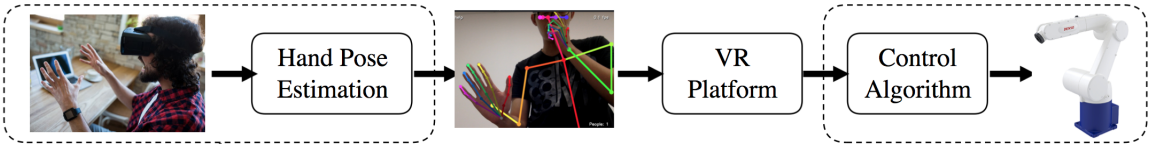
UM-SJTU Joint Institute

VE 450 Major Design Experience

Team 10 VR-Enabled Gesture Control of Robotic arm I

Design Review #3



Sponsor:
Professor Mian Li
Team Members :
Jingying Wang 516370910029 wjymonica@sjtu.edu.cn
Wentao Yang 516021910173 1185729505@sjtu.edu.cn
Zhiyuan Xiang 516370910126 xzy242215@sjtu.edu.cn
Minhao Jin 516370910116 jinminhao@sjtu.edu.cn
Niall Halloran 717020990012 717020990012@sjtu.edu.cn

Contents

1 Abstract

Today - manufacturing around the world is seeing a revolution in applying the newest technology and science to improve their output. To realize that need and conform with the growing trend of Industry 4.0; for this project we built a system which is loosely described as a Virtual Reality(VR) Gesture Controlled Robotic arm. In this system users would make gestures, moving their arms, grasp things,etc. These movements of the user would then be detected by RGB cameras where the movements would be translated in real time to the robotic arm thus following and imitating the user. Meanwhile the user will be able to visualize this movement via a virtual reality space that mirrors the real robot scene.

The project is made up of four parts in general. Hand and arm pose estimation at the client end, object detection and VR platform on the server, both socket based communication and robotic arm control at the factory end.

For each part, we have the relevant solutions. For the hand and arm pose estimation, we choose Openpose by CMU. For object detection, we use OpenCV. For socket based communication, we prefer TCP protocal. For robotic arm, we use HIWONDER robortic arm controlled by Inverse Kinematic(IK) on Raspberry Pi.

2 Problem Description & Introduction

The concept of Industry 4.0 was first coined by the German government in 2011 to promote the Digital Revolution in manufacturing [2]. One of the design goals of Industry 4.0 is to give better connection between people and machines [1]. Advanced human-machine interfaces, augmented reality devices and the use of trigger "real-time" training are all important components of Industry 4.0. The "smart factory" fostered by this concept involves the virtual copy of the physical world and real-time communication with humans [1]. With these advanced technologies – manufacturing of highly integrated products are on demand.

In general, our project aims to give a solution to remote control of machines in factories. To be more specific, our goal is to use different gestures to control the activities of the industrial robot to deliver designed functions, and the whole control process is visualized and demonstrated on a VR platform to help the online monitor. Therefore, this project can be divided into 3 stages.

The whole project is of a cascaded structure. The original data, hand gestures, go through the modules one by one and eventually reach the robotic arm in the factory. The robotic arm is expected to perform some limited gestures given by the hand, including moving, grasping, etc.

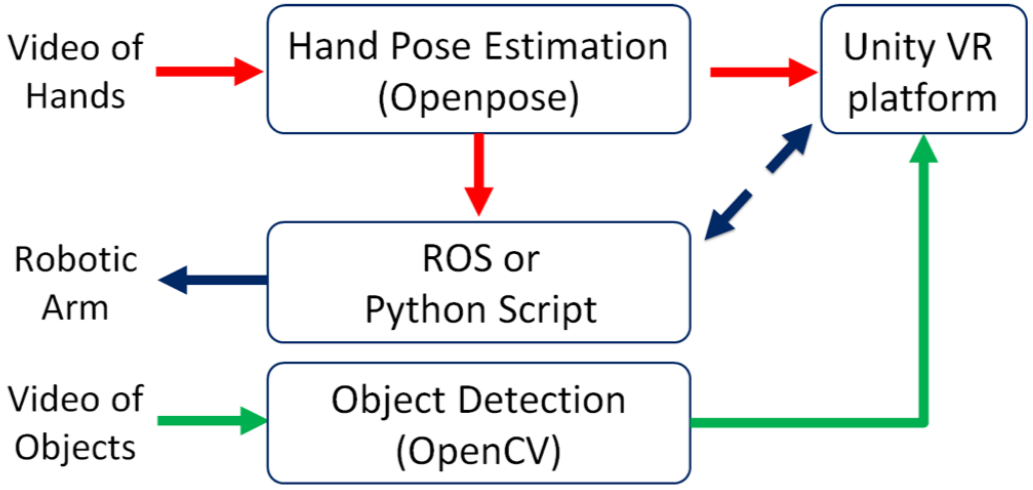


Figure 1: Overview of VR-Enabled Gesture Control of Robotic arm.

2.1 Client End

The first stage is on the client end. A client can only use the camera on his or her laptop and do some gestures in front of it. The pose of the client's hand is captured by the monocular RGB camera. Then each captured frame is sent to a real-time hand pose estimation model. The output of this model positions the key points of the client's hand.

The hand pose estimation model is expected to be implemented by OpenPose [8][9][10] by Carnegie Mellon University, which is a robust but huge model for 2D face, body, and hand skeleton detection.

2.2 Server

The second stage is implemented on a server. As the hand pose estimation model is expected to be huge and complex, the computations will be run remotely. The VR platform will demonstrate the expected movements of the robotic arm and will also run on the same server. In the VR environment, the model of the robot arm will be that of the one in the factory. The virtual model will then have the same degrees of freedom as the real one. The solved signal of the client's hand gesture is sent to VR platform and the robotic arms at each frame. The position and the turning angle of each free joints of the robotic arm model is set according to the hand's spatial information.

As the robot's arm is mechanically different to that of a human hand, the spatial information of the hands will be further processed to fit into the robotic arm model.

Since the hand pose estimation model and the VR platform is on the same server, we should link the two parts by setting the API between them.

2.3 Factory End

The final stage is the factory end, where the real robotic arm is located. All the movements of the real arm is synchronized with the virtual model.

The discrepancy between the virtual arm and the real one is expected to be at millimeter scale. To make the performance of the real one more stable, we apply some control algorithms like PID to make adjustment. Since the factory can be far away from the server, we will have to use wireless communication to transfer data from the server to the real arm.

3 Customer Requirements and Engineering Specifications

3.1 Customer Requirements

Since we are going to provide customers with solutions to remote control of machines in the factory, our ultimate goal should be letting customers operate the remote machines in front of a laptop or an ordinary computer, as well as preview the whole process via a VR platform. We want to create a space as if they are in the factories, pushing the real button, grasping real things. The customer requirements can be specified as follows.

Real-time: To achieve more accurate control of the machines, the robotic arm should act in the factory right after the instructions are given by customers through their hand gestures.

Hand pose estimation: The robotic arm is not required to mimic every movement of hands. Only some basic gestures should be included like moving hands, grasping things, pushing the bottom, and etc.

VR model and VR environment: Since the VR platform serves as the visual demonstration and preview of what will happen in the factory, the VR environment should be set exactly the same as the real factory and the VR model of the robotic arm should be the exact copy of the real one. In this way, customers can adjust their gestures according to the VR demonstration, and know whether their operation is apt in the factory.

Synchronization with hand pose: The motion of the free joints of the virtual robotic

arm model should be set according to the derived spatial information of the hands and the fingers of the customer. When a given gesture like grasping is detected, the virtual model should do grasping in the VR environment. The delay of this synchronization should be short.

Remote control: Since the machines might be at thousands of miles away from the customer, the instructions should be transferred through some wireless communication technology.

Synchronization with VR model: The real robotic arm should perform exactly the same as its virtual model on the VR platform does. The delay of this synchronization should be short.

3.2 Engineering Specifications

After analyzing customer requirement, we identified several engineering specifications as shown in the table below.

1	Accuracy of pose estimation in 62mPA
2	Speed of pose estimation in 20FPS
3	Accuracy of robotic arm in millimeter-scale discrepancy
4	Delay of communication between VR and hand pose estimation within 5ms
5	Delay of communication between VR and robotic arm within 5ms

Table 1: Specific Engineering Requirements

In order to build a real-time system, we want the delay of communication between VR platform and robotic arm and that between hand pose estimation system and VR platform to be as short as possible, and basing on current 4G wireless network speed, the delay can be controlled within 5ms. Also the hand pose estimation should be accurate and fast, according to current research in hand pose estimation field, an accuracy in 62mPA and images processing rate of 20 FPS would be reasonable.

For the hand pose estimation, since only some basic gestures should be included, we want it to be efficient, so its accuracy should be quite high and it should be able to process images fast.

Then for the VR model and environment, since this part is mainly handled by Group11, the only thing we can do is to perform an accurate and fast hand pose estimation. Reducing the delay of communication can also be helpful.

