UNIVERSITY *of* LIMERICK

OLLSCOIL LUIMNIGH

Mechanical arm synchronization

(An imitation of human arm action)

University of Limerick

Ollscoil Luimnigh

Student Name: Jiasen Tian

Student number:16060245

Supervisor: Annette.McElligott

17th April 2018

Final Year Project

Acknowledgements:

To my Parents for their support over the years.

To my supervisor, Annette McElligott, for her advice and patience throughout the year.

Table of Contents

Chapter 1: Introduction and Objectives

1.0 Introduction

Nowadays, using of robotics has exponential growth in various kinds fields such as embedded engineering, mechanical engineering, medical engineering and so on. More and more fields are working together to create more intelligence and flexible robots. With the rise of robotic technology, there is a problem happened - How can people controls robots easily? Because this situation, the academic research of Robotic smart controlling become more and more popular in recent years.

There are lot of robotic smart smart controlling system to mimic human arm action. Most of them use Leap Motion Controller or sensor glove to track hand posture, few people try to use camera to control the mechanical arm. This project will try to create a smart robotic arm controlling system using two USB cameras.

This chapter will introduce:

- Mechanics of robotics,
- Challenges in imitating movement,
- Benefits of researching on mechanical arm mimic human arm,
- The reason why would a person want to do this?
- Project objective.

1.1 Mechanics of robotics

1.1.0 Introduction

This part will introduce some essential knowledge about mechanics of robotics which we will use in this project.

1.1.1 Robot Joints

Robot Joints are most important part of mechanical arm, the following picture shows the basic joints found in typical robots.

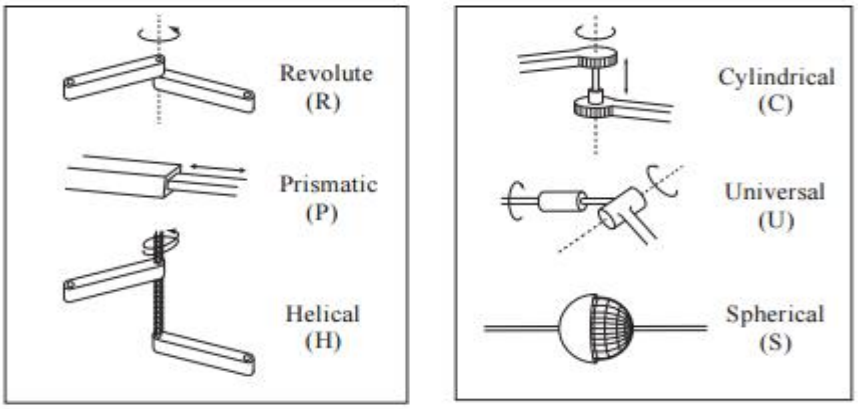


Figure 1.1.1.1: Typical robots joints
Source: (Park and Lynch 2016)

Every joint connects exactly two links. The situation never happen is joint that simultaneously connect more than three links.
The revolute joint (R), another name is hinge joint which allows for rotational motion around the joint axis. This is the most important and popular joint type and all mechanical arm joints are revolute joints. The prismatic joint (P), another name is linear joint which allows for translational motion along the direction of the joint axis. The screw joint (H), another is helical joint which allows simultaneous rotation and translation about a screw axis. (Park and Lynch 2016)

Revolute, prismatic, and screw joints all have one degree of freedom. Joints can also have multiple degrees of freedom. The cylindrical joint (C) is a two-dof joint that allows for independent translations and rotations about a single fixed joint axis. The universal joint (U) is another two-dof joint that consists of a pair of revolute joints arranged so that their joint axes are orthogonal. The spherical joint (S), also called a ball-and-socket joint, has three degrees of freedom and functions much like our shoulder joint.(Park and Lynch 2016)

A joint can be viewed as providing freedoms to allow one rigid body to move relative to another. It can also be viewed as providing constraints on the possible motions of the two rigid bodies it connects. “Generalizing, the number of degrees of freedom of a rigid body minus the number of constraints provided by a joint must equal the number of freedoms provided by the joint.” (Park and Lynch 2016)

The freedoms and constraints provided by the various joint types are summarized in Table.

Joint type	dof f	Constraints c between two planar rigid bodies	Constraints c between two spatial rigid bodies
Revolute (R)	1	2	5
Prismatic (P)	1	2	5
Screw (H)	1	N/A	5
Cylindrical (C)	2	N/A	4
Universal (U)	2	N/A	4
Spherical (S)	3	N/A	3

Figure 1.1.1.2: Joint type table
Source: (Park and Lynch 2016)

1.1.2 Robot Joint servo

In this project, we choose LD-27MG servos as mechanical arm joints, as figure1.1.2.1 shown.

Figure 1.1.2.1: LD-27MG digital servo
Source: (Amazon picture 2018)

This servo can be controlled by PWM signal and has 20kg large torque and 270 degree rotation. Those property is satisfied for project mechanical arm. In this project, we will use 6 LD-27MG digital servo as its joints servos.

Controlling servo is also essential part of project. Servo is driven by PWM (pulse-width modulation) signal. The parameters for the pulses are the minimal pulse width, the maximal pulse width, and the repetition rate. Given the rotation constraints of the servo, neutral is defined to be the center of rotation. Different servos will have different constraints on their rotation, but the neutral position is always around 1.5 milliseconds (ms) pulse width.(Wikipedia 2018) For LD-27MG servo, its pulse width is 0 ~ 2.5ms and thee neutral position is 1.5ms.
More details of PWM signal computational formula will be introduced in Chapter 4.

1.2 Challenges in imitation movement

1.2.1 Challenges in Building Imitative Robots

This part will introduce some difficulties points in building imitative robots. “From a practical perspective, building a robot is an enormous investment of time, engineering, money, and effort. Maintaining these systems can also be a frustrating and time-consuming process.” (Breazeal and Scassellati 2015)

To solute very specific problems of imitation, researchers should make some simplifying assumptions and trade-offs. As Breazeal and Scassellati said simplifications in the hardware design, the computational architecture, the perceptual systems, the behavioral repertoire, and cognitive abilities allow a research team to address the more complex issues without implementing complete solutions to other problems. Each researcher must be very careful to describe the assumptions that are made and the potential implications of these assumptions on the generality of their results. While these simplifications at one level are unavoidable, it is important to keep the big picture in mind. (Breazeal and Scassellati 2015)

1.2.2 Challenges in technological implementation in this project

There are three key difficulties in this project:

- Catch human arm in camera and get arm posture information
- Drive all the joints of mechanical arm simultaneously
- Communication between PC application and mechanical arm controlling circuit board

Catch human arm in camera and get arm posture information

The most difficult challenge in this project is using cameras to catch human arm posture and analyze source from camera video. The first hard point is how to distinguish human arm from complicated background. As picture 1.2.2.1 shown, this is a example of subtracting background from picture which implemented by OpenCV library function: BackgroundSubtractorMOG2. The test result was not meet expected level. (As picture 1.2.2.2 shown)

Figure 1.2.2.1: Video source

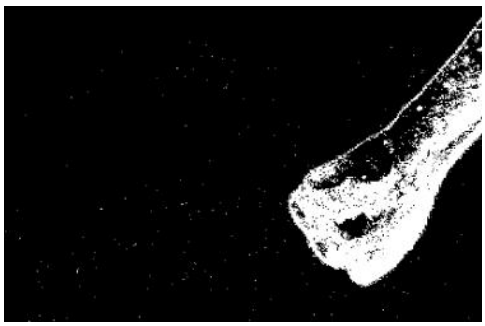


Figure 1.2.2.2: Video source after processing

It is obvious that there are a lot of white noise points in background which could greatly influence on processing and analyzing step.

Implementation Solution: Using skin-detect and morphological operation to remove noise and make the boundary of hand much clearer. It is also essential to use erode and dilate operations to process pictures' matrix.

Drive all the joints of mechanical arm simultaneously

As we all know, in the stm32 circuit board, CPU only can process one task at a time(if not use USOS II), which means the circuit board just can drive one joint servo motor at a time and other joint servo motors have to pause until current task finish. If joints have to change its angle one by one rather than change all the joints at the same time, action of mechanical arm could slow and slack.

Solution: Using for loop in driving function to change all joint servo motor's PWM signal simultaneously, which means all the joint servo motors could rotate its axis simultaneously. For example, there is an action of mechanical arm. Change joint 1 from 90 degree to 95 degree; change joint 2 from 90 degree to 95 degree; change joint 3 from 90 degree to 95 degree; change joint 4 from 100 degree to 105 degree; change

joint 5 from 100 degree to 95 degree; change joint 6 from 90 degree to 95 degree. In servo driving function, there is a for loop which will control change joints PWM signal five times, all joint's angle will increase 1 degree in each operation time. In this way, all the joints can be controlled at same time.

Communication between PC application and mechanical arm controlling circuit board.
If PC application can't send the valid action data in time or successful, action of mechanical arm could be very strange, lack smoothness and uniformity. This action data should easy to understand for both PC and circuit board.

Solution: Creating a new communication protocol between PC and circuit board. The PC will send human arm information data in every second. The data will contain the change of each joint angle(old angle and new angle) in one second. For example, in this project, PC will send a 36 digits number to circuit board every second, which contains every joints' old action angles and new angles. When circuit board get the action code, it will decode the data and create new tasks to control its PWM signals which could change joint servos' rotation angle.

1.3 Benefits of researching on mechanical arm mimic human arm

If the technology of mechanical arm is enough mature, human will get huge liberation from heavy, easy-repeat or extremely exquisite work. This section will give three mechanical arm application examples in different fields.

1. Arms for Automotive Manufacturing

The production of cars would be extremely difficult if factory don't the mechanical arm. The first mechanical arm was used by "General Motors" factory which is first time to solved this situation. Using this mechanical arm, also known as an industrial robot, engineers were able to achieve difficult welding tasks. In addition, the removal of die-castings was another important step in improving the abilities of a mechanical arm. With such technology, engineers were able to easily remove unneeded metal underneath mold cavities. Stemming off these uses, welding started to become increasingly popular for mechanical arms. (Wikipedia 2018)

2. Surgical Arms

The first use of surgical arms is happened in 1985 when a neurosurgical biopsy was performed. Although 1985 was the first time a surgery actually took place with a mechanical arm, scientists have been working on developing surgical arms for centuries. In 1495, Leonardo Da Vinci designed a complex robotic arm, which paved the way for surgical arms of the future. In 1990, the FDA allowed endoscopic surgical procedures to be done by the Automated Educational Substitute Operator (AESOP) system. This was not the only improvement the FDA made, however. While the AESOP system was more of a Computer Motion system, the first surgery system came about in 2000. In 2000, Leonardo Da Vinci's discoveries led the "Da Vinci

Surgery System" to become the first robotic surgery system approved by the FDA. (Wikipedia 2018)

3. Lifelike Mechanical Arms
Lifelike mechanical arms, along with ordinary human arms, are so similar that it may be hard to distinguish between the two. The reason for this is because a spray, that places a coat on the prosthetic arm, makes the arm look real. "This futuristic fantasy is beginning to become more of a reality. Scientists are even starting to create sleeve type artificial skins to keep a prosthetic arm looking like a normal arm. This will allow people with prosthetics to not feel self-conscious of their robotic arm." (Wikipedia 2018) This project is similar with this type mechanical arm which

1.4 The reason why would a person want to do this?

In general, there are more than one hundred reason tell you why a person what to research on Mechanical arm synchronization - an imitation of human arm action .

