the epidemic, labor shortage, rising costs and other factors, there is a serious labor shortage in the catering industry. And this idea can solve the problem of manpower shortage of catering industry very well. We observed that in general the restaurants, the waiter has a considerable part of the effort spent on the above to provide customers with the commonly used tools, so, if we are smart car can provide similar services, substitute the waiter waiter can spend effort more in need of his place, thereby reducing the cafe demand for the number of employees, ease the labor shortage.

Face recognition and speech recognition dual function combination. According to the investigation, the current catering service robots on the market are basically based on the commands issued by the backstage person, combined with the camera to provide customers with food delivery services. But the smart car does not need background manual instructions, and combined with face detection and voice recognition dual functions, can provide more intimate service to customers.

But what have to admit is, the car working environment that we envisage is still quite ideal. Specifically, there are the following aspects to consider: One is the existence of restaurant background sound. The restaurant is full of cheers and loud voices. It is easy for customers who need service to put forward orders to be disturbed by the background sound of the restaurant, and the car is easy to misjudge. Second, even in a small closed environment, the environment of the restaurant is complicated, so the car may face some extra emergencies in order to run normally. We believe that obstacle avoidance can be added in the process of tracking; Third, the height of the car itself and the freedom of camera rotation limit the scope of its exploration, which is mainly caused by hardware. If we had a chance to put it into practice, we could move the camera up, increase the range of freedom and so on. In the actual simulation, we consider to further reduce the distribution of people, so as to facilitate the realization of car tracking face.

# 2 Detection on Pi – System Design

## 2.1 Face Detection

One key factor in guiding the car is the presence of faces. To detect that characteristic in an image frame obtained from the camera, we utilized the Face Detection Library – libfacedetection [1]. We compiled it into a dynamic library and called it in our project. The project itself provides a pre-trained ONNX model, and we directly exploited that. The model itself scores an `AP_easy=0.856, AP_medium=0.842, AP_hard=0.727` on `WIDER FACE` face detection benchmark.

The network structure is in Figure.1. Zoom in for detail.

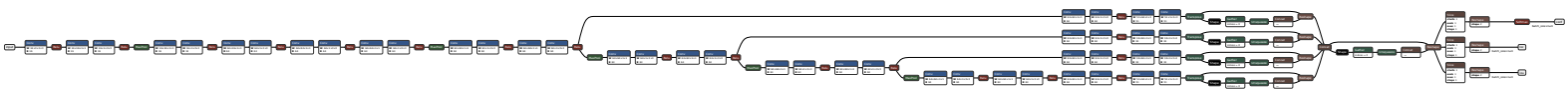


Figure 1: Network Structure

From our actual test, this structure can perform face detection fairly well. In a low-resolution ($300px \times 160px$) image (Figure.2), the network successfully identified three faces among.

We plan to recreate that training process on our own, but training is a GPU-intensive task. Also, it would require a large scale of face labelled dataset. For now, it is of our low priority.

## 2.2 Body Detection

OpenCV features an implementation for a very fast human detection method, called HOG (Histograms of Oriented Gradients) [2].

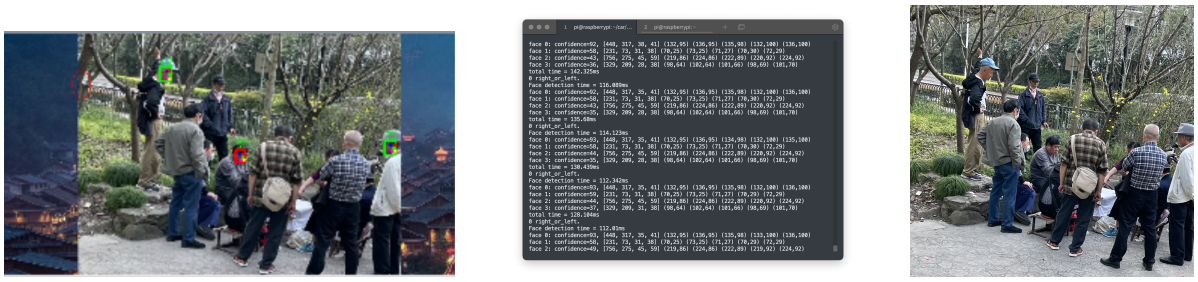


Figure 2: (a) Detected faces in small image, (b) Command line output, (c) Original image.

The basic idea of the method is the following:

- The picture is scanned with a detection window of varying size.
- For each position and size of the detection window, the window is subdivided in cells. The cells are in practice relatively small: they typically contain only a small part of the person to be detected, maybe the side of an arm, or the top of the head.
- In each cell, a gradient is computed for each pixel, and the gradients are used to fill an histogram: the value is the angle of the gradient, and the weight is the magnitude of the gradient.
- The histograms of all cells are put together and fed to a machine learning discriminator to decide whether the cells of the current detection window correspond to a person or not.

The machine learning discriminator used here is the SVM.

### 2.2.1 Tuning the hyperparameters

The detection of body can be also fairly computational-expensive. Hence, the trade-off between accuracy and speed for using the detector is pretty drastic. Still, we will try our best to get almost-real time predictions on videos with pretty high accuracy.

There are three impactful parameters here: `winStride`, `padding`, and `scale`. `winStride` defines a step size for the detector window to move in the horizontal and vertical direction. The smaller the step size, the more important and fine-grained details we will be able to capture. The `scale` hyperparameter defines the factor by which the image is resized at each layer of an image pyramid. The `padding` parameter is used to pad the sliding window detector in the horizontal and vertical direction. The `padding` values indicate the number of pixels to pad the sliding window.

Here, we choose `winStride` = (8,8), `padding` = (12,12), and `scale` = 1.05.