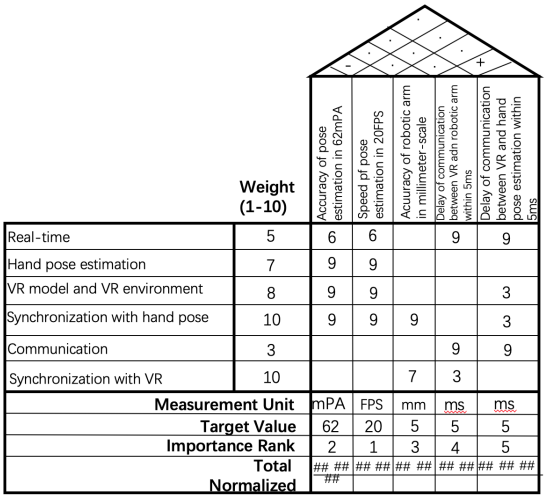
And to synchronize the arm in VR platform with hand pose, the hand pose estimation should send continuous and accurate data to VR platform within a short delay,

so that some precise and consistent movements can be displayed in VR platform. Thus, the accuracy and rate of hand pose estimation is going to have a huge impact here, and short delay of communication is also needed.

Similarly, to synchronize the robotic arm with the arm in VR platform, we should control the robotic arm with high accuracy, basing on the size of the robotic arm, the deviation should be controlled in millimeter-scale. Also, the delay of communication should be as short as possible, within 5ms should be enough.

Finally for remote control, in order to provide users with a good experience for the remote control, we want the communication to be fast and stable, thus the delay of the communication between different parts should be short enough, the same as the above, we believe within 5ms is enough.

3.3 Quality Function Development(QFD)



	Weight (1-10)	Accuracy of pose estimation in 62mPA	Speed pf pose estimation in 20FPS	Acuuracy of robotic arm in millimeter-scale	Delay of communication between VR adn robotic arm within 5ms	Delay of communication between VR and hand pose estimation within 5ms
Real-time	5	6	6		9	9
Hand pose estimation	7	9	9			
VR model and VR environment	8	9	9			3
Synchronization with hand pose	10	9	9	9		3
Communication	3				9	9
Synchronization with VR	10			7	3	
Measurement Unit		mPA	FPS	mm	ms	ms
Target Value		62	20	5	5	5
Importance Rank		2	1	3	4	5
Total		## ## ##	## ##	## ##	## ##	## ## ##
Normalized						

Figure 2: QFD chart

In order to meet all the customer requirements, we have several engineering specifications. To help us have a better understanding of the priority order of these engineering specifications, and the relationship between them, we made a QFD chart.

In the chart, we list all the customer requirements together with all the engineering specifications we have identified. Each row represents one customer requirement, and each column represent one engineering specification. Each engineering specification are weighted - 9 being the highest weight and most important. For example, the hand pose estimation's accuracy should be quite high and it should be able to process images fast, so for this requirement, we give the highest weight to accuracy

of hand pose estimation and speed of hand pose estimation these two engineering specification, all the other specifications are not important for this requirement.

Also, we also set different weights to different customer requirements to reflect the attitude of customers towards different requirements. From 0 to 10, where 10 means the customers strongly emphasize this requirement, while 0 means the customers don't care about it. So in our project, the synchronization between hand pose estimation system, VR platform and robotic arm is the most important requirement according to our sponsor, while the real time requirement and remote control are relatively unimportant.

Thus, considering the weights of different customer requirements and the weights of different engineering specifications for different requirements, we get the importance rank for different specifications. We find the accuracy and speed of hand pose estimation are the most important specifications, then follows the accuracy of robotic arm, while the delays of communication are relatively unimportant.

By analyzing the relationship between different specifications, we find there is a negative relationship between accuracy and speed of hand pose estimation. High accuracy means more computational power is needed in hand pose estimation, which will slow its speed. Since we are only required to recognize some basic gestures in hand pose estimation, we give the speed of hand pose estimation a higher priority.

This QFD chart not only helps us to clarify the relationship between customer requirements and engineering specifications, but also help us to figure out the work we need to do and their priorities. With this QFD chart, we can make a reasonable and clear future plan for the whole project.

4 Concept Generation

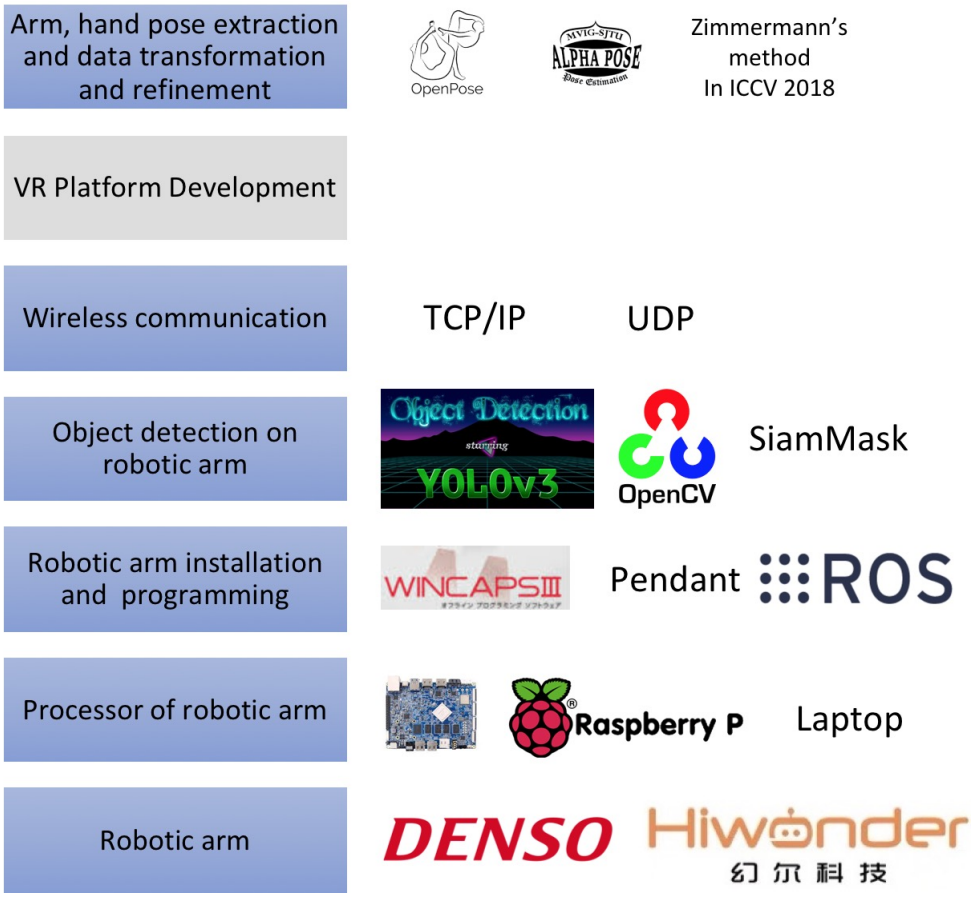


Figure 3: Morphological chart.

4.1 Hand & arm pose estimation

For hand pose estimation part, we want its accuracy to be at least 62 mPA, and the processing speed should be larger than 20 FPS.

To make the accuracy of hand pose estimation as high as possible, we consulted several papers to find a proper theoretical method. But we found that there are too few papers on hand pose estimation, so we decided to expand our search to pose estimation, because the pose estimation, especially the arm detection can help us to indirectly confirm the position of the hand.

In this case, the original hand pose estimation is changed to hand arm pose estimation. We found at present, the most suitable and powerful theoretical method for

pose estimation is called convolutional pose machine, which will use multiple stages of convolutional network layers to process the images.

We then narrowed down our search to open source systems which support pose estimation and are based on convolutional pose machine, as they are more likely to meet our specification about accuracy. Totally there are three open source systems that support pose estimation: OpenPose, AlphaPose and Zimmermann's method in ICCV.

For Openpose, it can complete pose estimation and hand pose estimation in 2D at the same time, and for AlphaPose, it is only able to complete 2D pose estimation, while Zimmermann's method can only detect hand pose in 3D.

Considering that arm detection can assist hand pose estimation to improve its accuracy, we have two options in this part to choose from: either we only use OpenPose or we combine AlphaPose and Zimmermann's method in ICCV together.

4.2 Object Detection

When users are controlling the robotic arm, two important issues that they need to know are where the target objects are located and how far it is from the robotic arm. However, when he or she is controlling the robotic arm remotely and cannot see the scene at the robotic arm end, then a VR platform simulating the real scene is need. Having the positions of objects known is a must.

The question is simple - how do we locate objects in the scene? Intuitively, the way humans locate objects is to first capturing the image containing objects, then analyze which corner they are in, and then track the objects with their eyes. One strategy that imitates this process is to use cameras to do localization. Since the location of one dimension is already known to be on the plane or at the same height of the end of robotic arm, we only need to derive the 2D position. In this sense, this problem can be solved using object detection in computer vision using a single RGB camera.

In the computer vision field, there are a lot of useful machine learning models to address this problem. We usually come up with some popular and classic methods like You only look once (YOLO), and OpenCV. The most recently published state-of-art methods like SiamMask are also worth paying attention to.

Most models for object detection can be divided into two categories – detectors and trackers. Primarily, detectors recognize and define the bounding boxes of the objects in the images. Trackers on the other hand tracks the moving trajectory of the objects, usually with the initial bounding boxes given. For real-time object

detection, we can either use detector to process every frame or use trackers to process the video.

Therefore, the concept we generated in this part are YOLO, a popular and state-of-art detector, OpenCV, an old fashion tracker, SiamMask, a state-of-art tracker published in 2019.