The most important point for one of reasons is tender human body, the technology of mechanical arm could tremendously decrease human problems. For example, Repetitive motion injuries, unlike other workplace injuries, don't occur because of accidents. They occur from doing a job properly thousands to tens of thousands of times. Pick and place activities create repetitive motion injury risk. To avoid losses in work hours and Worker's Compensation claims, think laterally. Robot arms streamline the pick and place tasks of your business by producing fast and accurate results that leave higher value tasks to the people. Using the the technology of mechanical arm mimic human arm action could let robot arm remember and repeat those kinds of pick and place works, mechanical arm just need to change it joints servo motors rather than spent a lot of money in hospital.

Another point of reasons which people would never ignore is that this technology could make people easier to use robot arm and promote the development of mechanization. This technology will let more and more people touch and use mechanical arm even though they have no idea how does mechanical arm driven and how to programming for it. Everyone can easy control the arm and do something works are dirty, arduous and simplex. For instance, those people who has disability or injures and somebody hard to do some toilsome works can use this technology to control mechanical arm easily, they just use their arm to show the simple posture without any hard exercise or strength.

1.5 Objectives

This project will use two cameras to capture video of human arm action. This data will be analyzed by the proposed system. This application will process and analysis the source of camera video, then the application will output the information from this

analysis to the controlling circuit board using serial communications Interface or WIFI interface. After receiving the data from PC, the controlling circuit board, with code previously downloaded in its STM32 chip, will analysis the data and change different pins signal (PWM Signal) to change the mechanical arm joints so that the mechanical arm can change its posture. In this way, this mechanical arm could implement the human action tracking and imitating.

Chapter 2: Research

2.1 OpenCV

This chapter will introduce some necessary OpenCV function's algorithm and how do them implemented. Here is list of mentioned algorithm.

- Tracking feature points in video
- Extracting the foreground objects in video
- The Mixture of Gaussian method

Tracking feature points in video

To start the tracking process, the first thing to do is to detect feature points in an initial frame. You then try to track these points in next frame. You must find where these points are now located in this new frame. In this project, we will try to track hand's position. Obviously, since we are dealing with a video sequence, there is a good chance that the hand on which the feature points are found has moved(the motion can also be due to camera motion). Therefore, you must search around a point's previous location in order to find its new location in the next frame. This is what accomplishes the cv::calcOpticalFlowPyrLK function. (Laganiere 2001)

Extracting the foreground objects in video

When a fixed camera observes a scene, the background remains mostly unchanged. In this case, the interesting elements are the moving objects that evolve inside this scene. In order to extract those foreground objects, we need to build a model of the background, and then compare this model with a current frame in order to detect any foreground objects. (Laganiere 2001) This is what we will do in this recipe. Foreground extraction is a fundamental step in intelligent surveillance applications and it is essential process of image processing.

The Mixture of Gaussian method

The preceding simple method to extract foreground objects in a scene works well for simple scenes showing a relatively stable background. However, in many situations, the background scene might fluctuate in certain areas between different values, thus causing frequent false foreground detections. These might be due to, for example, a moving background object(for example, tree leaves) or to glaring effect(for example, on the surface of water). In order to cope with this problem, more sophisticated background modeling methods have been introduced. (Bradski and Kaehler 2008) And The Mixture of Gaussian method is one of these algorithms. It proceeds in a way similar to the method presented in this recipe with the following additions:
The method maintains more than one model per pixel(that is, more than one running average). This way, if a background pixel fluctuates between, let' s say, two values, two running average are then stored, A new pixel value will be declared as foreground only if it doesn't belong to any of the maintained models. This more sophisticated algorithm is obviously more complex to implement than our simple background/foreground segmentor. (Bradski and Kaehler 2008) Fortunately, an

OpenCV implementation exists called cv::BackgroundSubtractorMOG2 and is defined as a subclass of the more general cv::BackgroundSubstractor2 class. (But the following code doesn't work anymore, next to find out the new versions function - Ptr<BackgroundSubstractorMOG2> mog= createBackgroundSubstractorMOG2();
It is successful)

2.2 Hand Tracking And Gesture Detection

The aim of the project was to device a program that is able to detect out hands, track them in real time and perform some guesture recognition. It is do be done with simple signal processing performed on images obtained from USB camera. This section will introduce more details of realization process.

- Detecting Background
- Background Subtraction
- Contour Extraction
- Convex Hull and Defects
- Tracking and Finger Detection

Detecting Background

Given the feed from the camera, the first thing to do is to subtract the background. We use running average over a sequence of images to get the average image which will be the background.

$$CurBG[i][j]=alpha\ CurBG[i][j] + (1- alpha\ )CurFrame[i][j]$$

This equation works because of the assumption that the background is mostly static. Hence for those stationary item , those pixels aren't affected by this weighted averaging and alpha x+(1-alpha)x=x . Hence those pixels that are constantly changing isnt a part of the background, hence those pixels will get weighed down. Hence the stationary pixels or the background gets more and more prominent with every iteration while those moving gets weighed out. Thus after a few iterations , you get the above average which contains only the background. (Srinivasan 2013)

Background Subtraction

A simple method to start with is we can subtract the pixel values. We use an OpenCV inbuilt background subtractor based on a Gaussian Mixture-based Background/ Foreground Segmentation Algorithm.Background subtraction involves calculating a reference image, subtracting each new frame from this image and thresholding the result which results is a binary segmentation of the image which highlights regions of non-stationary objects . (Srinivasan 2013) Then we use erosion and dilation to enhance the changes to make it more prominent. After these operation, video source become more clear and easy to analyze.

Contour Extraction

Contour extraction is use OpenCV's library extraction function. It uses a canny filter. You can tweak parameters to get better edge detection. In this step, we could find one more contour in frame so that we should choose proximate one to represent hand's contour.

Convex Hull and Defects

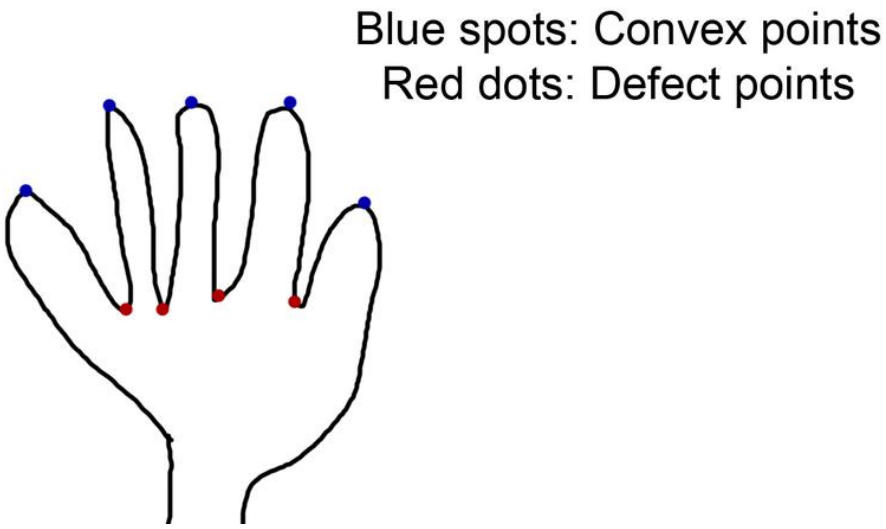


Figure 2.2.1: Hand's convex points and defect points
Source: (Srinivasan 2013)

Now given the set of points for the contour, we find the smallest area convex hull that covers the contours . And we consider this contour as our hand contour. "The observation here is that the convex hull points are most likely to be on the fingers as they are the extremities and hence this fact can be used to detect no of fingers." (Srinivasan 2013) But cause our whole arm is in the camera, there will be other points of convexity too. So we find the convex defects ie, between each arm of the hull, we try to find the deepest point of deviation on the contour.

Tracking and Finger Detection

Thus the defect points are most likely to be the center of the finger valleys as pointed out by the picture. Now we find the average of all these defects which is definitely bound to be in the center of the palm but its a very rough point. So we average out and find this rough palm center. Next we assume that the palm is angled in such a way that its roughly a circle. So in order to find the specific palm center, we take 3 points that closes to the rough palm center and find the circle center and radius of the circle passing though these 3 points. We can consider this circle center as palm canter. Due to noise, this center always jump in the frame, in order to stabilize it , we take an average over a few iterations. Thus the radius of the palm is a indication of the depth of the palm and we know the center of the palm . Using this we can track the position of the palm in real time and even know the depth of the palm using the radius. The next challenge is detecting the number of fingers. We use a couple of observations to

do this. For each maxima defect point which will be the finger tip, there will be 2 minimal defect points to indicate the valleys. Hence the maxima and the 2 minimal defects should form a triangle with the distance between the maxima and the minimas to be more or less same. Also the minima should be on or pretty close to the circumference of the palm. We use this factor too. " Also the the ratio of the palm radius to the length of the finger triangle should be more or less same . Hence using these properties, we get the list of maximal defect points that satisfy the above conditions and thus we find the number of fingers using this. If no of fingers is 0 , it means the user is showing a fist."(Srinivasan 2013)

Chapter 3: Design of Proposed Solution

3.0 Introduction

There are four parts to the design of the proposed solution. First part is the design of a controlling circuit board for basic controlling preparation. Second part is the design of a framework of video catching and analysis. Third part is the design a communication link to be used for communicating between the PC application and the controlling circuit board. Fourth part is controlling the mechanical arm by circuit board.

3.1 Controlling circuit board design

The most important part of circuit board design are the schedule pins of the STM32 chip. This circuit board use STM32F103RBT6 chip as main chip and its package is LQFP64 (cf. Figure 3.1) which means it has 64 pinouts. But we don't use all the pins in this circuit board.