## 2.3 Integration

Now that we have both face detection and body detection, a simple notion is that we detect face first, then detect body if there's no presence of face, since human body is easier to detect when the person is more distant with the camera on the robot car.

### 2.3.1 Image resizing

The image input on my dummy camera has a resolution of $1280px \times 720px$. Detecting faces on a `Raspberry Pi 4B, Broadcom BCM2835, Cortex-A72 (ARMv8)64-bit SoC @ 1.5GHz` could take up to 3000 ms, which is totally unacceptable. An easy solution is to downscale the image so as to speed up the whole process at the cost of detecting accuracy.

Therefore, we cannot detect faces that are too small after downsampling the input image. That's when body detection comes into play. We've carefully adjusted those resampling parameters so that when faces cannot detect anymore, we detect bodies. The final goal of this project is to make the robot car follow people. By combining the two tools, we can fulfill this goal.

### 2.3.2 Control Logic

Sure, we can easily locate human faces/bodies in an image frame now. The control logic should be pretty straightforward. When the face/body is too small, we move forward to the target. When the face/body is on the left/right side, we turn our car correspondingly.

However, there's still something more worth being dived into. With a 100 to 400 ms lag, the human could already leave the camera scope while the car is still searching for it. So we added a variable called `right_or_left` to emulate the momentum of the turning. The variable `is_forwarding` performs a similar function in controlling whether the car should move forward or not.

For instance, when the human is at the leftmost of the frame and finally gets out of the frame, the robot car tends to keep turning left for a little while, just to ensure that we don't miss the human in that case.

As mentioned above, the detecting time varies between whether an acceptable face has been found. But we want the car's turning to be independent of the detecting part. That's why we've launched a thread for that. By routinely checking the `is_forwarding` and `right_or_left` variables, we can control our robot car in a more stable way.

### 2.3.3 Video Stream Buffer

Another problem we've encountered is the image frame reading method. By directly reading frames from the VideoCapture object, we essentially read frames from a designated buffer. The buffer is a queue-like data structure. The camera continuously writes frame to the buffer, and our program takes out one buffer every time it finishes processing the previous one. Since it typically takes hundreds of milliseconds for us to finish processing one frame, the frame that our program read could be from several seconds ago!

To address this problem, we again resort to multithreading. Instead of using a queue-like buffer, we create our own stack-like buffer. In our project, there are two separate threads: `frame_read` and `frame_write`. `frame_write` is responsible for writing whatever we get from the camera to the custom buffer, and purging in interval when there are too many frames. `frame_read` is basically the original program, except now it reads frames from our custom buffer. We can obtain the latest image frame from the camera by utilizing such parallelization, reducing the lag between human movement and robot car action.

# 3 Detection on Pi – Demonstration & Implementation

## 3.1 GPIO interface

GPIO is short for General Purpose Input and Output Port, which allows software control of outputs and inputs. the GPIO pins of the Raspberry Pi are connected to external devices for communication, control, and data acquisition. We use a character device library (libgpiod) to interact with the Linux GPIO and to control the behavior of the four motors. The Raspberry Pi GPIO pins used are pins 18, 23, 24 and 26, which are connected to motor IN1, motor IN2, motor IN3 and motor IN4 on the motherboard, respectively. The relationship between the pin levels and the car motion state is obtained by testing as Table 3.1

| Pin | 18 | 23 | 24 | 25 | Movement |
|---|---|---|---|---|---|
| State | HIGH | LOW | HIGH | LOW | Forward |
| | LOW | HIGH | LOW | HIGH | Backwards |
| | LOW | HIGH | HIGH | LOW | Turn Left |
| | HIGH | LOW | LOW | HIGH | Turn Right |

Table 1: Truth Table of Car Movement

## 3.2 Human selecting logic

When there are multiple targets in the car's field of view, the car will select a person from them to track. The selection logic is: when there are multiple faces in the picture, the target with the highest confidence is selected. If two targets both reaches the confidence threshold, the larger target is selected. When no face is detected in the field of view, the car selects the target with the highest body confidence for tracking.

## 3.3 Action Logic

In order to realize the dynamic tracking of people by the car, it is necessary to turn left and right, forward and backward and stop according to the position of people relative to the car. We set the following rules: when the face falls in the left third of the field of view of the car, the car turns left; when the face is in the right third of the field of view, the car turns right; when the face is in the middle third of the field of view of the car, the car keeps the original direction. The time for the robot car to make an angle-adjusted turn is 100 milliseconds each time, and the body turns 22.5° after the test. In addition, the distance between the car and people can be reflected by the proportion of the face to the car's field of view, we set when the face box area is less than 1/36 of the car's field of view, that people and cars are far away, at which time the car advances a distance.

# 4 Detection on Raspberry Pi – Problems

## 4.1 Image blur – Would it affect detecting quality?

Since we used two separate threads to perform fetching images from the camera and turning the car to left/right, it's not hard to predict that the image will blur completely when taken during the turn. Therefore, we introduced a Semaphore-like structure to prevent such a scenario. The idea is taken from Operating Systems. Specifically, when the car turns, we prevent the camera from capturing frames by imposing a lock on it.

But there arises a new problem. The core of that lies in the camera characteristic: the image will blur whenever the car or the human is moving horizontally. There's no elegant solution to that. Hence, the car cannot track a moving object. It can only track a relatively static one.

## 4.2 Detection onboard – Acceptable lag?

At $300px \times 160px$, it takes 100 ms to perform a face detection, and another 300ms at $700px \times 375px$ to perform a body detection when there's no (acceptable) face in the frame. This could result in a perceivable lag when the car is trying to find and locate a human.