4.3 Socket Based Communication

Using socket based communication is a common measure in communication between server and several processors. Socket based communication includes TCP/IP and UDP. These two protocols are the most commonly protocols in the communication part and almost all the applications are using these two communication measures. Therefore, there is no denying that we will also take these two communication measures into account.

Both TCP and UDP are protocols. They are used for sending bits of data.

TCP/IP needs connection. It is a connection-oriented protocol and it can provide stable connection. It has hand shake protocol like SYN, SYN-ACK and ACK to make sure the receiver has already received the data transmitted from the transmitter.

Meanwhile UDP doesn't need a connection. It is connection-less, so it is much faster than TCP. However, it is not that stable as it's not responsible for checking if the receiver has already received the data it transmits. In other words, it has no hand shake protocol.

4.4 Robotic Arm

When considering the robotic arm, we need to take three aspects into account, which are robotic arm related processor, robotic arm supported kinematics and robotic arm itself.

4.4.1 Robotic related processor

There are three kinds of processor that are commonly used. They are the laptop, RK3399 and Raspberry Pi. The architecture of the laptop is AMD64, which is different with that of arm64, belonging to RK3399 and Raspberry Pi. This difference can show on the software when we control the robotic arm. For example, Robot Operating System (ROS) is a powerful OS that can control various kinds of robotic arm, however RK3399 can only support the newest version of ROS . The newest version isn't compatible with most kinds of robotic arms. Meanwhile Raspberry PI doesn't even support ROS. It is significant to choose the robotic related processor

since it will determine if we can use some specific powerful OS to control the robotic arm.

What's more, since object detection is run on the processor, the use of processor will also determine the functionality and performance we have on object detection. Nowadays, there exists a lot of object detection algorithms. Generally speaking, the better performance that algorithm provides, the larger computation it needs. Therefore, using a laptop will always be better than using arm64 based processors.

However, the most important criteria on choosing the processors are compatibility. If a processor is very compatible with that robotic arm, then it will be easier in implementing project related functionality. Less bugs will save a lot of time and we can make use of it to refine and expand the functionality of robotic arm.

4.4.2 Robotic arm supported kinematics

For the robotic arm supported kinematics, there are two solutions we can choose. One is Inverse Kinematics (IK) and another is Geometric angle calculation. IK is a very convenient kinematics that given a destination location, the algorithm will determine the path going from the current location to the destination automatically. In contrast, Geometric angle calculation will be more straightforward. It will just rotate each ankle to the target angle. This method is harder to implement, but its processing speed will be far faster than IK.

4.4.3 Robotic arm itself

Choosing the robotic arm is also an important issue. There are two kinds of robotic arm which belongs to industry level and laboratory level. Industry level robotic arm is more stable, but the problem is that it is far more complicated than the experimental level robotic arm because it has thousands of protection warning and will automatically turns off when meeting any issue. What's more, recovering the robotic back to normal state is not an easy issue since it is very complicated and have many protection operations. In contrast, using laboratory level robotic arm will be easier, however the stability and performance will not be that good as the industry level robotic arm.

5 Concept Selection Process

5.1 Hand & arm pose estimation

In this part we evaluate the two possible solution for hand arm pose estimation: one is using OpenPose to complete both arm detection and hand pose detection,

another one is combining AlphaPose and Zimmermann's method in ICCV together to detect 3D hand pose and use 2D body pose to assist it.

According to engineering specifications, we have both processing speed and estimation accuracy requirement for this par. Our hand pose estimation and VR system needs to run on the same server, so for the overall performance, the computational power required in hand pose estimation should also be considered. At the same time, in order to facilitate the VR system model to reflect the hand movement more accurately, it would be better if we can get a 3D construction.

Considering the above requirements and practical conditions, we will evaluate these two options for four aspects: speed, computation, 3D construction and accuracy.

5.1.1 Speed

For the first option, by testing OpenPose on the server, we found its processing speed could be kept stable at around 20 FPS, while for the second options, since it need to run AlphaPose and Zimmermann's method together, when we test it on the same server, the total processing speed could only get about 15 FPS. In this case, OpenPose has a better performance in speed aspect.

5.1.2 Computation

For OpenPose, when we run it on the server, the CPU and GPU utility were both lower than 50%, but for second options, since we need to run two systems at the same time to finish the overall pose estimation, the CPU and GPU utility are higher than that of OpenPose, both are over 50% and close to 70%. In this case, OpenPose need less computational power than the second method.

5.1.3 3D construction

Since OpenPose only support 2D hand pose estimation, and one server can only support one process to run OpenPose, because OpenPose will require an exclusive GPU to run, and one server can only have one GPU. In this case, we cannot build the 3D construction basing on only one 2D hand pose estimation. But for the second method, since Zimmermann's method in ICCV can detect 3D hand pose, it is possible to make a proper 3D construction.

5.1.4 Accuracy

For OpenPose, basing on its official document, it can get 62 mPA accuracy with a single 1080Ti GPU on COCO test set, of course its accuracy meets our requirement. While for second option, Alphapose can get about 65 mPA on COCO test set with

a single 1080Ti GPU, it also meets our requirement. In accuracy part, both two methods meet our requirement, and the second method may perform slightly better in this aspect.

After analyzing, we give the first method, OpenPose, 4 marks for speed, 2 marks for computation, 4 marks for accuracy but 0 for 3D construction. After multiplying with the corresponding criteria rating, its final score is 2.4. While for second method, we give it 3 marks for speed, 1 mark for computation, 1 mark for 3D construction and 4 marks for Accuracy. Its final score is 2.3.

Since OpenPose has higher final score, we decided to choose OpenPose to solve the hand arm pose estimation.

Pose Estimation	Speed	Computation	3D Contruction	Accuracy	Total
Units	**Fps**	**Exp**	**Bool**	**mPA**	
Criteria rating	0.2	0.2	0.3	0.3	
OpenPose	20	90	0	62	
Score	4	2	0	4	
Weighted rating	0.8	0.4	0	1.2	2.4
AlphaPose + Zimmermann's method	15	100	1	65	
Score	3	1	1	4	
Weighted rating	0.6	0.2	0.3	1.2	2.3

Figure 4: Selection Matrix on hand and arm pose estimation.

5.2 Object Detection

We evaluate the performance of YOLO, OpenCV, and SiamMask in 5 dimensions which are speed, weight, limitation in object type, tracking function, and accuracy.

5.2.1 Speed

Since the objective is real-time detection, speed of models is a vital factor. Speed here is evaluated according to frames per second (FPS) processed. Usually, to cheat

human's brain that the video is fluent, FPS should be at least 10, and in VR environment, at least 30 FPS is needed to ensure fluent rendering. Therefore, models with larger value of FPS are preferred. [Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video] According to Mohammad et al. [14], Fast YOLO can "run an average of 18FPS on a Nvidia Jetson TX1 embedded system." This speed is not satisfying. According to Wang et al. [13], SiamMask "produces class-agnostic object segmentation masks and rotated bounding boxes at 55 frames per second." However, we tested SiamMask on CPU and the speed was estimated to be around 1 FPS, let alone implementing it on processors like Raspberry Pi or RK3399, which is far from our expectation and requirement. We tested OpenCV method on both CPU and RK3399 and the speed is 30 50 FPS, which meets our demand.

5.2.2 Weight

Since the object detection models will be implement on processors, the size and the complexity are both of great concern. YOLO and OpenCV are both famous for their light weight. YOLO uses "a single neural network to the full image." The project files for OpenCV models are all typically small. With only packages, cv and dlib, installed, OpenCV models can be run on Raspberry Pi and RK3399. The project file of SiamMask is a little bit larger and requires package like pytorch installed, but some processors like RK3399 do not support such complex packages. It is worth mentioning here that we should trade-off between the detection speed and the complexity. Usually, the more complex the model is and the more delicate the design is, the faster the speed is.

5.2.3 Limitation of Detection

(Almost) all detectors are limited in the object types they can detect, since those detectors are all trained by various datasets. The detector can only recognize the region with objects that it knows. Trackers need manually selected initial bounding boxes, so all kinds of objects we want can be selected. In this sense, OpenCV and SiamMask have no limitation in object type. The detection scope of YOLO is restricted to 20 kinds and excludes industrial tools.

5.2.4 Tracking function

Since trackers predict the next position of objects and derives more motion information, so they are naturally faster. Tracker preserves identity, while detectors locate the objects of each frame independently. When multiples objects are crowded in one scene, detectors may fail to distinguish the trajectory of each one. In this sense,

OpenCV and SiamMask are naturally preferred.

5.2.5 Accuracy

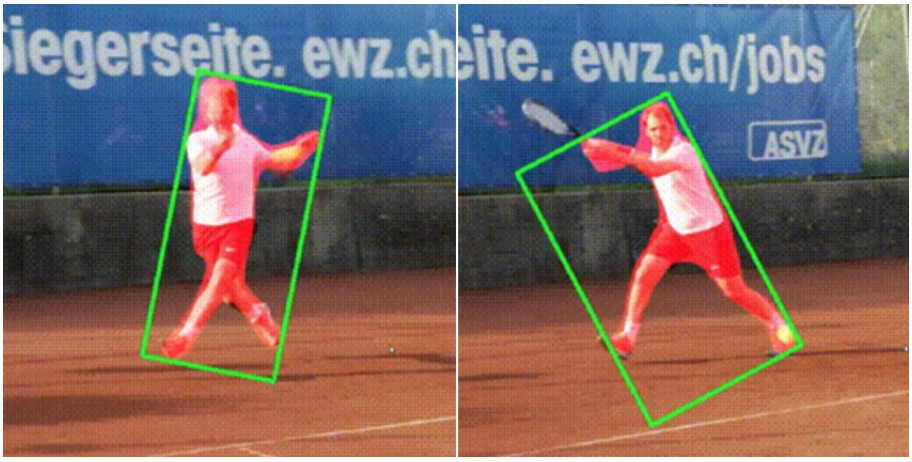


Figure 5: Demo of SiamMask.