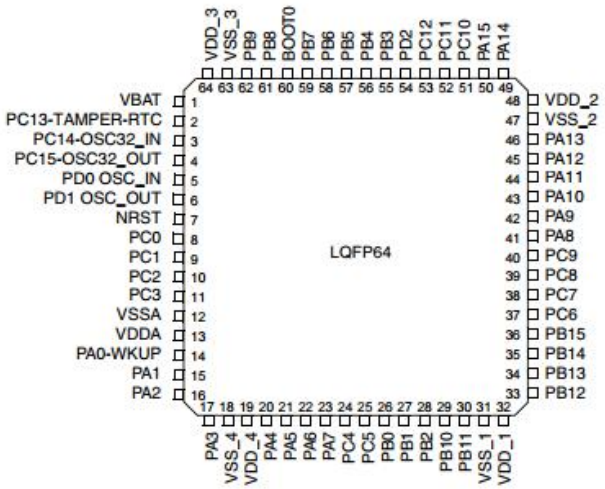


Figure 3.1: LQFP64 Package
Source: (STMicroelectronics 2013)

Manuscript of stm32 chip schedule

As shown in Figure 3.2, this is my original design of the pinout schedule. This circuit will use 52 pinouts of the STM32f103RBT6 chip, not all of pins. The primary function of stm32 is to output six or more PWM signal. Meanwhile, there are other assistant functions such as LED, switch, key, buzzer, WIFI interface, JTAG interface, USART and so on.

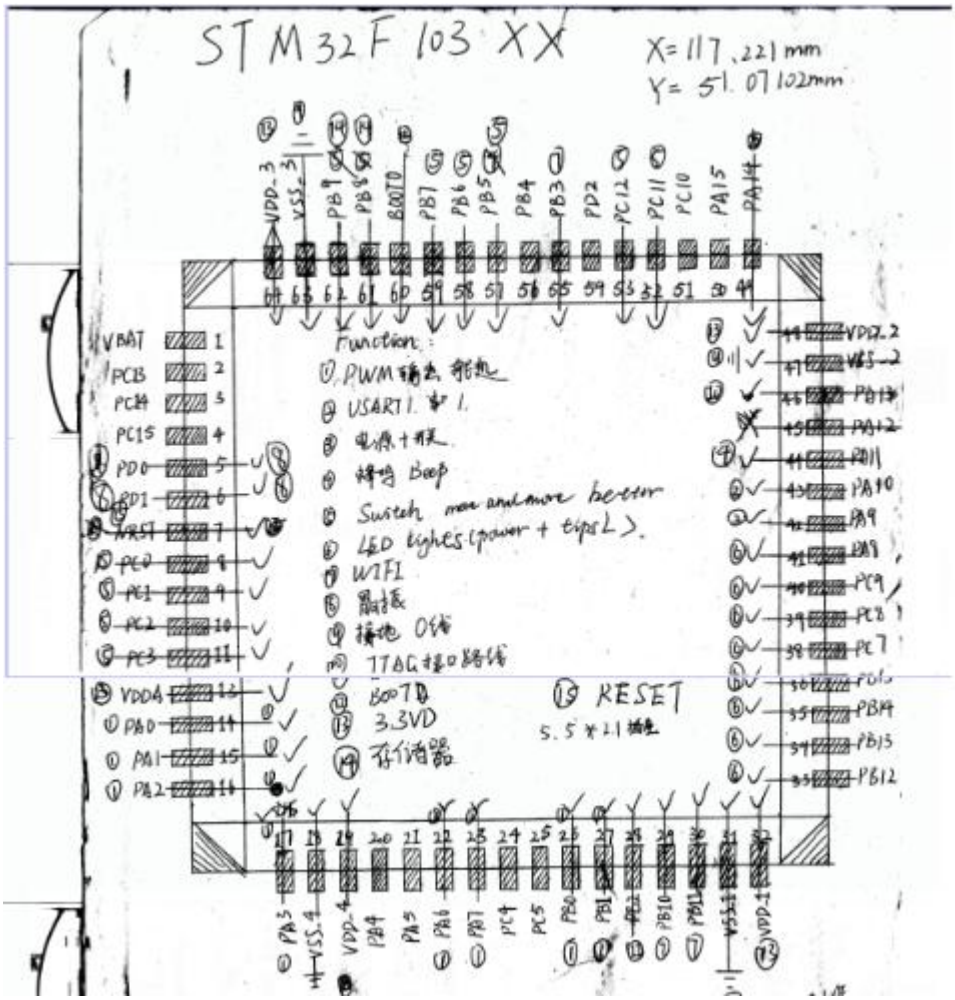


Figure 3.2: Pinout Schedule

Pinout schedule details

In this circuit board diagram there are 14 important parts (cf. Table 3.1)

- STM32F103RBT6 Processor,
- Reset circuit,
- BOOT0&BOOT1,
- decoupling capacitor,
- JTAG interface,
- RS232 serial interface,
- servo motor controlling circuit,
- low level buzzer,
- keys,
- power input, protection switch and indication,
- main power supply 5V transfer 3.3V,
- WIFI interface,
- LED circuit and
- EEPROM 24CXX.

LQFP64	Pin Name	Type	Main Function	Alternate Functions		Function Description
				Default	Remap	
5	OSC_IN	I	OSC_IN		PD0	crystal oscillator
6	OSC_OUT	O	OSC_OUT		PD1	crystal oscillator
7	NRST	I/O	NRST			reset
8	PC0	I/O	PC0	ADC12_IN10		Key
9	PC1	I/O	PC1	ADC12_IN11		Key
10	PC2	I/O	PC2	ADC12_IN12		Key
11	PC3	I/O	PC3	ADC12_IN13		Key
12	VSSA	S	VSSA			ground zero line
13	VDDA	S	VDDA			3.3VD
14	PA0-WKUP	I/O	PA0	TIM2_CH1		PWM signal
15	PA1	I/O	PA1	TIM2_CH2		PWM signal
16	PA2	I/O	PA2	TIM2_CH3		PWM signal
17	PA3	I/O	PA3	TIM2_CH4		PWM signal
18	VSS_4	S	VSS_4			ground zero line
19	VDD_4	S	VDD_4			3.3VD
22	PA6	I/O	PA6	TIM3_CH1		PWM signal
23	PA7	I/O	PA7	TIM3_CH2	TIM1_CH1	PWM signal
26	PB0	I/O	PB0	TIM3_CH3	TIM1_CH2	PWM signal
27	PB1	I/O	PB1	TIM3_CH4	TIM1_CH3	PWM signal
28	PB2	I/O	BOOT1/PB2			BOOT1
29	PB10	I/O	PB10	USART3_TX	TIM2_CH3	WIFI
30	PB11	I/O	PB11	USART3_RX	TIM2_CH4	WIFI
31	VSS_1	S	VSS_1			ground zero line
32	VDD_1	S	VDD_1			3.3VD
33	PB12	I/O	PB12			LED
34	PB13	I/O	PB13			LED
35	PB14	I/O	PB14			LED
36	PB15	I/O	PB15			LED
37	PC6	I/O	PC6		TIM3_CH1	LED
38	PC7	I/O	PC7		TIM3_CH2	LED
39	PC8	I/O	PC8		TIM3_CH3	LED
40	PC9	I/O	PC9		TIM3_CH4	LED
41	PA8	I/O	PA8			LED
42	PA9	I/O	PA9	USART1_TX		USART1
43	PA10	I/O	PA10	USART1_RX		USART1
44	PA11	I/O	PA11	TIM1_CH4		buzzer
46	PA13	I/O	JTMS/SWDIO		PA13	JTAG interface
47	VSS_2	S	VSS_2			ground zero line
48	VDD_2	S	VDD_2			3.3VD
49	PA14	I/O	JTCK/SWCLK		PA14	JTAG interface
52	PC11	I/O	PC11		USART3_RX	Key

53	PC12	I/O	PC12		USART3_CK	Key
55	PB3	I/O	JTDO			WIFI
57	PB5	I/O	PB5			Key
58	PB6	I/O	PB6			Key
59	PB7	I/O	PB7			Key
60	BOOT0	I	BOOT0			BOOT0
61	PB8	I/O	PB8	TIM4_CH3	I2C1_SCL/ CANRX	AT24LC0x storage
62	PB9	I/O	PB9	TIM4_CH4	I2C1_SDA/ CANTX	AT24LC0x storage
63	VSS_3	S	VSS_3			ground zero line
64	VDD_3	S	VDD_3			3.3VD

(I = input, O = output, S = supply).

Table 3.1: More Detailed Design

3.2 PC application framework design

There are four aspects to the proposed application:

1. Catch source of two USB camera.
2. Process and analyze the source of video.
3. Communicate with circuit board.
4.Display the statement of running result.

1. Catch source of two USB camera.

We can use OpenCV library function: VideoCapture to catch and extract the source of two cameras.

```
VideoCapture cap_under(0);
VideoCapture cap_side(2);
```

2. Process and analyze the source of video

This is the most difficult and important part of this part, which contains image processing, noise and background subtraction, morphological operation and analysis. More details will be given in Chapter 4.

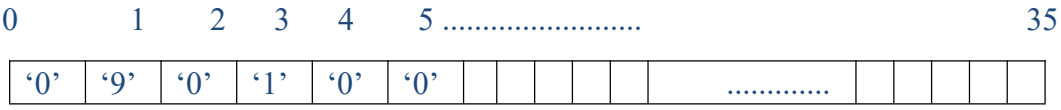
3. Communication with circuit board

This project will use serial communication technology to communicate between the PC application and the STM32 circuit board. The application will use serial port to send or receive the data.

4. Display the results while running
This application needs a display page to show running results and statements on video source from camera, processed video source and mechanical arm joint's angle and so on.

3.3 Communication Protocol

A pithy communication protocol could make communication much more convenient. So a PC application and circuit board should have a simple communication protocol. In this project, the application will send a char type array whose length is 36. Three char bits represent a 3-digit number. For example, for the array shown here the characters in position 0, 1, 2 are '0', '9' and '0', which represents 090 as a 3-digit number.



This array will bring twelve 3-digit numbers which represent the start angle of each joint and the end angle of each joint, for each of the six joints. The circuit board side will accord with this communication protocol to decode the data and control the mechanical arm's joints.

3.4 Using circuit board to control mechanical arm

The functions of the circuit board are to receive data which was sent from PC application and drive joints of mechanical arm to control arm action. More details will be given in Chapter 4.

Chapter 4: Implementation

4.0 Introduction

This chapter gives details of the design of the controlling circuit board (cf. Section 4.1) , the controlling application implementation (cf. Section 4.2) and the STM32 circuit board program implementation (cf. Section 4.3).

4.1 Design of Controlling Circuit Board

Table 4.1 gives the circuit board design components. It lists each component's serial number, name, weld package, welding pad number, quantity, parameter, tab and polarity (if it has polarity).