Though we did not thoroughly test it, it could be a throttle neck for this project.

Our current solution is to further turn down the image solution to speed up the detection process. But this is at the cost of detection accuracy after all.

You may see that there's a more distinct lag in the real-time demo, that's because of the transmission of the processed image frame. Once we disable that, detection could be a lot faster.

Android implementation is on the go. We believe that the detection process could be 3 to 4 times faster.

## 4.3 Camera Z-axis degree of freedom

In our field test, we figured out that turning the Z-axis of the camera is essential when the human is too close/far to the camera.

With lack of motors, we cannot alter the Z-axis of the camera. However, by carefully adjusting the direction of camera, we can still capture human faces from robot car successfully.

# 5 Detection on Android Phone – Core System Design

To address the above-mentioned problems, we've decided to shift the face detection task to an Android phone with a more powerful CPU. In practice, the response is instant, and the captured image didn't blur even when the robot car is turning. In order to implement the change, we need to work out a solution to transmit the face detection result to the raspberry pi, which controls the robot car's direction.

In addition, we've implemented a Chinese speech recognition engine. It can recognize the user's voice input and act correspondingly, as described in the Application Scenario section. We've also implemented two functions: Obstacle Avoidance and Automatic Route Following. This can help the robot car to better serve customers in a complicated environment.

## 5.1 Simulink on Matlab

Simulink is a visual simulation tool in MATLAB developed by Mathworks. Simulink is a modular graph environment for multi-domain simulation and model-based design. It supports system design, simulation, automatic code generation, and continuous testing and validation of embedded systems. Simulink provides a graphical editor, a library of customizable modules, and solvers for dynamic system modeling and simulation.

Simulink is integrated with MATLAB, and MATLAB algorithm can be integrated into the model in Simulink, and the simulation results can be exported to MATLAB for further analysis. Simulink applications include automotive, aviation, industrial automation, large-scale modeling, complex logic, physical logic, signal processing and other aspects. A simulink interface is shown in Figure 3.

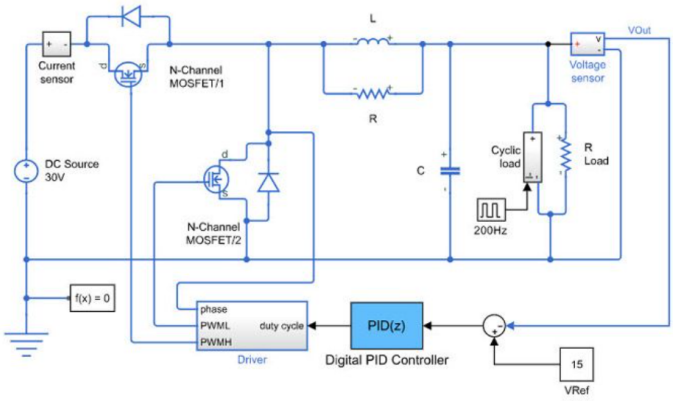



Figure 3: Examples of simulink interfaces

Viola-jones algorithm is a very classic face detection algorithm, which was proposed in 2001 and is widely used for its high efficiency and fast detection.

The general logic of the algorithm is to use Haar features to describe the common attributes of the face, and then establish a feature called Integral Image, which can easily obtain a variety of different rectangular features. Then the Adaboost algorithm is used to train and a strong classifier is obtained.

## 5.2 Face Detection on Android Phone using Simulink

### 5.2.1 Haar features

The face captured from the front will have some features, for example, in grayscale mode, different parts of the face have different degrees of light and shade. For example, the eye area

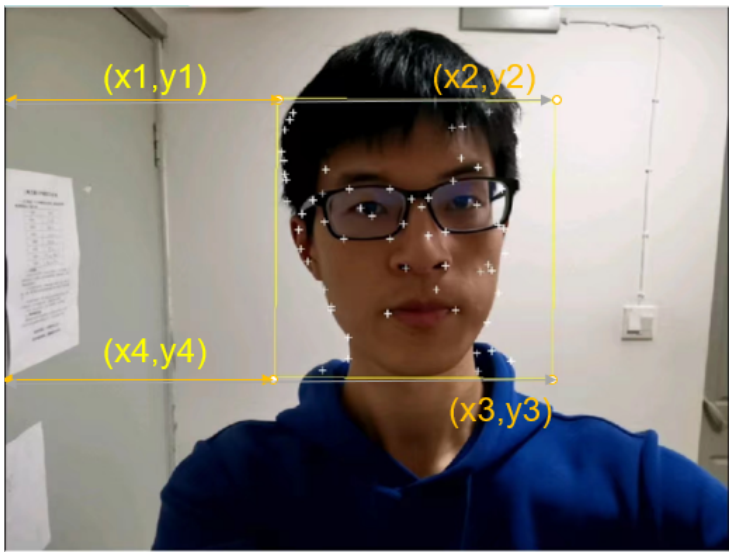



Figure 4: Face Detection using Android phone

will be darker than the cheek area, the lip area will be darker than the surrounding area, and so on, as shown in Figure 5.

Figure 5: Different areas of the face have different levels of light and shade

Based on face light and shade features and their distribution, the following four rectangular features can be described, as shown in Figure 6.

So, how does Haar feature apply to frontal images?

As shown in Figure 7, face specific location of the distribution of light and shade can be combined with certain rectangle features, such as the right thin line features of the rectangular characteristic corresponding facial parts for eyes and nose Due to the dark eyes, and the bridge of the nose is bright, so we can place a piece of bright color with two pieces of dark rectangle rectangle to briefly describe the positive face So, how do we determine whether an area is a light or a dark area? We need to sum the pixels of the white region and the pixels of the black region, and then sum the difference and that can be done by integrating the image.