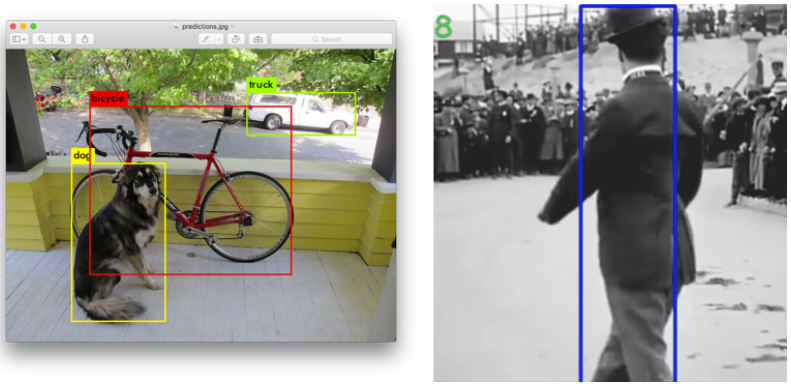


Figure 6: Demo of YOLO and OpenCV.

Accuracy for detectors refers to whether the bounding boxes are strictly bounding the outline of the moving objects. Accuracy is usually measured by the value of mean average performance (mAP). Some methods have not mentioned this value, so we cannot give an overall comparison quantitively. The mAP of YOLO is 57.9% tested on COCO dataset, that of SiamMask is 60.47% on VOT-2016 dataset, and that of OpenCV is not measured. Visually, the bounding box of OpenCV is a little bit looser than that of the other two, which will lead to inaccurate estimation of the

centers of objects. SiamMask, as the newest method, also is very robust in detecting the rotation of objects.

Object Detection & Tracking	Speed	Lightweight	Limitation of Detection	Tracking	Accuracy	Total
Units	**Fps**	**Exp**	**Exp**	**Bool**	**Exp**	
Criteria rating	0.3	0.2	0.1	0.1	0.3	
YOLO	18	60	50	0	60	
Score	3	3	3	0	3	
Weighted rating	0.9	0.6	0.3	0	0.9	2.7
Naïve OpenCV Method	30	90	100	1	40	
Score	4	4	5	1	2	
Weighted rating	1.2	0.8	0.5	0.1	0.6	3.2
SiamMask	1	50	100	1	80	
Score	1	3	5	1	4	
Weighted rating	0.3	0.6	0.5	0.1	1.2	2.7

Figure 7: Selection Matrix on object detection.

We evaluate the three models on these 5 dimensions. Speed and accuracy to are the two most important features, and the weight is also vital. OpenCV gains the highest overall score, and is finally selected to be used in object detection.

5.3 Socket Based Communication

In this field, two commonly used communication protocols are considered, that are TCP and UDP. Theoretically, TCP is used for some conditions that need stable connection and doesn't require a fast and efficient transmission rate. UDP is used for some conditions that requires fast and efficient transmission rate but transmission failure is not that important.

There are several important criteria that should be considered.

5.3.1 Communication quality

It is important to guarantee the quality of communication since the information of each frame will be very important and some lost of information will directly influence the fluency and accuracy of the behavior of robotic arm. For the quality, based on the difference of mechanism on these two protocols, the connection-oriented protocol, TCP, will have a better connection quality. Although the hand shake protocol will make the speed to establish connection lower, because our project doesn't care so much on that properties and we can spend some time waiting for the connection establishment, it will not be a major issue.

5.3.2 Communication time

The communication time is also super significant. If it is too long, then the information transmitted to the VR and robotic arm will be far behind the current hand pose and arm state and the information will be out dated. Then our real time assumption will be broken. Due to the speed of electromagnetic wave [U+FF0C] the transmission speed will not have a big difference between TCP and UDP. The speed will only be dependent on the distance between the transmitter and receiver. Therefore, it will not have a large effect to affect our decision on using TCP or UDP.

5.3.3 Ability for broadcasting

The ability for broadcasting is also important. If it is hard to implement broadcasting on a typical communication protocol, then it is hard to let server communicate with VR and robotic arm at the same time. Either VR or robotic arm will receive the out dated data and this will affect synchronization between the robotic arm and VR system. Therefore, the broadcasting is very important. If we are using UDP, we can try to build a system like a chatting room. Then all the transmitters and receivers can both transmit and receive data. They are all called terminal. Since any terminal can transmit message to other terminal, we can build a system like a chatting room, then our project need will be satisfied. If we are using TCP, we can use multi-thread system. One thread is charged for construct the data that waits to be transmitted. Other threads will be responsible to read the data and then transmit it to their target receiver. Because each thread has different targeted receiver and all the threads can run almost in the same time, therefore, this kind of system can also meet our project's needs.

	Communicatio quality	Communication time	Broadcasting ability	Total
Units	**exp**	**exp**	**exp**	
Criteria rating	0.2	0.3	0.5	
TCP	5	4	1	
Score	5	4	1	
Weighted rating	0.3	1.2	0.5	2
UDP	3	4	1	
Score	3	4	1	
Weighted rating	0.6	1.2	0.5	2.3

Figure 8: Selection matrix on choosing communication protocol

We can see that both UDP and TCP have a good communication time and the ability of broadcasting is also wonderful. Since the connection quality for TCP is better than that for UDP, we decide to use TCP as communication protocol.

5.4 Robotic Arm

In the field of robotic arms, the main three aspects that we are considering are robotic arm related processor, robotic arm supported kinematics and robotic arm itself. There are several criteria that we need to take into account.

5.4.1 Robotic arm related processor

Compatibility for robotic arm related processor

For the robotic arm related processor, one of the important criteria is the compatibility. When using some industry level robotic arm, it is a good choice to use a laptop for it's ROS compatibility. What we need to do is add a robotic arm description file into the workspace of ROS. This description file is very easy to get for the industry level robotic arm, and for the laboratory level robotic arm, most of the times we should create this file by ourselves which is very inconvenient. If we are using laboratory level robotic arm, it is better to use ARM64 based processor since many laboratory level robotic arm supports these kinds of processor very well.

Computation and application for robotic arm related processor

Another important criteria on choosing the processor is its computation and application. The laptop is the obvious choice since it has the larger computation and AMD64 supports various kinds of application. Compared with two different arm64 based processor RK3399 and Raspberry Pi - the computation of RK3399 is better than Raspberry PI. However, although both RK3399 and Raspberry Pi run Linux as its operating system, Raspbian used by Raspberry Pi is far more convenient and has more applications than the Ubuntu modified with Debian used by RK3399.

Controller	CPU Capacity	Support ROS	Funtionality	Compatibility	Total
Units	**GHz**	**exp**	**exp**	**number**	
Criteria rating	0.4	0.1	0.2	0.3	
Raspberry Pi	1.5	0	5	6	
Score	3	0	5	6	
Weighted rating	1.2	0	1	1.8	4
RK3399	1.7	0	2	2	
Score	4	0	2	4	
Weighted rating	1.6	0	0.4	1.2	3.2
Laptop	3.2	1	5	3	
Score	5	1	5	3	
Weighted rating	2	0.1	1	0.9	4

Figure 9: Selection matrix on choosing robotic arm related processor

From the figure9, we can see that both laptop and Raspberry Pi work well. Thus, if we are using a laboratory level robotic arm, then it is fine to use Raspberry Pi for the processor. If we are using industry level robotic arm, it's best to use the laptop.

5.4.2 Robotic arm supported kinematics

Possible choice for robotic arm supported kinematics If we are using ROS, the we must use IK since ROS just supports an input as the destination location and it will path the way automatically. The DENSO robot, a kind of industry level robotic arm, must turn on it's b-CAP slave signal, unless ROS will be unable to control it. For laboratory level robotic arm, since almost all the low level code are open to the user, both IK and geometric angle calculation will be fine in being implemented and used.

In summary, if we are using industry level robotic arm, it is highly possible that we can just use IK. If we are using laboratory level robotic arm, both of them can be used.

Convenience and accuracy for robotic arm supported kinematics

Generally speaking, IK will be far more convenient be used than geometric angle calculation since an input representing the destination location is enough. It is not necessary to calculate any angle for each ankle to rotate and it is highly user friendly. However, IK can not always help robotic arm reach the target area in very accurate way. If we really know the angle each part needs to rotate, then IK will not be that accurate compared to using geometric angle calculation.