Serial Number	Name	Package	Welding Pad Number	Quantity	Parameter	Tab	Polarity
R1, R2, R15, R16, R12, R29, R11, R21, R22	Resistance	0805	2	9	10k	103	
R17, R14	Resistance	0805	2	2	1K	102	
R18	Resistance	0805	2	1	220	220	
AR1, AR2, AR3, AR4	Network Resistor	0603	8	4	10K*4	103	
R13	Resistance	0805	2	1	1M	105	
C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C11, C12, C13, C14, C17, C19, C21, C22	Capacitance	0805	2	18	0.1uF		
C15, C16	Capacitance	0805	2	2	22pF		
C18, C20	Capacitance	YJ-TAN-7343	2	1	1000uF		
SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8, SW9, SW10	Key	Double Pad	2	10	6 * 3.6		
Q1	Bipolar Junction Transistor	SOT23	3	1	8550	2TF	Polar direction
LED1, LED2, LED3, LED4, LED5, LED6, LED7, LED8, LED10. LED10	LED	0805	2	10	Red		Polar direction
U1	IC	SO16NB	16	1	MAX3232		Polar direction
U2	IC	TO-252L	4	1	AMS1117-3.3		Polar direction

U3	Double line Female Header	WIFI-8P	8	1	WIFI		Polar direction
U4	IC	SO8NB	8	1	AT24CXX		Polar direction
U6	IC (STM32)	LQFP-64	64	1	STM32F103 RBT6		
Y1	Quartz crystal resonator	XTAL	2	1	8MHz		
FUSE2	Fuse	1812	2	1	117P		
JP3	Source jack-socket	DCIN	3	1	5.5 * 2.1		Polar direction
JP1	DR9 female header	DSUB1.385	11	1	DR9 female header		Polar direction
JP2	Pin header	HX2.54-5P	5	1	2.54		Polar direction
S1	Toggle switch	DH-K6-2	8	1	Toggle switch		
P1, P2, P3	Pin Header	HDR1X8	8	3	1 * 8		
D1		DH-1N4007	2	1	1N4007	M7	Polar direction
LS1	Buzzer	BEEP001	2	1	3V		Polar direction

Table 4.1: Circuit Board Design Components

In this project, I used Altium Designer 9 to draw a schematic circuit diagram (cf. Figure 4.1 and Figure 4.2).

STM32F103R Processor: ARM 32-bit Cortex™-M3 CPU Core
In this project, a STM32 processor is one of essential materials that will drive the circuit board to control the mechanical arm. I use the STM32F103RB as my processor. The STM32F103xx medium-density performance line family incorporates the high performance ARM Cortex™-M3 32-bit RISC core operating at 72 MHz frequency, with high speed embedded memories (Flash memory up to 128 K bytes and SRAM up to 20 K bytes), and an extensive range of enhanced I/Os and peripherals connected to two APB buses.

All devices offer two 12-bit ADCs, three general purpose 16-bit timers plus one PWM timer, as well as standard and advanced communication interfaces: up to two I2Cs and SPIs, three USARTs, an USB and a CAN. These feature make the STM32F103 suitable for wide range of applications such as motor drives, application control, medical and handheld equipment, PC and gaming peripherals, GPS platforms, industrial applications, PLCs, inverters, printers, scanners, alarm systems, video intercoms, and HVACs.

Reset circuit: Once you click the key in reset circuit, the program in STM32 will restart.

BOOT0&BOOT1: Pins BOOT0 and BOOT1 select how the STM32 starts. Both BOOT0 and BOOT1 connects to ground and with a 10K resistance, which can easy to control how to start (cf. Table 4.2)

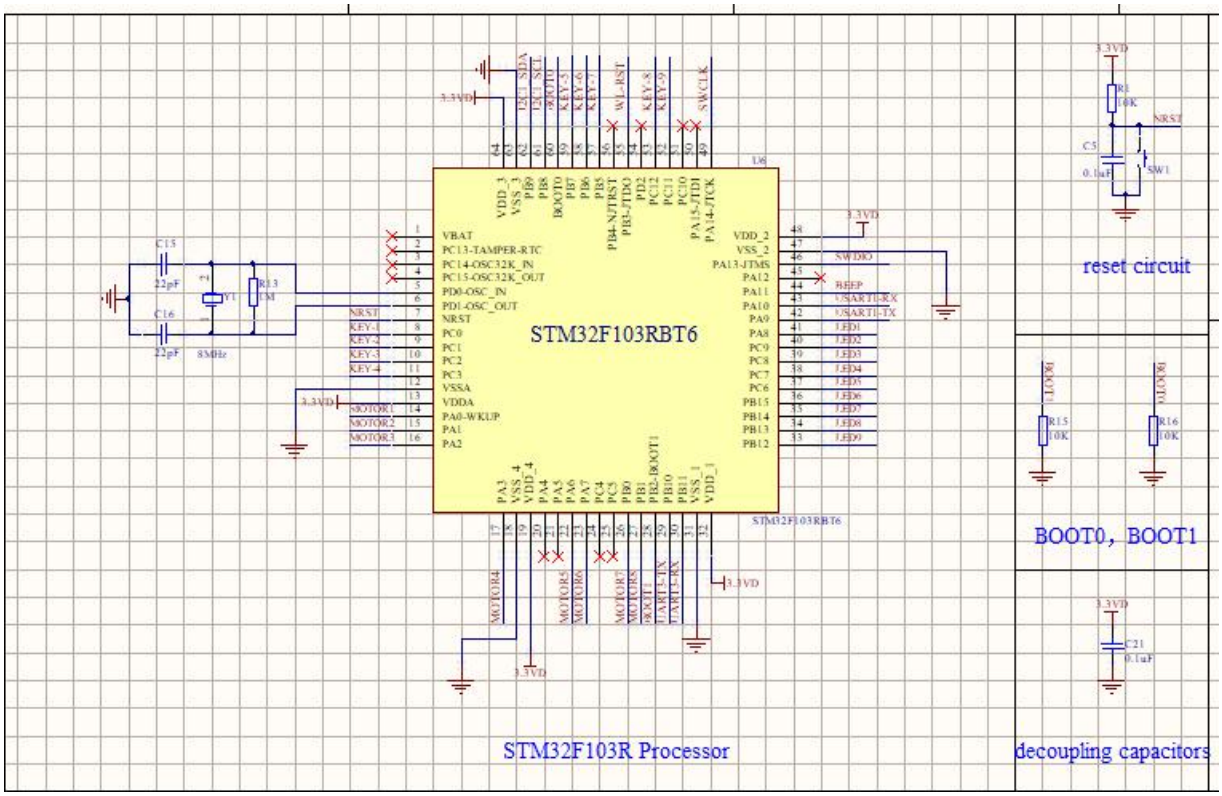


Figure 4.1: Schematic Circuit Board

BOOT1	BOOT0	Boot mode
X	0	User Flash memory
0	1	System memory (bootloader)
1	1	Embedded SRAM

Table 4.2: Pins BOOT0 and BOOT1

Decoupling capacitors: The purposes of decoupling capacitors to reduce or cut electric current fluctuations.

JTAG interface: This is interface point where we download the programming to the STM32 core chip.

RS232 serial interface: Communication interface for circuit board and PC or other device.

Keys: Designed for testing various function written in C (different arm action controlling codes).

Servo motor controlling interface: There are eight line pin headers which can control eight servo motors simultaneously. But this project will use just six pin headers to control six joint motors.

Low level buzzer: Buzzer is an audio signalling device. Typical uses of buzzers and beepers include alarm devices, timers, and confirmation of user input such as a key click or communication signal.

Power input, protection switch and indication: Main switch of power supply, there is a switch and a fuse to avoid voltage overloading which protects whole circuit components on the board. And a LED light can indicate whether circuit board has a suitable power supply.

Main power supply 5V transfer 3.3V: Main power will be translate 5V into 3.3V which is suitable voltage for STM32 chip.

WIFI interface: Alternative communication interface for circuit board. It can use WIFI signal to communicate with whose device who can receive and connect this WIFI signal.

LED circuit: There are nine LED lights for code testing.

EEPROM 24CXX: It is an additional ATM storage which can store the data even though there is no power supply.

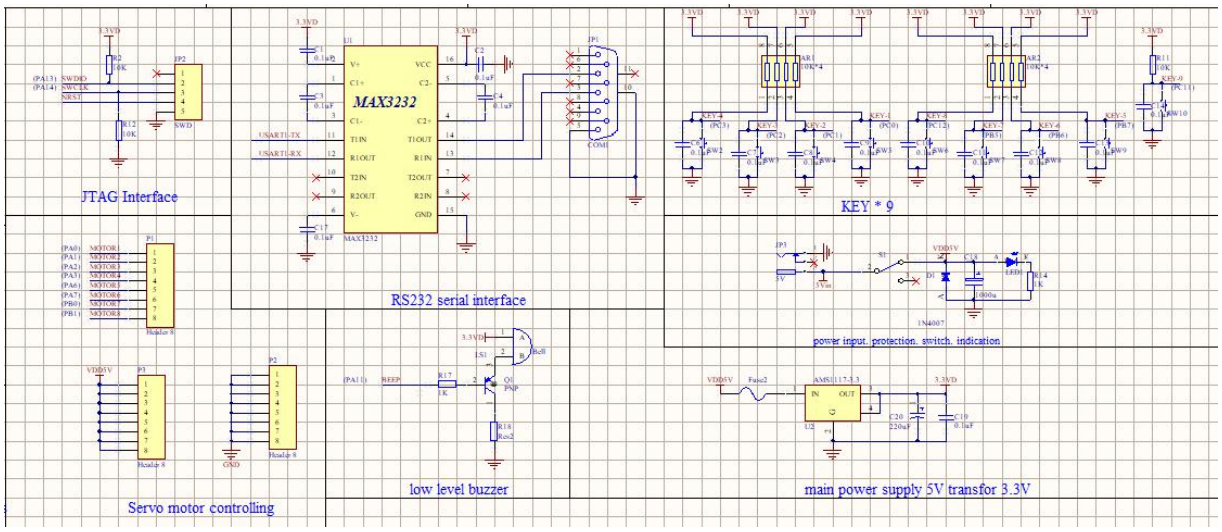


Figure 4.2: Schematic Circuit Board

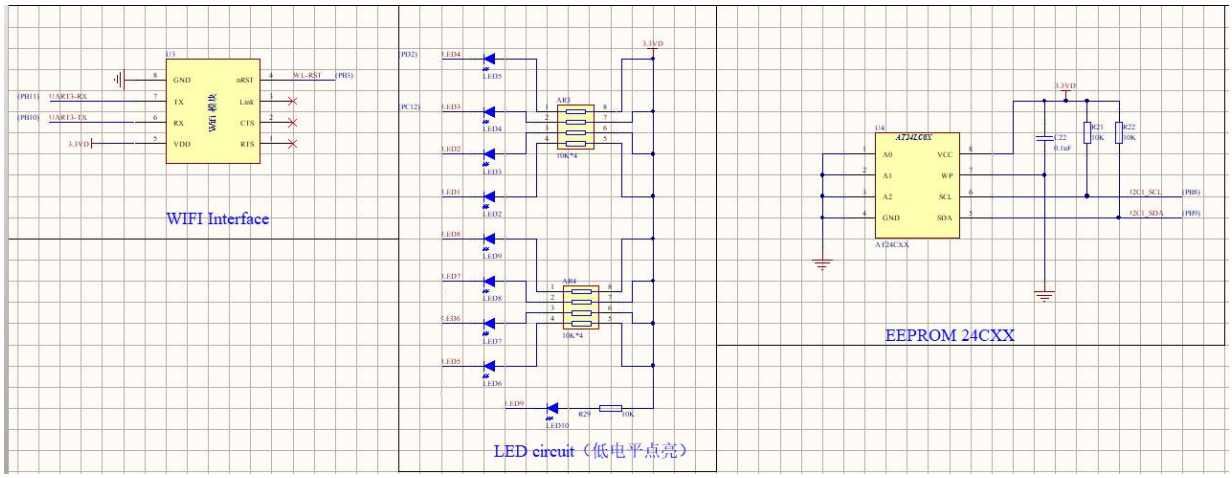


Figure 4.3: Printed Circuit Board Diagram

In this project, the key part of the mechanical arm is driven by the circuit board. I used Altium Designer to draw the PCB file according to the schematic diagram. Figure 4.4 shows the Top Layer of the circuit board while Figure 4.5 shows the Bottom Layer of the circuit board.