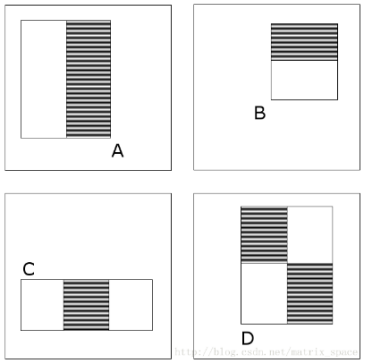



Figure 6: The rectangle features

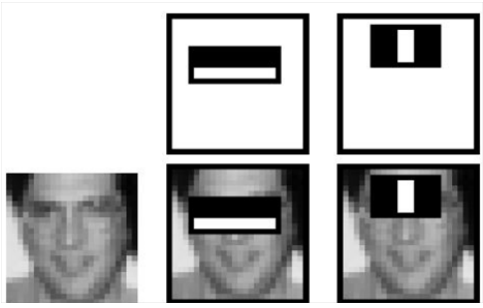


Figure 7: The application of different rectangular features on human face

### 5.2.2 Integral Image

For any point of an image, its integral image value is equal to the sum of all pixels located at the upper left corner of the point to establish a plane cartesian coordinate system with the upper left corner of the image as the origin, the positive direction of x axis horizontally to the right, and the positive direction of Y axis vertically downward. Then, the expression is as follows:

$$I(x,y) = \sum_{x \le x'} \sum_{y \le y'} f(x',y')$$

And the image integral satisfies the following relation:

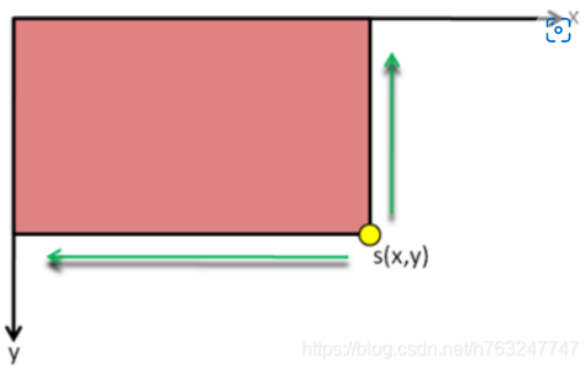



Figure 8: Integral image

$$I(x,y) = f(x,y) + I(x-1,y) + I(x,y-1) - I(x-1,y-1)$$

Where $I(x,y)$ refer to the integral value of the image corresponding to the point, and $f(x,y)$ represent the pixel value of the point itself, as shown in Figure 8.

### 5.2.3 Obtaining Image Features

For the rectangle ABCD shown below, its pixel sum can be represented by an image integral:

$$S_{abcd} = I(D) - I(B) - I(C) + I(A)$$

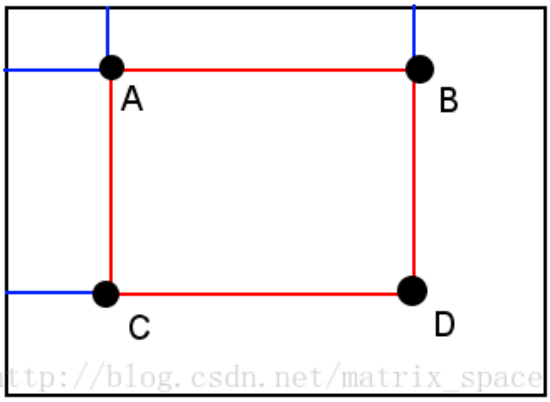



Figure 9: Calculating the pixel value of the matrix

Figure 9 means that the sum of pixels in any rectangular region can be represented by the four points above the image integral so that we can use the image integral at each point to get the pixels that identify the adjacent rectangles and use that to determine what the rectangular features are Therefore, combining the four kinds of rectangular features abstracted from Haar features, we can use the image integral value corresponding to the point to get the rectangular features contained in an image, and then compare the feature rectangular distribution of the face to get the positioning of the face.

### 5.2.4 Adaboost Classifier

It is one of the machine learning classification algorithms that can enhance the weak learner whose prediction accuracy is only slightly higher than random guess to the strong learner with high prediction accuracy through a kind of cascade relationship.

$$h(x) = sign(\sum_{j=1}^{M} \alpha_i h_j(x))$$

$h(x)$ is a strong classifier, and $h_j(x)$ is a weak classifier, which is a simple threshold function:

$$h_j(x) = \begin{cases} -s_j & f_i < 0 \\ \\ s_j, & otherwise \end{cases} \tag{1}$$

$\theta_j$ is the threshold, and $s_j$ is in the range from $-1$ to 1. $\alpha_j$ is the coefficient.

The realization logic in VJ algorithm is to recognize some feature matrix, so as a reference for a face location; In order to reduce the amount of computation, the recognition range will gradually become refined from large to small.

In MATLAB, the trained classifier can be directly used.

## 5.3 Face Tracking on Android Phone using Simulink

KLT algorithm is an efficient tracking algorithm based on image feature points, can be predicted in a set of successive images of target trajectory Effect is very good for face detection, face detection effect is superior, the same part of the VJ algorithm It has three assumptions: brightness constant (front and rear frame of content similarity judgment is not affected by brightness); Time continuous or small motion (to ensure that KLT can find points); The space is consistent, and adjacent points have similar motion and remain adjacent (all points in the same window have equal offsets).