	Accuracy	Convenience	Runtime complexity	Total
Unit	**Exp**	**Exp**	**Exp**	
Criteria rating	0.5	0.3	0.2	
Inverse kinemtics	4	5	5	
Score	4	5	5	
Weighted rating	2	1.5	1	4.5
Geometric angle calclation	5	2	4	
Score	5	2	4	
Weighted rating	2.5	0.6	0.8	3.9

Figure 10: Selection matrix on choosing robotic arm supported kinematics

From the figure10, we can see that the accuracy and run time complexity for Inverse kinematics and geometric angle calculation don't have a large difference. And because that using inverse kinematics is more convenient than using geometric angle calculation, we finally choose inverse kinematics.

5.4.3 Robotic arm itself

Convenience and accuracy for robotic arm itself

We take the laboratory level robotic arm, HIWONDER, and industry level robotic arm, DENSO vm6083, into account. For convenience, using HIWONDER will be easier since it has good guides online and all the low layer code is open to us. So we can change whatever we want and many functions are easy to be implemented. In contrast, DENSO is far more complicated to use. DENSO robotic arm has thousands of errors and many errors are just to protect it self. It is very time consuming when meeting these kinds of issues and since its upport is not that good, fixing these kinds of problems is not that easy.

For the accuracy, since the motor DENSO uses are far better than what HIWONDER uses, when the robotic arm is operating, there doesn't exist redundant vibration for DENSO robotic arm, while HIWONDER robobotic arm will have some vibration. What's more, DENSO has a separate specific controller controlling the robotic arm while HIWONDER just makes use of Raspberry Pi. The controller ability for DENSO is also better than that for HIWONDER.

	Stability	Compatibility	Covenience	Total
Units	**exp**	**exp**	**exp**	
Criteria rating	0.4	0.3	0.3	
DNESO vm6083	5	4	1	
Score	5	4	1	
Weighted rating	2	1.2	0.3	3.5
HIWONDER	3	4	5	
Score	3	4	5	
Weighted rating	1.2	1.2	1.5	3.9

Figure 11: Selection matrix on choosing robotic arm iteself

From the figure12, we can see that although the accuracy, stability for DENSO robotic arm is much higher than the Raspberry Pi, it is very inconvenient in use. This will take a lot of time addressing some not meaningful things. Therefore, finally, we chose to use HIWONDER since it is easy to use and its accuracy is not that bad.

6 Selected Concept Description

This project is consisted of sub-tasks including hand pose estimation to solve the motion of hands, VR platform development for visual demonstration of control, Wireless communication to send data, object detection to locate objects being operated, and robotic arm control with Inverse Kinematics (IK) implemented by Python Programming.

First of all, users' gestures are solved by OpenPose in hand pose estimation part. The input of this module is the real-time video. We defined 4 hand states (from closing to opening), and moving directions in three axis in the 3D space (which are left-right, down-up, backward-forward) as ouput.

Second, we send the output of hand pose estimation to the VR platform and the robotic arms using Socket Based Communication.

Third, the robotic arms are controlled by python scripts based on Inverse kinematics. The input of this module is the instructions sent by hand pose estimation module.

Finally, the positions of objects in the real scene are solved by OpenCV model. The centers of objects in the defined region are sent back to VR using Socket Based Communication.

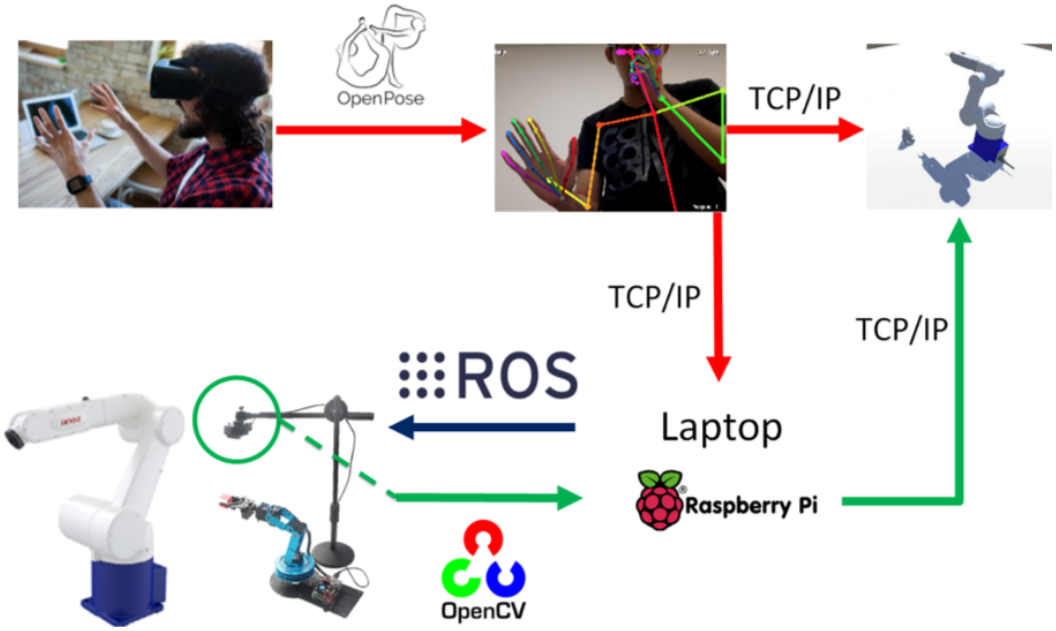


Figure 12: Concept diagram.

6.1 Engineering Design Analysis

6.1.1 Hand & arm pose estimation

OpenPose is a open source system for pose estimation, it can finish hand detection, body detection and face detection at the same time. It also support multi-person pose estimation. Because it is a very complete open source system, there are very few parameters that need to be set by us. But it allows us to choose which kind of function to activate.

For our project, our target is to accurately detect the hand pose and control the robotic arm accordingly. In this case, the face detection is unnecessary, and activating face detection will slow down the entire system. Also, we have only one robotic arm to control, in order to prevent multiple persons controlling one robotic arm at the same time, multi-person pose estimation needs to be prohibited. On the server, when we activated the face detection, the processing speed of OpenPose went down

from 25-35 FPS to 15-25 FPS. And the body pose estimation is necessary as mentioned in Concept Generation part, it can assist hand pose estimation and improve its performance.

Thus, for OpenPose, we need to activate its body pose estimation part together with hand pose estimation part, but prohibit the multi-person pose estimation and face estimation functions.

6.1.2 IK Algorithm of Robotic Arm Control

HIWONDER robotic arm has 6-degree of freedom thus 6 servos. Among the 6 servos, only 4 of them determine and contribute to the arrival of the end point in the space. Thus the turning angles of the 4 servos are vital parameters to be determined. (x, y, z) denotes the given end point, $x-y$ plane is the operation plane, and z denotes the height, α denotes the angle between the gripper and the operation table, l_0, l_1, l_2, l_3 denotes the length of each sub-arms, θ_6, the turning angle of base of HIWONDER, thus can be calculated as,

$$\theta_6 = \arctan \frac{y}{x}.$$

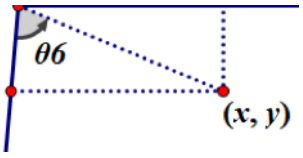


Figure 13: Close look of θ_6.

With l_1 and l_2 known, we can derive θ_4 and θ_5 with $Z_1 + Z_2$ and m_1 known. $n_1 = Z_1 + Z_2$ can be calculated as

$$n_1 = Z_1 + Z_2 = z - l_0 - l_3 \times \cos\alpha,$$

And m_1 can be calculated as

$$m = x^2 + y^2,$$

$$m_1 = m\text{–}l_3 \times \cos\alpha.$$

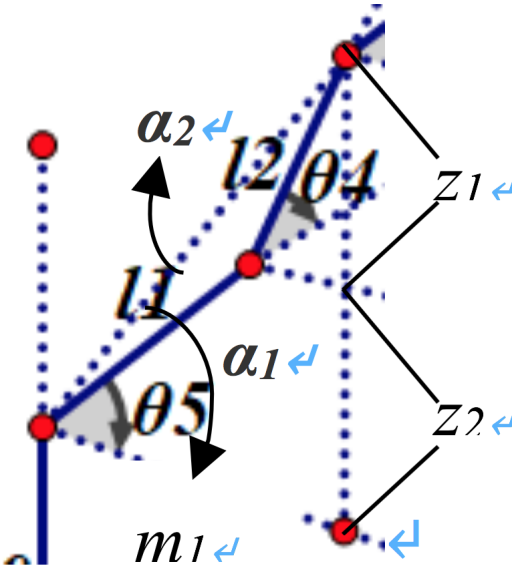


Figure 14: Close look of θ_4 and θ_5.