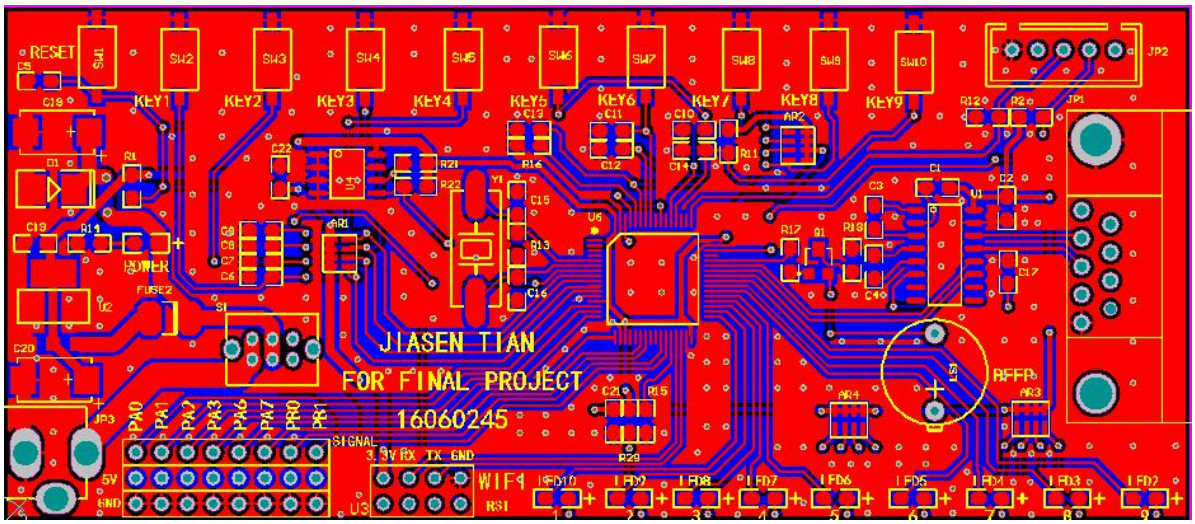


Figure 4.4: Top Layer of Circuit Board

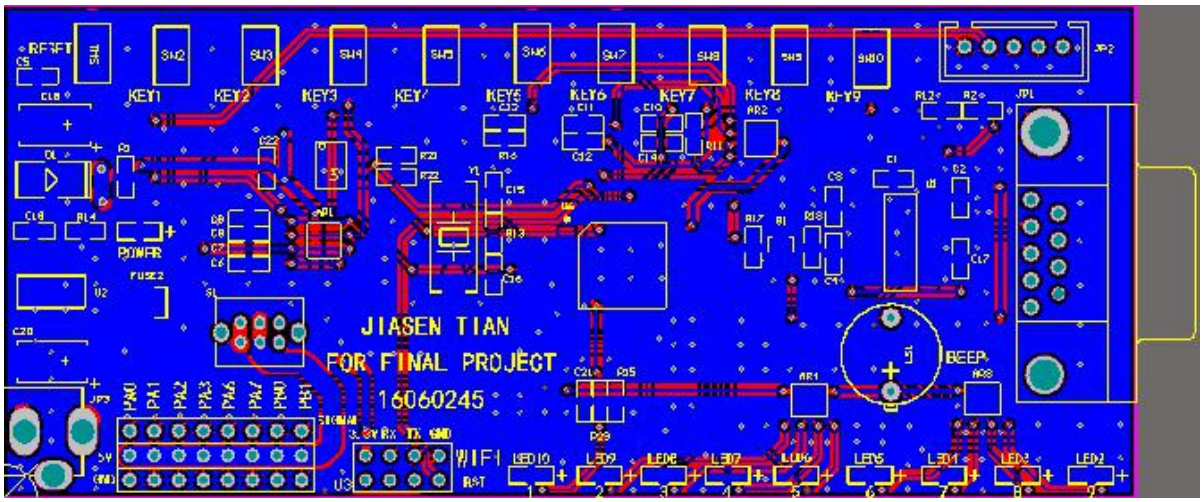


Figure 4.5: Bottom Layer of Circuit Board

After finishing the circuit board PCB file, I send my PCB file to J&C CO. LTD which is a famous circuit board manufacturer in China. One week later, I get my new empty circuit board (cf. Figure 4.6).

Figure 4.6: Empty Circuit Board

Next, the various components were welding onto the circuit board (cf. Figure 4.7).

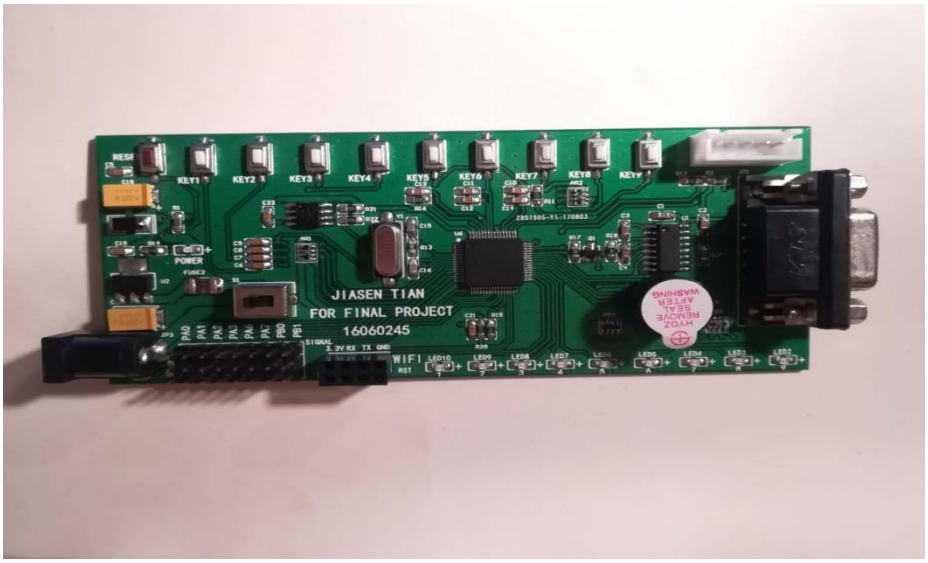


Figure 4.7: Circuit Board with Welded Components

The function of each key component:

- Pin PA0 will drive Joint 1 (cf. Section 4.2);
- Pin PA1 will drive Joint 2;
- Pin PA2 will drive Joint 3;
- Pin PA3 will drive Joint 4;
- Pin PA6 will drive Joint 5;
- Pin PA7 will drive Joint 6.
- Ten LED lights are used for testing functions and data.
- Ten Key switches are also used for testing functions and switching the code fragments.
- Testing of each component was successful.
- This controlling circuit board is now accessible to control the mechanical arm.

4.2 Implementation of Controlling Application

The controlling application of this project (cf. Figure 4.8) is implemented in Qt 5.5. There are two parts to the implementation. The first part is the page and the second part is the internal algorithm implementation.

The display page was implement by Qt Creator. Qt Creator is a useful designing framework application that makes creating a display page or HTML page very easy.

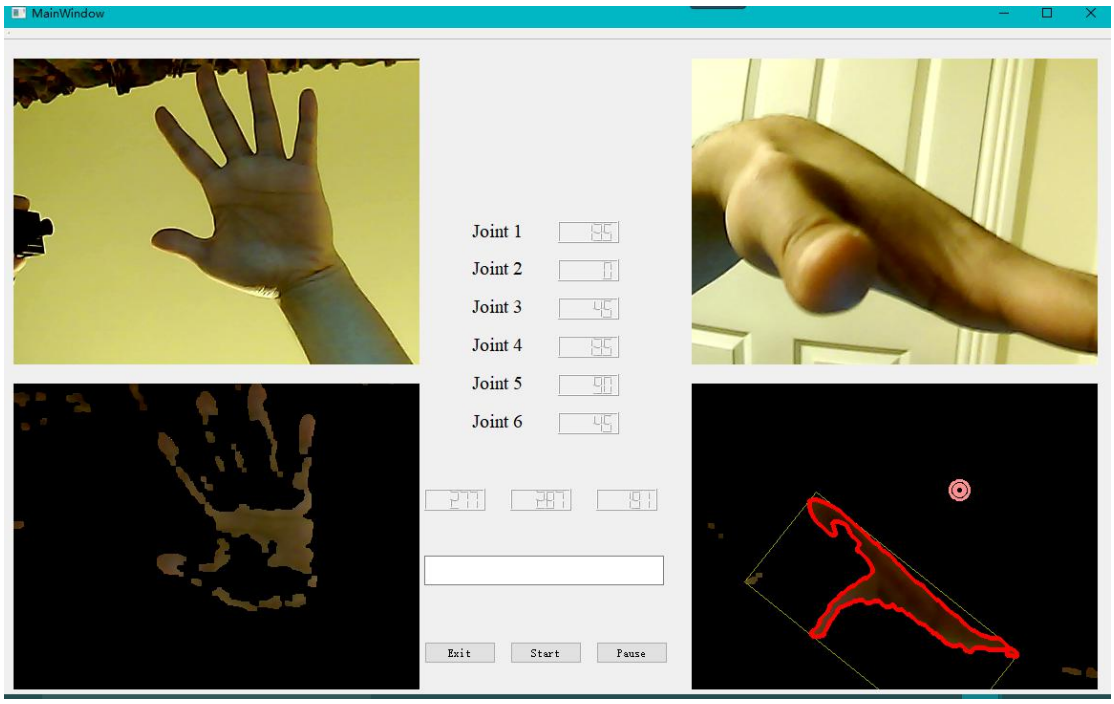


Figure 4.8: Controlling Application

On the application display page there are six joint angle display areas where will show the angle of each arm joint and three LCDNumber areas will show the three-dimensional coordinate of the centre of the detected hand.