The algorithm determines the position of the corresponding point on the next frame by looking for the region with the smallest gap between the surrounding area of a point on the previous frame and the next frame. This sentence can be equivalent to minimizing the following formula:

$$\varepsilon(d) = \varepsilon(d_x, d_y) = \sum_{x=u_x-w_x}^{u_x+w_x} \sum_{y=u_y-w_y}^{u_y+w_y} (I(x,y) - J(x+d_x, y+d_y))^2$$

KLT algorithm is given, in short, the image on the traverse to find out the feature points, namely angular point, commonly used Harris corner points, then with the given target template, namely after the tracking of feature points and alignment calculation of an iterative calculation of incremental translation, to implement tracking, can also be considered, after image registration affine transformation parameters calculated goal iteration tracking.

## 5.4 Transmitting Face Location to Raspberry Pi

### 5.4.1 UDP protocol

UDP uses a simple connectionless communication model with a minimum of protocol mechanisms. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network; there is no guarantee of delivery, ordering, or duplicate protection. If error-correction facilities

are needed at the network interface level, an application may instead use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose. The structure of UDP protocol is shown in Figure 10.

UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing in the protocol stack. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for packets delayed due to retransmission, which may not be an option in a real-time system.

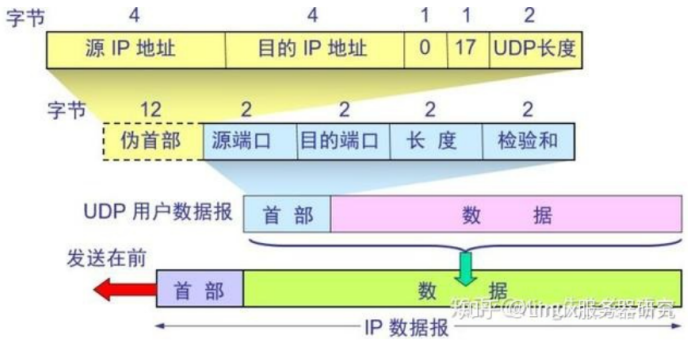



Figure 10: UDP Protocol

In the UDP protocol layer model, the UDP application accesses the UDP layer and uses the IP layer to transmit data packets. The data part of the IP packet specifies the source and destination host addresses for the UDP packet's IP layer header, and the UDP layer header specifies the source and destination ports on the host UDP transport segments consist of 8-byte headers and payload fields.

### 5.4.2 Demonstration & Implementation

In the `DrawAnnotation` function, as shown in Figure 11, we added a `UDP Send` block to send the face recognition result to Raspberry Pi. Since we plan to implement Chinese speech recognition, we are forced to use a C program to control the robot car's behaviour. Therefore, it is crucial for us to decrypt what's been sent from the Android phone using UDP protocol.

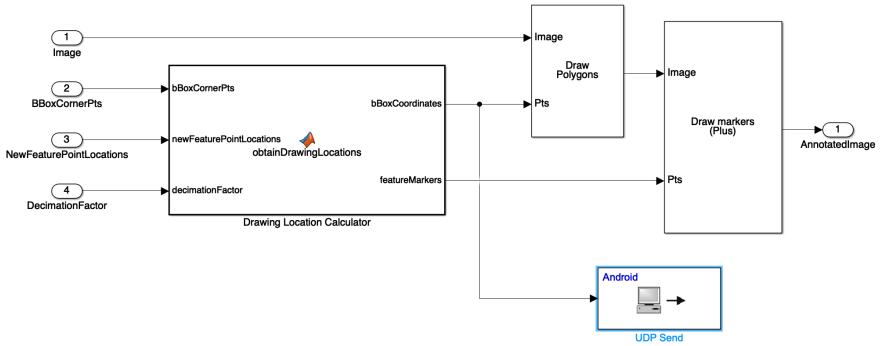



Figure 11: UDP send component in the DrawAnnotation function

We set the "Dest IP address" parameter to "255.255.255.255", so that the data is broadcasted to every user in the Local Area Network. By inspecting the data packets in Wireshark, as shown in Fig 12, we can spectate the pattern of the four location points of detected face, which are `(x1,y1)`, `(x2,y2)`, `(x3,y3)` and `(x4,y4)`. Therefore, by following a specific pattern and decode the location data from eight 16 bits hexadecimal data, we can extract where the face is and hence derive the operation of the robot car.

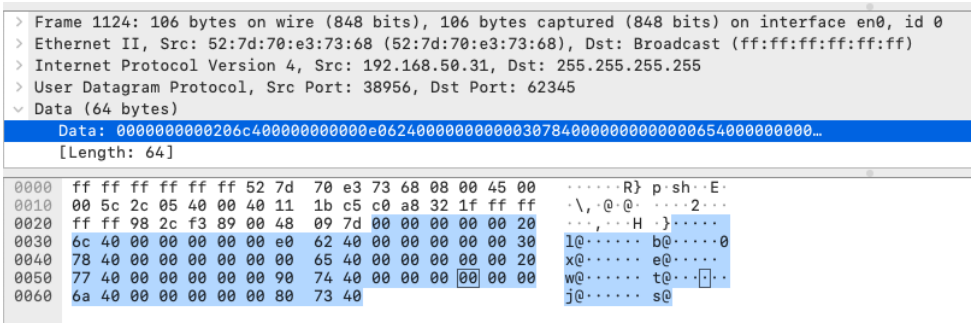



Figure 12: UDP data packet analysis with Wireshark

However, we found out that the mapping between the hexadecimal value and the actual decimal value are not linear. For instance, we were hoping that if hex value `0x0000000000003840` represents the decimal value 24, and hex value `0x0000000000003940` represents the decimal value 25, then we can deduce that `0x0000000000003a40` represents the decimal value 26. While this pattern does exist in the actual mapping, it does not hold often. The hex value `0000000000005040` and `0000000000405040` represents decimal value 64 and 65 respectively, which does not conform to the above rule. In the end, we've decided to write such mapping brutally into a text file, and let the C program read this file and build a `hex->dec` map every time. This file contains 1200 8-**bytes** hexadecimal value, which each corresponds to the decimal value from 0 to 1199, as shown in Figure 13(a). Therefore, our C program can correctly build such conversion map and decode the face location correctly from the input of Android phone automatically, as shown in Figure 13(b).