Therefore θ_4 can be calculated according to Cosine Theorem as

$$\cos\theta_4 = -\frac{{m_1}^2 + {n_1}^2 - {l_1}^2 - {l_2}^2}{2l_1l_2}.$$

θ_5 can be calculated as

$$\cos\alpha_1 = \frac{m_1}{\sqrt{{m_1}^2 + {n_1}^2}},$$
$$\cos\alpha_2 = \frac{{m_1}^2 + {n_1}^2 - {l_1}^2 - {l_2}^2}{2l_1\sqrt{{m_1}^2 + {n_1}^2}},$$
$$\theta_5 = \alpha_1 - \alpha_2.$$

With θ_4 and θ_5 known, θ_3 can be calculated as

$$\theta_3 = \alpha - \theta_5 + \theta_4.$$

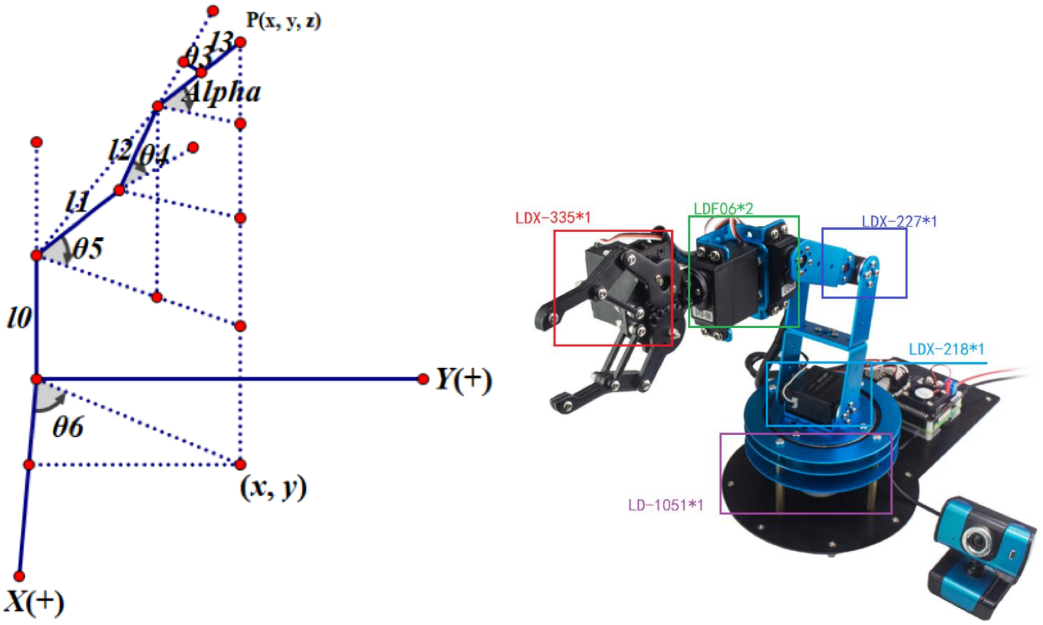


Figure 15: HIWONDER robotic arm and its degree of freedom.

Since the robotic arm also have scope of movement in the space, we cannot assign any end point to the robotic arm. To exploit and visualize how the arm can go, we plot all reachable points of HIWONDER on a 2D scatter plot. The robotic arm definitely cannot reach the given point if either x or y is beyond this scope. Even though x and y are all within this scope, the value of z will also determine whether the arm can access there. Therefore, the HIWONDER robotic arm are not able to reach the points plotted with any height. During the control we should carefully define x, y, and z.

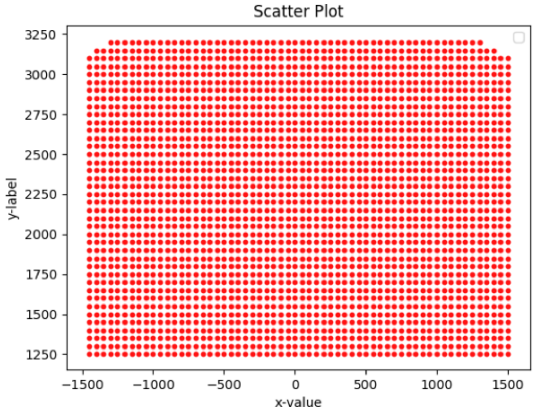


Figure 16: Achievable points of HIWONDER on 2D plane.

6.1.3 Object Detection

Object detection, tracking, and segmentation is a typical topic of computer vision in machine learning. Detection problems are usually handled by convolutional neural network (CNN) like R-CNN (Regions with CNN features), SSD (Single Shot Multi-Box Detector). R-CNN is a high capacity CNN propagating the region-proposals from bottom to top. SSD is a feedforward CNN where each feature map cell is linked to a set of default boxes. Tracking problems used to be handled by online discriminative classifier and now the most popular kind of models is Siamese Tracker. Siamese models are characteristic as vector output in learning the similarity function of two inputs. This project requires fast and accurate real-time object detection. Those state-of-art methods are all satisfying, but since our processor is not necessarily a laptop with CPU, we need to select the model best compatible with our processor. The size of capture frame by camera is 640×480. The saved initial bounding box should follow the same format in OpenCV model.

6.2 Design Description

6.2.1 Hand & arm pose estimation

The realization of hand & arm pose estimation with OpenPose can be divided into five sub-tasks:

1. Use OpenPose to extract the keypoints of body and left hand.
2. Use keypoints to determine the movement of the hand in left and right directions.
3. Use keypoints to determine the movement of the hand in up and down directions.
4. Use keypoints to determine the movement of the hand in forward and backward direction.
5. Use keypoints to determine the hand gesture.

For the first sub-task, OpenPose has already provided python API, which means we can use python script to call the function in OpenPose. What's more, it provides the tutorial of python API. For each keypoint, it is an array of three elements. If we think of an image as a two-dimensional plane, the X-axis represents the left and right direction, and the Y-axis represents the up and down direction, then the first element of a keypoint is its x-coordinate in that image and the second element is its y-coordinate in that image, and the third element is a number between 0 and 1, which represents how likely OpenPose thinks this keypoint is correct.

After we finish first sub-task, and before we move to other sub-task, we sum up the third element of every hand keypoint, if the sum is less than 1, which means OpenPose is basically unlikely to confirm or detect the hand, we're going to assume that the hand is not moving at that moment. As long as OpenPose can confirm and detect the hand, we will move on to second and third sub-task

For the second sub-task, we can only make it after the first sub-task is finished. After extracting the keypoints of body and hands, we need to determine the movement of the hand in left and right direction. To achieve it, we create a variable to record the x-coordinate of the keypoint which represents the wrist on the previous frame. Then at each frame, we calculate the difference between the x-coordinate of that keypoint in current frame and the corresponding x-coordinate on the previous frame. If the difference is larger than a positive threshold, we regard the hand is moving to right between this two frame, else if the difference is less than a negative threshold, we regard it is moving to left. According to different difference value, we regard the moving speed of the hand is different. The reason why we set two thresholds here is that, OpenPose cannot detect every keypoint with one hundred percent accuracy, which means even if the hand is not moving, the keypoints will wobbles a little bit. To eliminate this error, we set the thresholds to make sure the hand is really moving.

For the third sub-task, it is same as the second sub-task, we just need to create a variable to record the y-coordinate of the keypoint which represents the wrist on the previous frame, and calculate the difference, and determine the movement of hand in up and down direction together with its speed. Here, of course, we also need to set the threshold to eliminate the error.

For the fourth sub-task, it is the hardest sub-task among all five sub-tasks, because we only get 2-dimensional coordinate of each keypoint from OpenPose, and to accurately determine the movement of hand in 3-dimensional space, it is not enough. So here, instead of directly using the coordinates of keypoints to determine the movement, we use the distances between two specific pairs of keypoints. Through observation and lots of data testing, we found that when we are moving hands forward, our arms tend to straighten, in this case, the length of the line between the wrist and elbow projected onto the 2-dimensional image is shortened, while the length of the line between the elbow and shoulder on 2-dimensional image will be longer. Nearly every time we move hand forward or backward, these two length will change according to this rule, but when the hand move only in left and right or up and down directions, these two length can change as well. To avoid misjudgment in this case, we made some compromises, only when we judge from the previous two sub-tasks that the hand is neither moving in left or right direction nor moving in the up and down direction will we move to this sub-task to judge whether the hand is moving backward or forward.

For the last sub-task, to determine the hand gesture, we need the keypoints that represent the five fingertips. By calculating the variances of the x-coordinates and y-coordinates of these five keypoints, we can determine whether the hand is in closed state or open. If the sum of these two variances is larger than a predetermined threshold, we regard the hand is closed. Else if the sum is less than another threshold, we regard the hand is open. In this way, we can determine some simple hand gesture to control the gripper's closure or opening.

Together, these five sub-tasks make up the whole hand & arm estimation process.

6.2.2 Socket Based Communication

To achieve TCP communication, we need to set the server and clients respectively. In TCP, clients can connect to server, and server will broadcast information to all its clients.

In our system, the hand & arm pose estimation part, is the server, while the controller of HIWONDER arm and the VR systems are the clients to that server, because the after finishing the hand & arm pose estimation, we need to send the information about hand movement and hand gesture to the arm controller and VR system, in this way the robotic arm can be controlled by user's hand, and the VR model can also synchronize with robotic arm.

To make the hand & arm pose estimation part serve as a server, we need to use multi-threading. It totally need 4 threads, one is the main threads, which is responsible for creating other 3 threads and assign different work to them. Also, the main threads will keep listening to the channel through a while loop, and create a new thread which is responsible for a new connection if a new client's request is received . In our case, since the server only need to connect with VR system and robotic arm controller, the main thread will only create two extra threads for connection. The other thread left is used to perform hand & arm pose estimation.

Information exchange between threads is implemented through global variables. If the thread for pose estimation gets any movement information, it will store that information in the corresponding global variables, and use a semaphore to wake up the two other threads which are responsible for connection, after that, this thread will fall asleep. The threads that are responsible for connection won't send any information to the clients unless it is woken up by the semaphore, then they will refer to the corresponding global variables to fetch the information and send it to clients. Once all these two threads have finished sending information, they will wake up the pose estimation thread, then they will fall asleep. In this case, we can make sure two clients will receive exactly the same information all the time, and no information will be sent multiple times. Because once a new information is

generated by pose estimation thread, it won't generate the next information until the two connection threads have finished their jobs, and the connection threads won't send the same information multiple times, cause only when a new information is generated will they be woken up and send the new information to client.