At bottom of the page, there are three buttons:

- Exit button - which closes the application once clicked;
- Start button - starts the application once clicked; and
- Pause button – which pauses the application when application is processing.

In the left hand side and on the right hand side there are areas to display video. The top frames display the two original video sources which are caught from two cameras. The bottom frames show video sources that are processed by application algorithm.

Overall this application catches the source of two cameras, processes the sources, analyzing the shape of hand, getting position of hand in 3D coordinate. Finally, according the position of human arm, the application will calculate the angles of mechanical arm joints and return the result to circuit board by serial communication. This section will give more detail of implementation for these parts.

1. Catch two camera sources

VideoCapture is a OpenCV library class, it can access manager priority to get the USB port source (USB camera). The application accesses the USB ports whose numbers are 0 and 2.

```
VideoCapture cap_under(0);
VideoCapture cap_side(2);
```

After catching camera source, the application stores video source into a matrix arrays(frame_under and frame_side) every second. If matrix array frame_under or frame_side is empty, it means PC application failed to access USB camera source and the application will output an error tip and terminate the whole process.

```
cap_under >> frame_under;
cap_side >> frame_side;

if( frame_under.empty())
{
    cout << " < < <  First camera failed to catch!  > > > ";
    break;
}
if( frame_side.empty())
{
    cout << " < < <  Second camera failed to catch!  > > > ";
    break;
}
```

2. Processing video source

The objective of processing is that using OpenCV functions and algorithms to subtract the background and noise according to skin-color detection.

When application succeeds in catching the video, it will continue to process the source. The first step uses the OpenCV function median Blur() to remove Salt And Pepper Noise in camera video.

```
//median Blur to remove Salt And Pepper Noise.
medianBlur(frame_under, frame_under, 5);
medianBlur(frame_side, frame_side, 5);
//
frame_under.copyTo(temp_under);
frame_side.copyTo(temp_side);

cvtColor(temp_under, temp_under, CV_BGR2YCrCb);
cvtColor(temp_side, temp_side, CV_BGR2YCrCb);

split(temp_under, channels_under);
split(temp_side, channels_side);

Y_under = channels_under.at(0);
Cr_under = channels_under.at(1);
Cb_under = channels_under.at(2);

Y_side = channels_side.at(0);
Cr_side = channels_side.at(1);
Cb_side = channels_side.at(2);
```

Second step is get a copy matrix arrays and transform values into YCrCb color space. This makes it much easier to do skin-color detection. Y represents the brightness, Cr

and Cb are representatives of chromaticity which has a small relationship with brightness. In this project, Cr and Cb values are the most important to distinguish human arm from the background. Splitting the YCrCb matrix into three different space will make processing easier.

The third step is to traverse the YCrCb matrix arrays to detect color of human hand which can help us distinguish human hand from noise and background as shown in the following figure.

```
Mat dstTemp_under(frame_under.rows, frame_under.cols, CV_8UC1);
Mat dstTemp_side(frame_side.rows, frame_side.cols, CV_8UC1);

for (int j = 1; j < Y_under.rows - 1; j++)  {
    uchar* currentCr = Cr_under.ptr< uchar>(j);
    uchar* currentCb = Cb_under.ptr< uchar>(j);
    uchar* current = dstTemp_under.ptr< uchar>(j);
    for (int i = 1; i < Y_under.cols - 1; i++)  {
       if ((currentCr[i] > 137) && (currentCr[i] < 175) && (currentCb[i] > 100) && (currentCb[i] < 118))
          current[i] = 255;
       else
          current[i] = 0;
       }
}

for (int j = 1; j < Y_side.rows - 1; j++)  {
    uchar* currentCr = Cr_side.ptr< uchar>(j);
    uchar* currentCb = Cb_side.ptr< uchar>(j);
    uchar* current = dstTemp_side.ptr< uchar>(j);
    for (int i = 1; i < Y_side.cols - 1; i++)  {
       if ((currentCr[i] > 137) && (currentCr[i] < 175) && (currentCb[i] > 100) && (currentCb[i] < 118))
          current[i] = 255;
       else
          current[i] = 0;
       }
}
```

The range of human skin in Cr value is about 137~175, and range of Cb value is about 100 ~ 118. We assume that there is no other hand, human body or anything chromaticity is nearly like human skin. So in my code, I traverse all the pixels in YCrCb matrix array to find the color of which pixels are close to human skin color. If this pixel's Cr value is greater than 137 and less than 175 and if its Cb value is greater than 100 and less than 118, we will assume that this pixel is part of human hand area. So if this pixel point is valid, we will set it's Y value to 255 (bright). Othewise, this pixel's Y value will be set as 0 (dark).

The fourth step is a morphological operation to remove noise and make the boundary of hand much clearer. This project uses erode and dilate operations to process matrix.

- Erosion: This operation is the sister of dilation. What this does is to compute a local minimum over the area of the kernel.
- Dilation: This operation consists of convoluting an image A with some kernel (B), which can have any shape or size, usually a square or circle.

After applying the morphological operations, it is time to analyze the source.

```
Mat element = getStructuringElement(MORPH_RECT, Size(3,3));
//erode operation
erode(dstTemp_under, dstTemp_under, element);
erode(dstTemp_side, dstTemp_side, element);
//morphological operation
morphologyEx(dstTemp_under, dstTemp_under, MORPH_OPEN, element);
morphologyEx(dstTemp_side, dstTemp_side, MORPH_OPEN, element);
//dilate operation
dilate(dstTemp_under, dstTemp_under, element);
dilate(dstTemp_side, dstTemp_side, element);
morphologyEx(dstTemp_under, dstTemp_under, MORPH_CLOSE, element);
morphologyEx(dstTemp_side, dstTemp_side, MORPH_CLOSE, element);
```

3 Find contour in the matrix array and analyze the position of palm center

The first step is to find all suspected contours in processed video source as follows:

```
contours_1.clear();
contours_2.clear();
hierarchy_1.clear();
hierarchy_2.clear();
filterContours_1.clear();
filterContours_2.clear();
//Find contour of hand
findContours(dstTemp_under, contours_1, hierarchy_1, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
findContours(dstTemp_side, contours_2, hierarchy_2, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);
```

After initialization and clearance, we use the OpenCV function findContours() to find hand contours in matrix array which we process first. The contour points found will be stored in a Vector object: contours_1 and contours_2. The hierarchical relationship will be stored in Vector object: hierarchy_1 and contours_2.

The second step is to distinguish hand contours from noise contours. We assume that the human hand size in the video source should not be more than 6000 pixels, so we ignore all the noise contours whose pixel size are less than 6000 as follows:

```
for (size_t i = 0; i < contours_1.size(); i++)
{
    if (fabs(contourArea(Mat(contours_1[i]))) > 6000)
    {
        vector<vector<Point> > tcontours;
        tcontours.push_back(contours_1[i]);
        filterContours_1.push_back(contours_1[i]);
```

The third part is the key of the whole project, locating the center of the hand palm and analyzing number of fingers in video. These are achieved by the following steps:

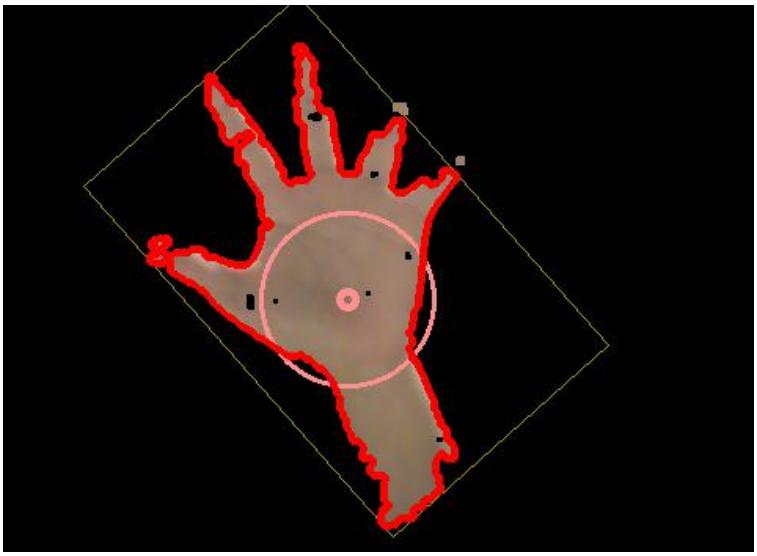
1. Find minimum area rectangle to enclosed hand

We assume that the minimum area rectangle is probably null, so we use OpenCV function: minAreaRect() to get the minimum rectangle's four vertex points as follows:

```
//Find minimum area rectangle to enclose hand
RotatedRect rect=minAreaRect(Mat(tcontours[0]));

vector<Vec4i> defects;
if(hullsI[0].size() > 0){
   Point2f rect_points[4];
   rect.points(rect_points);
   for(int j = 0; j < 4; j++){
     line(dst_under, rect_points[j], rect_points[(j + 1)%4],Scalar(0,155,155), 1, 8);
   }
```

After finding four vertex points, we will draw lines, one by one to link neighboring points and finally we can get an irregular rectangle where human hand shape is located as shown in the following diagram



2. Locating palm center in video frame.

Locating the palm center is also a difficult task. It is essential to find a rough palm center to locate the human hand position.

First, we use the OpenCV function convexityDefects() to find defect points of hand contours. Those defects points usually are finger valley points which are near the palm center, so we can record their positions and calculate an average point. We can assume this average point as a rough palm center as follows:

```
Point rough_palm_center;
convexityDefects(tcontours[0], hullsI[0], defects);
if(defects.size() >= 3){
   vector<Point> palm_points;
   for(int j = 0; j < defects.size(); j++){
      int startidx = defects[j][0];
      Point ptStart(tcontours[0][startidx]);
      int endidx = defects[j][1];
      Point ptEnd(tcontours[0][endidx]);
      int faridx = defects[j][2];
      Point ptFar(tcontours[0][faridx]);
      rough_palm_center += ptStart + ptEnd + ptFar;
      palm_points.push_back(ptFar);
      palm_points.push_back(ptStart);
      palm_points.push_back(ptEnd);
   }
   rough_palm_center.x/=defects.size()*3;
   rough_palm_center.y/=defects.size()*3;
   Point closest_pt=palm_points[0];
```

Second, we can use the rough palm center to find three closest defects points which could draw a circle. The center of circle will be a better and more accurate palm center. The circle' s size is also similar to human hand size.

```
vector<pair<double, int> > distvec;

for(int i = 0; i < palm_points.size(); i++){
   distvec.push_back(make_pair(dist(rough_palm_center, palm_points[i]), i));
}

sort(distvec.begin(), distvec.end());
pair<Point, double> soln_circle;

for(int i = 0; i + 2 < distvec.size(); i++){
   Point p1 = palm_points[distvec[i + 0].second];
   Point p2 = palm_points[distvec[i + 1].second];
   Point p3 = palm_points[distvec[i + 2].second];
   soln_circle = circleFromPoints(p1, p2, p3);
   if(soln_circle.second!=0)
     break;
}
palm_centers_1.push_back(soln_circle);
if(palm_centers_1.size() > 10){
   palm_centers_1.erase(palm_centers_1.begin());
}
Point palm_center;
double radius=0;
for(int i = 0; i < palm_centers_1.size(); i++){
    palm_center += palm_centers_1[i].first;
    radius += palm_centers_1[i].second;
}
palm_center.x/=palm_centers_1.size();
palm_center.y/=palm_centers_1.size();
radius/=palm_centers_1.size();
```

If there are no more than 10 palm centers found, the final center and size of hand will use the average value of those points. But most times we only find one center point.