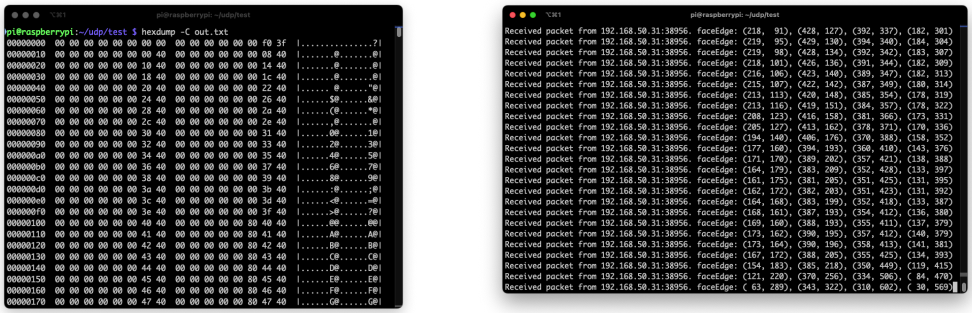



Figure 13: (a) Saved mapping from Hex to Dec value (b) Decoded face detection result

## 5.5 Chinese speech recognition

We implemented a Chinese speech recognition engine, so as to provide certain services to customers when their faces were detected. The server can run locally and do not require an

Internet connection. However, since the Raspberry Pi board we are using is based on ARMv8 architecture, and the model is trained on a x86 machine, we decided to deploy the server on a jCloud virtual machine using Docker for simplicity. Later, we can move the server program to local Pi board, in order to shake off the dependence of a network connection.

The steps of speech recognition involves Feature extraction, passing through an neural network, CTC collapsing, and building a statistical language model to convert Pinyin to Chinese words.

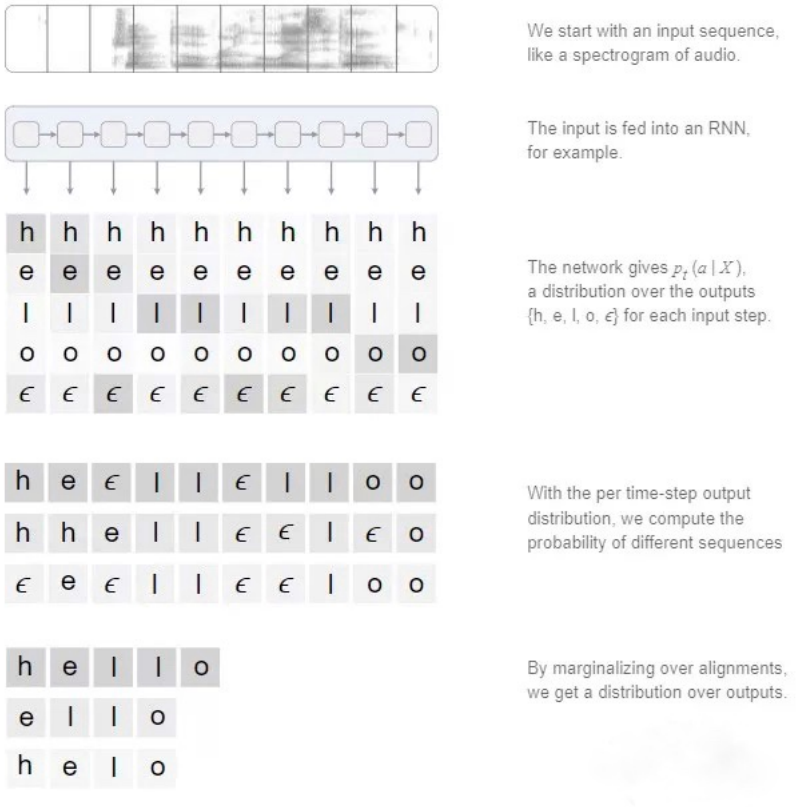



Figure 14: The speech recognition workflow

Figure 14 roughly shows the workflow of speech recognition. For Chinese speech, we need an extra step. Since the model only outputs syllables results, we need the statistical language model to convert Pinyin to actual Chinese characters. But this model does not work well every time. For our application scenario, we only need to respond to a limited set of voice commands. Therefore, determining commands directly from Pinyin could be a lot more accurate.

Since we do not possess such large Chinese speech dataset, and the GPU of our laptop is not suitable for training task, we decided to use a trained model to deduce result. The server is deployed on a cloud host, and the client (Raspberry Pi) can call the API by passing a base64 encoded audio file. The server decodes that file, and return the result from the neural network. We've written a Python script to do such job. When called from command line, the script records voice input from the built-in microphone, stops recording when the speaker has stopped talking, sends the recording file to the server through its API, and parse the final Pinyin results with a classifier. The demo result is shown in Figure 16.