To make robotic arm controller and VR system serve as clients, we also need to use multi-threading. Since the case of VR system is the same as that of robotic arm controller, here we will only introduce robotic arm controller. For the robotic arm controller, it will have two threads. One thread is responsible for receiving information from server, and another thread is responsible for controlling the arm according to the received information.

Information exchange between threads is also implemented through global variables as mentioned above, but the strategy is a little bit different. When information receiving thread received any control information, it will record that information in corresponding global variables and then wake up the arm controlling thread through a semaphore, and the arm controlling thread will read the information through global variables, initialize the global variables, then do the operation. But when the arm controlling thread is busy, the information receiving thread will keep working and "add up" the received information. For example, if information received a information that tell the arm to move to left with distance x, and later it received another information that tell the arm to move to left with distance y, then the information receiving thread will sum these two information up as a information that tell the arm to move to left with distance x+y. By using this strategy, we can prevent any information missing, because the information will always be received and stored.

6.2.3 Controll of Robotic Arm

The HIWONDER is controlled by python scripts. `main.py` calls different functions to realize the control. Function `kinematic_analysis(x, y, z, Alpha)` is to calculate the turning angels of the 4 critical joints as well as test whether each turning angle is feasible. If the angle is beyond the turning scope of the servo, then the function will return `false` and deny the request of reaching that end point. Function `ki_move(x, y, z, movetime)` first go through every possible value of α which is the angle between the gripper and the operation table and try `kinematic_analysis(x, y, z, Alpha)`, ranging from -25 degree -65 degree. The best α is chosen. The turning angle of the 4 critical joints are thereby derived by calling `kinematic_analysis(x, y, z, bestAlpha)`. Funciton `setServo(pin, angle, movetime)` is called to set the angels of servos.

First, we initialize the pose and the position of HIWONDER and store the initial

end point position in `X`, `Y`, and `Z`.

Second, the signals from hand pose estimation are received. The first digit refers to the hand state. The opening degree of the gripper is controlled by setting the servo using `setServo(1, angle, movetime)`. Then following three digits denotes the moving direction on x, y, z axis. If the received the signal of x direction is `-1`, then the value of `X` in the script is updated as `X = X + (-1)*step`. The previous value of `X` is stored in `preX`.

Third, we call the function `ki_move(x, y, z, movetime)` to direct the end point of HIWONDER to the new position. During the moving time, the signals are still being received. The received signals are accumulated and we update the position of end point in the next round. For example, the received signal of x direction during the moving time is `1, -1, 1`, then the accumulated moving step in next round is `(1-1+1)*step`.

Additionally, if the new end point of HIWONDER instructed by us is tested to be beyond the reachable scope by `kinematic_analysis(x, y, z, Alpha)`, the end point position will be remained at `(preX, preY, preZ)`.

6.2.4 Object Detection

Before starting the object detection module, we need some preparations. In terms of devices, a camera and a processor should be ready. The processor should support python with version higher or equal to 3.5. We configure MaskRCNN and OpenCV models on the processor according to the instructions of open source. Important packages like cv, dlib, numpy, and tensorflow are to be installed. Trained models and weights should also be downloaded.

First of all, to capture the scene of the operation table with objects, we should install a camera on top of the robotic arm, with a clear view of all objects. We use cv package in python to open the camera.

Second, we should calibrate the position of the camera and the robotic arm. The boundary and the center of the operation tables is clearly labeled. A python scrip is run to calibrate the camera. A "+" is drawn in the center of the captured scene. We adjust the camera to make the center of the frame image coincide with that of the operation table. To determine the placement of robotic arm, we first set the robotic arm to stretch to the center of its reachable area, and then coincide the end of the robotic arm (which is the gripper) with the center of the operation table. In this way, the center of the operation scope and the center of captured scene is overlapped, thus synchronizing the detected area with the operation area.

Third, MaskRCNN model is run to initialize the bounding box. MaskRCNN will rec-

ognize the type of the objects and output the positions and the sizes of the bounding boxes. This message is saved locally and sent back to VR platform (on server) by TCP/IP socket for initialization of the virtual scene with objects. MaskRCNN here is used for better automation. MaskRCNN as a detector definitely has limitation in detection scope. We can manually complement the initial bounding boxes if some objects are ignored. Besides, if the processor does not support any kind of detector, we can also initialize all the bounding boxes manually, sacrificing automation.

Finally, OpenCV algorithm reads the file saving the initial bounding boxes of objects and start tracking. OpenCV will also output the positions of centers of the objects. Similarly, those position data are sent back to VR (on server) to update the position of object models.

6.3 Manufacturing Plan

6.3.1 System related materials

The materials related to our system are as follows:

- A server with GPU at least GTX 1080Ti for running OpenPose and Unity
- A AMD64 based laptop or ARM64 based processor like Raspberry Pi or RK3399 for controlling the robotic arm
- Two cameras. One is for pose estimation and another is for object detection.
- A router for communication between server and other processors.

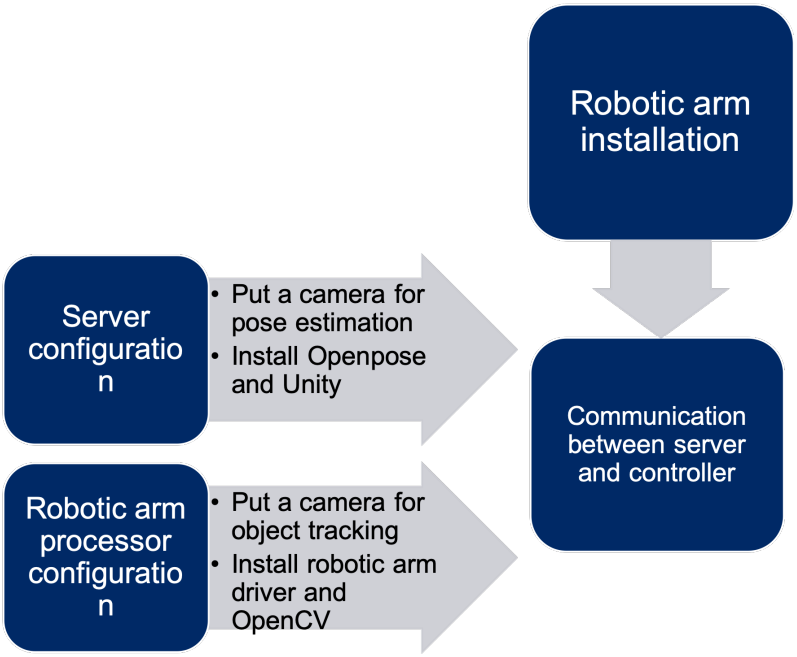


Figure 17: Preparation process

In the robotic arm installation part, all what we need to do is to follow the manual related to that specific robotic arm and install it. One key issue is that we should be sure of safety since it is very important.

For the Server configuration, Ubuntu 16.04 is highly recommended as an OS running on the server. The reason is that running OpenPose on Ubuntu is relatively easy and stable. It is almost impossible to exist the condition that OpenPose crashes with out some human errors. We should install the GPU driver and use GPU mode OpenPose. This will be highly useful in improving the computation of server and the FPS will be much higher than using CPU mode. What's more, we need to install Unity in Ubuntu. Although the official Unity doesn't support Ubuntu, there exists some unofficial versions and there don't exist some major problems. Therefore, it is good to have a server installed in Ubuntu16.04. And installing OpenPose and Unity on that is fine.

For Robotic arm processor configuration, we need to put a camera for object detection. We need to install robotic arm driver to drive the robotic arm and install OpenCV for detection. If we are using the laptop, we can even install some more powerful detection system like siammask or maskrcnn for object detection. But it may not work that good in the ARM64 based processor.

The last part is to setup the communication. The first think we should make sure is

if the processor or server has WLAN card. If not, we need to buy a WIFI adapter. Then, since we decide to use TCP for socket based communication. It is important to know the server's IP address since server will always act as a transmitter and never becomes a receiver. All the other processors without server should know its IP address and all the processors including the server should be in the same LAN. In other words, all the processors should connect to one router.

For the whole preparation process, robotic arm installation server configuration, and robotic arm processor configuration can be done at the same time. After that, the final step is to build and test the communication between server and controller. Figure17 just shows the whole preparation process.