Third, it is essential to figure out the number of fingers in the video so that we can use this value to check whether the hand is making a fist or opening.

```
for(int j=0;j<defects.size();j++)
{
    int startidx=defects[j][0]; Point ptStart( tcontours[0][startidx] );
    int endidx=defects[j][1]; Point ptEnd( tcontours[0][endidx] );
    int faridx=defects[j][2]; Point ptFar( tcontours[0][faridx] );
    //X o--------------------------o Y
    double Xdist=sqrt(dist(palm_center,ptFar));
    double Ydist=sqrt(dist(palm_center,ptStart));
    double length=sqrt(dist(ptFar,ptStart));

    double retLength=sqrt(dist(ptEnd,ptFar));
    //Play with these thresholds to improve performance
    if(length<=3*radius&&Ydist>=0.4*radius&&length>=10&&retLength>=10&&max(length,retLength)/min(length,retLength)>=0.8)
        if(min(Xdist,Ydist)/max(Xdist,Ydist)<=0.8)
        {
            if((Xdist>=0.1*radius&&Xdist<=1.3*radius&&Xdist<Ydist)||(Ydist>=0.1*radius&&Ydist<=1.3*radius&&Xdist>Ydist)){
                //line(frame_under, ptEnd, ptFar, Scalar(0,0,0), 1 );
                no_of_fingers_under++;
            }
        }
}
no_of_fingers_under = min(5,no_of_fingers_under);
cout<<"NO OF FINGERS IN UNDER CAMERA: "<<no_of_fingers_under<<endl;
```

The ratio of the palm radius to the length of the finger triangle should be very similar. So using these properties, we get the list of maximal defect points that satisfy the above conditions and thus we find the number of fingers. If the number of fingers is zero this means human hand is making a fist.

This is one camera functions descriptions, the other camera functions are similar.

4. Mechanical arm joint angle computational algorithm

For the mechanical arm joint angle algorithm we use the hand's 3D coordinates and number of fingers as function parameters. The function will use some logical algorithms to calculate the angle of each mechanical arm joint and send the data into the serial port waiting to communicate with the circuit board.

```
transformVideoData(hand_x, hand_y, hand_z, no_of_fingers_under);
```

Unfortunately, because of limited time, the angle computational algorithm was not implemented perfectly. This project just uses some simple logical judgments. Some details are given in Chapter 5: Testing and Evaluation.

5. Communication mechanism

The final function of the application is to communicate with the circuit board, using a serial connection cable. The key of communication is serial port communication. Here is the information for serial port configuration.

```
bool InitPort(UINT  portNo = 1, UINT  baud = CBR_115200, char  parity = 'N',
              UINT  databits = 8, UINT  stopsbits = 1, DWORD dwCommEvents = EV_RXCHAR);
```

- UINT portNo is Port number, default value is 1 (COM1). But we use COM5 in our project
- UINT baud is Baud rate, default value is 115200
- Char Parity is whether use odd-even check, default value is 'N' which means no odd-even check
- UINT databits is number of data bits, default value is 8
- UINT stopsbits is format of stop bits, default value is 1
- DWORD dwCommEvents, default value is EV_RXCHAR, which will create a new event even if the port receive or send one single character.

In this project, the function sendData() is used to send the angle data into the serial port. finalOrder is an array of int type that contains the mechanical arm joint angle data. mySerialPort is objective COM.

```
sendData(finalOrder, mySerialPort);
```

4.3 STM32 circuit board program Implementation

The main function on the circuit board side is to receive the data from the PC application, and control mechanical arm joint's angle.

The main function of stm32 chip has just one while loop running all the time.

```
while(1){
  if(DMA_GetFlagStatus(DMA1_FLAG_TC5)!=RESET){

      joints_settle_data(dma_buffer);
      joints_control();

      DMA_ClearFlag(DMA1_FLAG_TC5);
  }
}
```

DMA: Direct Memory Access.

- DMA_GetFlagStatus is a check function who to check whether a DMA address was received from the serial port.
- DMA_FLAG_TC_5 means address of COM5. If DMA was not written by COM5 data (no data received), the function will return RESET. Otherwise, the C function, joint_settle_data() will decode data and store it into a global array variable: dma_buffer. Next, the C function joints_control() will control stm32 chip to output PWM signal, according the angles information in dma_buffer. Finally, the DMA_ClearFlag(DMA1_FLAG_TC5) function will delete the COM5 flag and wait for the next data to be sent from PC serial port.

Serial communication is key when linking a PC application with the circuit board part. This project's circuit board uses serial communication interface (USART) to receive or send messages. USART is abbreviation for Universal Synchronous Asynchronous Receiver Transmitter. Here is an introduction about some important default parameters in COM and DMA initialize function.

```
void COM_DMA_Init(void){
   USART_InitTypeDef USART_InitStructure;
   USART_InitStructure.USART_BaudRate = 115200;
   USART_InitStructure.USART_WordLength = USART_WordLength_8b;
   USART_InitStructure.USART_StopBits = USART_StopBits_1;
   USART_InitStructure.USART_Parity = USART_Parity_No;
   USART_InitStructure.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
   USART_InitStructure.USART_Mode = USART_Mode_Rx |USART_Mode_Tx;
   USART_Init(USART1, &USART_InitStructure);
   USART_Cmd(USART1, ENABLE);
}
```

In function COM_DMA_Init(), USART_InitTypeDef is a STM library class which is used to define and initialize the USART parameters. In this project, we set baud rate to 115200, which is the same as the PC application serial port's baud rate. Word length is 8 bits. Stop bits is 1, which is the same as the PC application. USART parity is no which means there is no odd-even check. Hardware Flow Control is none which means no hardware flaw. USART mode type is send(USART_Mode_Tx) and received(USART_Mode_Rx). We set USART1 as our default communication port.

In this project, PWM signal is key to control joint servos. The mechanical arm we use has 6 servos as its joints, so we need use 6 difference PWM signal to control six joints respectively. When stm32 receives new data from the PC, stm32 will decode this data, calculate the angle changing of each joint (Rotating spindle from old angle to new angle). Finally, it will transfer angle number into PWM signal. In this section, we will give a fragment code to describe some main operations.

First, stm32 should decode the data from PC and transfer angle figure in PWM signal figure.

```
void joints_settle_data(uint8_t* dma_buffer){
  int i;
  int cont = 0;
  for(i = 0; i < 36; i += 6){
    if(old_angle[cont] == getangle(dma_buffer, i))
      old_angle[cont] = getangle(dma_buffer, i);

    old_PWM[cont] = 360 + old_angle[cont] * 8;
    new_angle[cont] = getangle(dma_buffer, i + 3);

    new_PWM[cont] = 360 + new_angle[cont] * 8;

    change[cont] = getchangeangle(old_PWM[cont], new_PWM[cont]);
    changerate[cont] = change[cont]/8;
    old_angle[cont] = new_angle[cont];
    cont++;
  }
}
```

The global array dma_buffer is a char array and its length is 36. It contains start angles (old_angle) and end angles (new_angle) of joint's motion information. The C function getangle(dma_buffer, i) is use for transforming char type into int type. Start angle of each joint servo is stored into a global int type array (old_angle[6]) and End angle of each joint is stored into a global int type array: new_angle. The next step is transforming the angle into PWM signal figure using the following reduction formula:

$$PWM = angle * 8 + 360$$

After transformation, Start PWM figures will be store in global int type array old_PWM[6] and End PWM figures will be stored in a global int array new_PWM[6].

There is another operation is that calculating changing rate of PWM. One servo can't rotate its spindle very fast, which could make mechanical arm action rigid and slack. We should control it rotate step by step from its start angle to end angle. This project assume that a set of angle changing will take 1sec, so we divide rotation angle into 8 steps for one changing action. For instance, if data requires joint 6 rotate its angle from 90 degree to 170 degree in one second, joint 6' s servo will change its angle to

100 degree at 125ms, 110 degree at 250ms, 120 degree at 375ms, 130 degree at 500ms....., until the joint 6' s angle become 170 degree at 1000ms (1 sec). This method can make mechanical arm action slow and coherent, which also can protect servos avoid from gear abrasion. As picture shown.

```
change[cont] = getchangeangle(old_PWM[cont], new_PWM[cont]);
changerate[cont] = change[cont]/8;
```

The function calculate the change rate of PWM signal and store it in a global array changerate[6].

Second, it is time to control joint servo, which is the final point of programming. This project have a function called joints_control() as follows:

```
void joints_control(){
   int i;
   for(i = 1; i <= 8; ++i){
     joint1_change_control(old_PWM[0] + changerate[0]);
     joint2_change_control(old_PWM[1] + changerate[1]);
     joint3_change_control(old_PWM[2] + changerate[2]);
     joint4_change_control(old_PWM[3] + changerate[3]);
     joint5_change_control(old_PWM[4] + changerate[4]);
     joint6_change_control(old_PWM[5] + changerate[5]);
     old_PWM[0] += changerate[0];
     old_PWM[1] += changerate[1];
     old_PWM[2] += changerate[2];
     old_PWM[3] += changerate[3];
     old_PWM[4] += changerate[4];
     old_PWM[5] += changerate[5];
     delay_ms(250);
   }
}
```

In this function, it has a for loop to operate PWM signal using *old_PWM*, *new_PWM* and *changerate* global array. This for loop controls six PWM signal to increase or decrease, step by step. After every time joint's angle changes, the for loop will delay by 250 ms to make servo rotation slow down.

There are six joint_change_control functions to drive six joint servos, and all of them have the same mechanism. This report only introduces joint1_change_control(int) function as an example.

In the function joint1_change_control(), the int type parameter will transfer in uint16_t type and the parameter is passed to function joint1_set_PWM(uint16_t).

```
void joint1_change_control(int signal){
  joint1_set_PWM((uint16_t) signal);
}
```

In function joint1_set_PWM(uint16_t), invokes the TIM_setCompare1() function which is contained in the STM32 library. This function can set pin PA0's PWM value, which drives joint 1 servo motor.

```
void joint1_set_PWM(uint16_t signal){
  TIM_SetCompare1(TIM2, signal);
}
```

After this operation, pin PA0's PWM signal is changed.