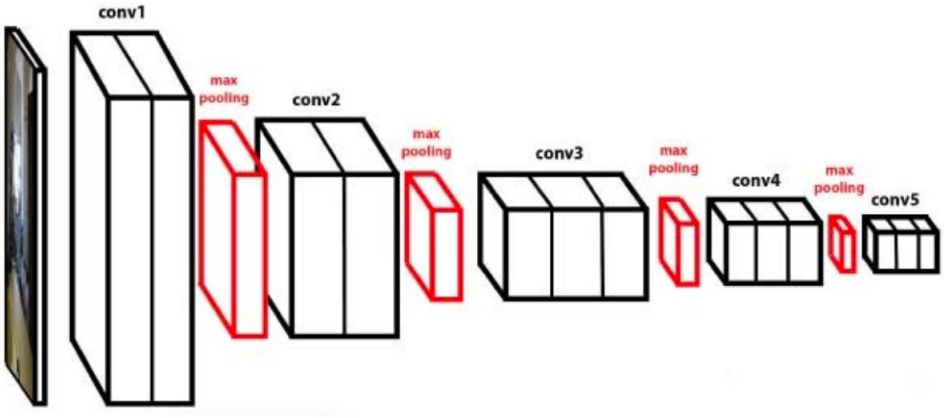



Figure 15: The RNN network structure

```
please speak a word into the microphone
done - result written to demo.wav
{"result": "\u628a\u83dc\u7aef\u4e0a\u6765",
age": "all level"}
把菜端上来
{"result": ["ba3", "cai4", "duan1", "shang4",
tatus_message": "speech level"}
['ba3', 'cai4', 'duan1', 'shang4', 'lai2']
{"result": "\u628a\u83dc\u7aef\u4e0a\u6765",
age": "language level"}
把菜端上来
```

Figure 16: Chinese Speech Recognition Result

### 5.5.1 Using Android phone as a microphone

One critical problem is that the Raspberry Pi board does not equip a microphone, nor do we have a USB external microphone. The only workaround is to leverage the microphone on the Android phone, but that requires us to build a stable and real-time connection between the phone and the Pi, and we also need to simulate a virtual microphone on the Raspberry Pi. Luckily, there's a low latency voice chat application called Mumble that supports both Android clients and Linux clients. Therefore, by setting up such meeting room within the same LAN, the virtual microphone can take the sound from the meeting room as input, and perform Chinese speech recognition directly from the Android phone.

The latency is very low, both theoretically and in practice. It sounds almost like local loopback, and it has a theoretically 7ms WiFi delay + 2x 10ms codec delay = 27 ms delay.

The demonstration is shown in Figure 17. Figure 17(a) shows the Voice over Mumble server running on the Raspberry Pi board to receive voice signals from the client and simulate a virtual system microphone, and Figure 17(b) shows the Mumble Android client which transmits the voice signal from its microphone to Raspberry Pi server.

## 5.6 Integration - Thread Synchronization

To implement the two functions into one program, we've started three threads correspondingly: One is to read face recognition data from UDP socket, one is to control the car's direction

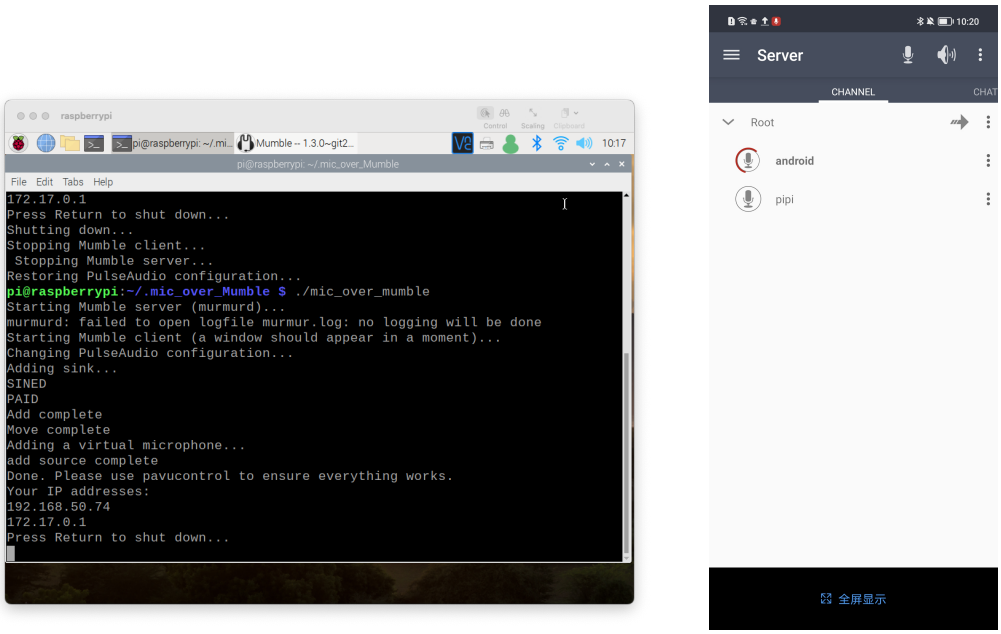



Figure 17: (a) Voice over Mumble server (b) Mumble client on the Android phone

according to the face location, and the last one is to listen for Chinese speech and control the car's direction accordingly. It is easy to notice that there is a control hazard here: What if the face is on the right and the voice tells us to turn left? Naturally, the priority of voice control should be higher, and the car should always follow the voice commands when it is available.

The solution is to introduce **MUTEX**, a mechanism that enforces limits on access to a resource when there are many threads of execution. We want the controlling process of the car to be atomic operations, and other threads should wait for this resource when it is hold by another thread.

```
1: mutex mtx;
2: mtx.lock();
3: // spin motors accordingly
4: mtx.unlock();
```

# 6 Additional Features

## 6.1 Obstacle Avoidance

In order to realize the intelligent trolley to achieve more combined with the actual face tracking, we added obstacle avoidance function on the trolley. The obstacle avoidance function is realized by ultrasonic module and infrared obstacle avoidance module.