This concludes the details of the implementation of this project which comprised two USB external cameras, a controlling application, a controlling circuit board and a mechanical arm.

Chapter 5: Testing and Evaluation

5.1 Testing

This project uses Qt Creator test tool: QtTestLib framework as project unit test tool. This project will test four important points in application.

First test point is testing output of function getAngleOfJoint6 in class. Test parameters is 320 and 240, and expected result is 90. If this function return 90, this test will pass.

```
void CTestCase::Test_Joint_6_Calculation()
{
    MainWindow w;
    m_iRtn = w.getAngleOfJoint6(320, 240);
    QVERIFY(90==m_iRtn);
}
```

Second test point is testing output of function getAngleOfJoint3 in class. Test parameters is 150, 150 and 150, and expected result is 90. If this function return 90, this test will pass.

```
void CTestCase::Test_Joint_3_Calculation(){
    MainWindow w;
    m_iRtn = w.getAngleOfJoint3(150, 150, 150);
    QVERIFY(90==m_iRtn);
}
```

Third test point is testing output of function getAngleOfJoint1 in MainWindow class. Test parameters is 0, and expected result is 90. If this function return 90, this test will pass.

```
void CTestCase::Test_Joint_1_Calculation()
{
    MainWindow w;
    m_iRtn = w.getAngleOfJoint1(0);
    QVERIFY(90==m_iRtn);
}
```

Forth test point is testing output of function dist in MainWindow class. Test parameters is Point a(1,1) and Point b(1,2), and expected result is 1. If this function return 1, this test will pass.

```
void CTestCase::Test_dist_Calculation(){
     Point a(1,1);
     Point b(1,2);
     MainWindow w;
     m_iRtn = w.dist(a,b);
     QVERIFY(1==m_iRtn);
}
```

When we run the test, and we get results as picture shown.

```
********* Start testing of CTestCase *********
Config: Using QtTest library 5.5.0, Qt 5.5.0 (i386-little_endian-ilp32 shared (dynamic) debug build; by GCC
4.9.2)
PASS   : CTestCase::initTestCase()
PASS   : CTestCase::Test_Joint_6_Calculation()
PASS   : CTestCase::Test_Joint_3_Calculation()
PASS   : CTestCase::Test_Joint_1_Calculation()
PASS   : CTestCase::Test_dist_Calculation()
Test end
PASS   : CTestCase::cleanupTestCase()
Totals: 6 passed, 0 failed, 0 skipped, 0 blacklisted
********* Finished testing of CTestCase *********
```

It shows 6 test passed, 0 failed, 0 skipped, 0 blacklisted which means all tests passed and all functions is correct.

5.2 Evaluation

Even tough this project is nearly finished, but there are some obvious defects in this project. This section will introduce more details of two obvious defects and how to deal with in further research.

1. The motion planning of mechanical arm was handled easily.
2. Mechanical arm's movement delaying.

1.The motion planning of mechanical arm was handled easily.

Reasons: In this project, the motion planning of mechanical arm is not finished, it was implemented by some simple if - else judgment statements to get angle in each joint, which is very rough. As my planning, there will be a mature and efficient algorithm in my code to calculate mechanical arm' s each joint angle when it get the 3-dimensional coordinate of your hand from camera source. Because time limited, the algorithm of calculate joints angle was implemented by simple if - else statement. For example, in Qt, C++ MainWindow class function: getAngleOfJoint5(int x, int y, int z).

Note: x, y, z is 3D coordinate point parameters of detected hand position. The purpose of this function is that getting the 3D position of detected hand and return the 5th joint's angle.

```
int MainWindow::getAngleOfJoint5(int x, int y, int z){
    if(y >= 320)
        'return 135;
    else if( y < 320 && y >= 160)
        return 90;
    else if(y>=0 && y < 160)
        return 45;
}
```

As picture shown above, this function was implemented in a very simple logical method. Code explaination: if y-coordinate greater than 320, we will assume that the human arm is drawing back his/her hand and this function will return 135 to application, which could let 5th joint' s angle rotate to 135 degree. Mechanical arm will act as drawing hand if the 5th joint's angle is 135 degree.

Further Solutions: There are lots of very effective method, one of popular method is inverse kinematics. (The Closed Form Solution of the Inverse Kinematics of a 6 - DOF Robot) The problem of the inverse kinematics for robot manipulators has always been a challenging problem in their design. Essentially, the problem is to find the vector of the joint angles, say for an naxis revolute manipulator, given the position and orientation of the end- effector or the gripper.

2. Mechanical arm's movement delay

Reasons: As we all know, STM32 chip only can process one task at a time. So when STM32 output the PWM signal to driven mechanical arm' s joints servos, it can't process other tasks at same time, for instance, receiving the data from PC or return a finish order to PC. All the tasks must wait until the running task finish and go to processing next task. It means that if Stm32 is outputting PWM signal to driven joints, it will not receive data from PC at this time. Stm32 will check and read new data after driven process, which could wait for 500ms~1s. However, PC will send data every second, but the driven arm process could take long time. All the task should wait and this situation could cause arm machine mimic movement delay. For instance, in circuit board C code main.cpp, there is a while loop in the main function which will running all the time.

```
while(1){
  if(DMA_GetFlagStatus(DMA1_FLAG_TC5)!=RESET){ //test angle
                                               // 090090090090090090090090090090095

      joints_settle_data(dma_buffer);
      joints_control();

      DMA_ClearFlag(DMA1_FLAG_TC5);
  }
}
```

As picture shows, in if statement, if circuit board DMA communication address is not empty(DMA_GetFlagStatus(DMA1_FLAG_TC5)!=RESET), which means the circuit board received data from PC application succeed and the stm32 chip will decode the data(joint_settle_data(dma_buffer)). After decode the data, angle information will be written in a array(dma_buffer). Function joints_control() will driven six joints of mechanical arm according angle information which was written in dma_buffer array.

Further Solutions: Using µC/OS-II to create multitasking application on circuit board, it allows stm32 process more than one task at same time. In further project, I will try to receive data and control arm machine at the same time.

Chapter 6 Conclusion

6.1 Conclusion

This project has simple implemented the propose purpose. The purpose of application is using two USB camera to catch human arm action. Application processes and analyzes video source which were caught from camera. After processing and analyzing source, the application uses its internal algorithm to detect human hand. In this step, this project uses some key algorithms such as background subtraction, morphological operation, skin color detection, hand structure analysis and so on. Thanks for powerful OpenCV library make it easy to handle and implement. After locating hand's 3D coordinate point, application will use its logical algorithms to calculate joints angle of mechanical arm. Finally, application sends all angle data to serial port to wait for serial communication. The purpose of circuit board side is receiving the data which was sent from UASRT part. According action ordering contained in data, circuit board outputs equivalent PWM signal to drive joints of mechanical arm - mechanical arm could finish designated movement. There are two hard points in this project. First point is circuit board designing. As a software student, it is my first time to touch hardware thing. So I have to start from very beginning hardware knowledge. Fortunately, the company I interned in summer holiday (Qingdao Ugrow) was doing a robotic research project which I could learn a lot hardware knowledge when I was working in this group. After designing and changing draft of circuit board, my new board was born which took me at least one month. Another difficult point is image processing, even though the OpenCV library is powerful and abundant, but it still spend my long time on choosing appropriate algorithms. Even though some algorithms is simple and useful, but most of them can't obtain satisfied effect, so that I have to improve images quality and reduce the noise rate before using those algorithm. In final result of my implementation, my mechanical can mimic human hand to do some simple actions such as make a fist, open hand, raise your hand, lay down your hand, change direction of hand position and so on.

This project is my perfect jumping platform for my further research, even though FYP deadline has come, but this project will never stop at this time. Cause there are some obvious points need to be improved and completed. First point is that motion planning of mechanical arm was handled easily. Because this point, the action of mechanical arm could unnatural and incoherent. So in the further research, improving my planing algorithm is my most essential point. Inverse kinematics will be my first choice of research direction. Another point is mechanical arm's movement delaying. As we all known, STM32 chip only can process one task at one time if there is no special arrangement. So the internal running is receive data, drive arm, receive data, drive

arm....., all the time. This situation can make mechanical arm has long reaction time. In the further research, μC/OS-II will become my first try in my circuit board code, which make multi-tasking become possible. It is simple but not easy.

As my interested field and academic research hot point, smart robot controlling system will become more and more popular in academic world. The application of smart controlling robot will also broad, which can be used in mechanical system, medical system, factory, families and so on. Research of smart controlling robots will never stop and I will continue to find and study in my future study and work life.

References

Park,F.C., and Lynch, K.M. (2015) *INTRODUCTION TO ROBOTICS MECHANICS, PLANNING, AND CONTROL* [online], Northwestern, available: http://hades.mech.northwestern.edu/images/2/2a/Park-lynch.pdf [accessed 17 Apr 2018].

Breazeal, C. and Scassellati, B. (2015) *Challenges in Building Robots That Imitate People* [online], robotic.media, available: http://robotic.media.mit.edu/wp-content/uploads/sites/14/2015/01/MITPress-Imitation.pdf [accessed 17 Apr 2018].

Wikipedia (2018) *Servo control* [wiki], available: https://en.wikipedia.org/wiki/Servo_control [accessed 17 Apr 2018].

Wikipedia (2018) *Mechanical arm* [wiki], available: https://en.wikipedia.org/wiki/Mechanical_arm [accessed 17 Apr 2018].

Laganiere, R. (2001) *OpenCV 2 Computer Vision Application Programming Cookbook* [online], Semanticscholar, available: https://pdfs.semanticscholar.org/1a0c/db6c753abd6e219ebb9b632dd308dc1f5f83.pdf [accessed 17 Apr 2018].

Bradski, G. and Kaehler, A. (2008) *Learning OpenCV* [online], Cuny.edu, available: http://www-cs.ccny.cuny.edu/~wolberg/capstone/opencv/LearningOpenCV.pdf [accessed 17 Apr 2018].

Srinivasan,L.(2013) 'Hand Tracking And Gesture Detection (OpenCV)', s-ln.in Blog [online], 14 Apr, available: https://s-ln.in/2013/04/18/hand-tracking-and-gesture-detection-opencv/ [accessed 17 Apr 2018].

STMicroelectronics (2013) *Medium-density performance line ARM-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. Interfaces* [online], available: http://www.st.com/content/st_com/en/products/microcontrollers/stm32-32-bit-arm-cortex-mcus/stm32-mainstream-mcus/stm32f1-series/stm32f103/stm32f103rb.html [accessed 17 Apr 2018].

Amazon Picture (2018) [online], available: https://www.amazon.com/LewanSoul-LD-27MG-Standard-Digital-Aluminium/dp/B07569WJ1M[accessed 17 Apr 2018].