### 6.1.1 Ultrasonic obstacle avoidance

Ultrasonic module a control port to send a high level of more than 10US, in the receiving port waiting for a high level output. Once there is an output you can turn on the timer timing, and when the receiving port goes low you can read the timer value, at this time is the time of this distance measurement, before you can calculate the distance. The working principle of the module is as follows.

- IO triggered ranging, giving a high level signal of at least 10us.
- automatic transmission of eight 40kHz square waves by the module, which automatically detects whether a signal is returned.
- When a signal is returned, a high level is output via IO, and the duration of the high level is the time from emission to return of the ultrasonic wave. Test distance = (high level time * speed of sound (340m/s))/2.

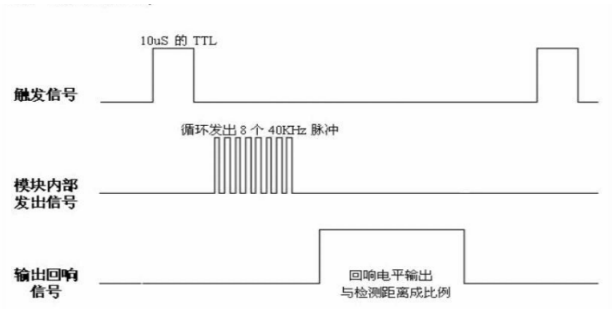



Figure 18: Ultrasonic Timing Chart

However, it should be noted that ultrasonic obstacle avoidance requires a large obstacle area and good flatness, otherwise it will cause a large error on distance measurement. We set up the ultrasonic obstacle avoidance logic as shown in the Figure 19.

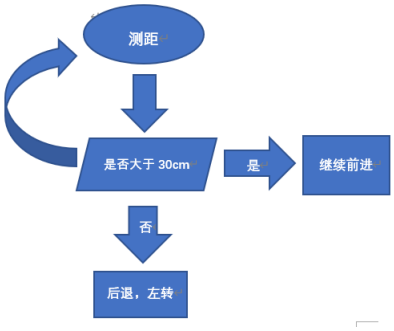



Figure 19: Ultrasonic obstacle avoidance logic

### 6.1.2 Infrared obstacle avoidance

The infrared emitting diode of the obstacle avoidance module continuously emits infrared light, which is accepted by the infrared receiver when it is reflected by an object and outputs

an analog value. The output analog value is related to the distance of the object and the color of the object. A comparator built into the module compares the output analog value with the set threshold value, thus outputting a high and low level. The threshold value can be adjusted by a varistor, and in the process, the threshold value of the obstacle avoidance distance is also adjusted. We install an infrared sensor on the left and right of the trolley, and the output level of the two infrared sensors can judge the position of the obstacle relative to the trolley, so as to control the movement of the trolley. The specific working logic is as follows.

- When the output of both left and right sensors are low, it means that there is an obstacle in front of the car, and the car backs up a short distance and then turns left.
- When the output of the left sensor is low and the output of the right sensor is high, it means there is an obstacle on the left side of the trolley and the trolley turns right.
- When the output of the right sensor is low and the output of the left sensor is high, it means there is an obstacle on the right side of the trolley and the trolley turns left.
- When the output of both left and right sensors are high, it means there is no obstacle in front of the trolley, and the trolley can continue to go straight.

In order to make the obstacle avoidance distance more accurate, we use the ultrasonic and infrared hybrid obstacle avoidance program. On the basis of infrared obstacle avoidance, the distance measured by ultrasound is also used as one of the conditions for determining whether there is an obstacle in front. In the test process, we found that the hybrid obstacle avoidance scheme is better than the single obstacle avoidance scheme.

### 6.2 Following the trail

In the application scenario we set, the cart is required to follow a specific trajectory. To achieve this function, we add a trajectory module to the trolley. There are three general options for tracing.

- Infrared pair of tube following method: using the different absorption effect of black and white on infrared.
- Camera-tracking method: using cameras to read road information, divided into analog and digital.
- Laser tube tracing method: similar to the principle of infrared tracing method, but the detection distance is longer.

After comparison, we choose the infrared tracing scheme. As mentioned in the previous section, the analog value output from the infrared sensor is related to the distance of the object and the color of the object. We install the sensor at the bottom of the cart and make the sensor output low when the ground is black by adjusting the threshold value. Then by setting the travel logic similar to IR obstacle avoidance, we can realize the black line tracing function.

## 7 Demos & Code repository

Code and demo video are available at https://github.com/VoidXia/Face-Detection-Car.

# 8 Result

To sum up, we successfully completed the Face Detection and Human Tracking Robot project task of Mathwork by using simulink module in MATLAB software, and designed a practical application scenario for the car on the basis of the project. That is, the restaurant service scene, so that the car has more practical effect. Therefore, we added the speech recognition function for the car, and tried the car's obstacle avoidance and tracking function, so that the smart car can be more adaptable to the actual scene application.

# 9 Acknowledgements

Thank you very much to the teachers of Outstanding Engineer Class and Mrs. Xu for providing us with such a valuable learning experience!

Thank you very much to Mrs. Yang and Mrs. Xu for your help all the time! Although we encountered many difficulties in the process of learning and communication due to the epidemic, thanks to the positive communication between the two teachers and all the team members, we were able to successfully finish the project.

Thanks to Mathwork for providing us with a wealth of learning materials and teaching resources that have enabled us to move from being unfamiliar with Simulink to becoming proficient with it.

Many thanks to the project's creator, Mr. Roberto G. Valenti, for answering our questions online!

# References

[1] Y. Feng, S. Yu, H. Peng, Y.-R. Li, and J. Zhang, "Detect faces efficiently: A survey and evaluations," *IEEE Transactions on Biometrics, Behavior, and Identity Science*, vol. 4, pp. 1–18, Jan 2022.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, June 2005.