6.3.2 System running process

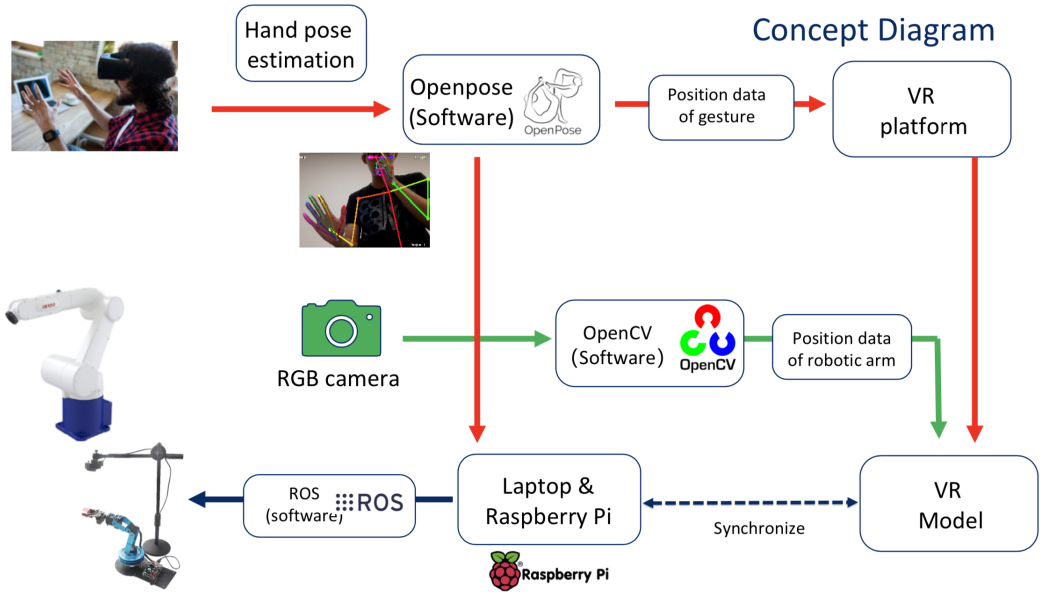


Figure 18: System running process

For the hand pose estimation part, we use Openpose to catch and track people's movement. Then it will feed back the data of the position of the gesture that our user do and output this data to the VR platform and laptop. At the same time, we use OpenCV to do the object detection, there is a small RGB camera on the robotic arm, it will ensure the position of the object and feed back the data of that and output the data to the VR platform. In the VR platform, the coordinate of the robotic arm and that of the object will synchronize with each other and finally output a 3D VR model. This 3D VR model will be output to laptop and Raspberry

Pi, and after all the data is transferred to the laptop and Raspberry Pi, we use ROS to control the robotic arm.

6.3.3 Budget consideration

The initial budget for each team is within 5k RMB. Since some of the materials like a server, two cameras and a router is available in the iDo Lab, it is not necessary for us to purchase it. We even have a DENSO robot in the lab, so when we are testing the functionality of DENSO robot, it is also unnecessary to cost some monet on budget. Finally, we have spent 600 RMB to buy the RK3399 microprocessor, 200RMB to buy the relevant products, 1600RMB to buy a new robotic arm. Thus the total cost is within 2.5k RMB. So the initial budget is enough.

6.4 Validation Plan

6.4.1 Pose estimation

Run Openpose to check FPS and accuracy of estimation. FPS is easy to test. Se can modify the program and let it output the time for handling each frame. Then we can calculate FPS. For accuracy mAP, it is hard to test since it needs professional dataset. Therefore, we must use some intuitive method to check it. One straightforward way is just to check whether the video is running smoothly and whether the returning state of hand and arm pose is correct. If it is running smoothly and returning the hand and arm pose correctly, then we can regard it as validated.

6.4.2 Object detection

Turn on the camera in robotic arm to check if it can run object detection. If we can run the object detection in a smoothly and relatively correct way, then we can regard it as validated. Here, the correct means that the detection system can correct detect the object and then draw a rectangle on that screen containing that object and return the correct pixel-level coordinates. This testing procedure is similar to the previous one.

6.4.3 Robotic arm

Turn on controller to check if it can control robotic arm manually. Then turn on the processor and check if we can control the robotic arm by using processor. This is to make sure both the robotic arm itself and robotic arm related processor can work at the same time. And another task is to run a simple IK program to check if it has any problem. If it can be finished successfully, then we we can finish this part.

6.4.4 Communication

Test communication between server and laptop or raspberry pi. Use a testing algorithm always sending testing data to the several processors to check if the communication works. What's more, we can type ping xxx.xxx.xx.xx on command line in the processor to test if it can connect to the server and the response time. This can make sure our communication time and if it is short to 20ms, then it is fine. If it is much longer, then the network may not be suitable for us to implement this system.

6.4.5 Synchronization

Open the whole project to check if the robotic arm in VR and real robotic arm have the same behavior. This is a little bit abstract, but if we think that they have the similar behavior, then we can regard this part as validated.

7 TimelinePlan

The project is divided into four sections. Section 1 is focusing on hand pose estimation and object detection, section 2 is more based on object detection on the robotic arm, while section 3 is about robotic arm installation and programming, finally, section 4 is about synchronizing the whole system together. We have already finished the pose estimation, object detection and robotic arm installation and programming part. We are now focusing on synchronization. We want to let the robotic arm in the real world scenario and VR system have the same behavior. The design Expo is on 11th December, by that time, we will definitely finish all the work before that time and make our customer and users satisfactory with our product.

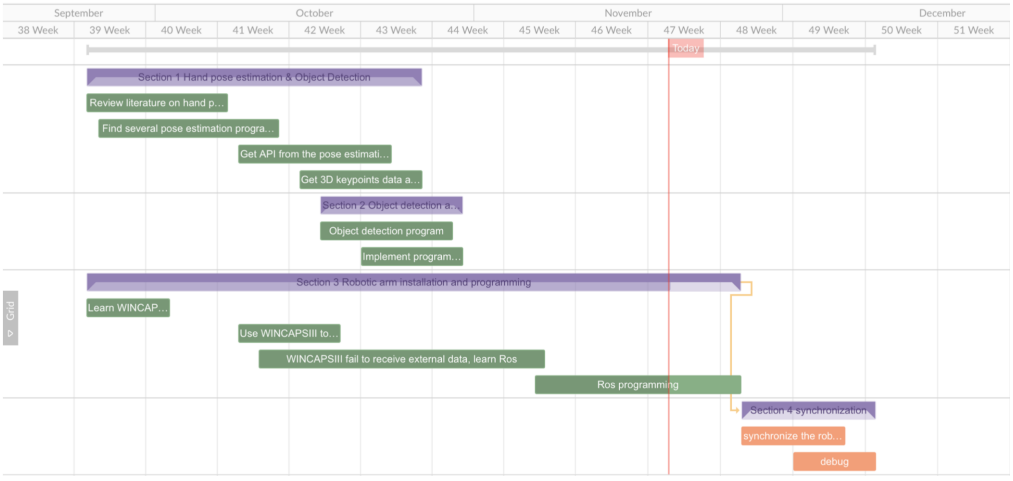


Figure 19: Gantt Chart

8 Conclusions

Our project, VR-Enabled Gesture Control of Robotic arm is a totally new proposal in "smart factory", but shares similarities with KIT learning factory and Digital Twin, which are the benchmarks. Our customer requires less time delay and the accuracy of the robotic arm following users' gestures. According to customer's requirements, we transfer them to the engineering specifications, our target is to ensure the accuracy and real-time of our system.

We totally have 4 parts to accomplish, handarm pose estimation, object detection, socket based communication and robotic arm. For handarm pose estimation part, we choose Openpose, AlphaPose and Zemmermann's to make the capture and track of users' gestures, we want to control the accracy of this pose estimation to be within 62mPA and the speed of pose estimation to be 20FPS, but when we run these three software together, the processing speed will lower down to 15 FPS and accuracy will also change to 65 mPA, these changes are within the tolerance of customer requirements, which means in the handarm pose estimation part. For object detection part, we have several choices, You Only Look Once(YOLO), OpenCV and SiamMask to realize that function. To select the best one, we test and compare the performance like speed, weight, limitations of detection, tracking function and accuracy, of these sofwares on the processor such as Raspberry Pi and RK3399, and comparing to YOLO, OpenCV performs better, it has no limitation in object type and the speed of OpenCV on CPU is 30 FPS while that on RK3399 is 50 FPS, which meets our demand. To synchronize all parts, communication is important, in socket based communication field, TCP and UDP are two commonplace protocols. We compare them in three aspects communication quality, communication time and ability for broadcasting, and finally decide to use TCP. Next, for the robotic arm control laptop, RK3399 and Raspberry Pi, the architecture of laptop is AMD64 which is different with that of arm64, belonging to RK3399 and Raspberry Pi. For the robotic arm control software, we initially want to use WINCAPSIII since it's the official software of denso, however, after test we found it can't receive the external data which means it can't fit in our system. Then we find another software ROS(Robot Operating System), but due to version, it can't support the Raspberry Pi and RK3399 can only support newest one. Thus, based on ROS, we find two choices, Inverse Kinematics(IK) and Geometric angle. Since IK is more convenient and we just need to set the destination and the relevant algorithm will be automatically calculated, it is preferred. Finally, for the robotic arm itself, as the industrial robotic arm is very precise and always turns on the protection mode, which will lock itself, we bought another laboratory-level robotic arm, HIWONDER and work on it.

This project basically realize its initial target, users can see the robotic arm HI-

WONDER follow and imitate their gestures and movement of hand. In this project, we have met a lot of difficulties and problems, we need to find the relevant solutions, we need to select the best performed one, we need to give up some solutions due to it can not fit in the whole system. However, we finally build the system successfully and we believe it will make our customers and users pleased, this project will have a very promising future and will perform well in the future Industry 4.0 time.

9 Analysis of Potential Problems

Until now, we have already synchronized the whole system, Openpose could catch and track the gestures of users and send that information to VR platform, which means the users could see how the robotic arm follow their gestures in a screen. Besides, OpenCV has also done a great job in object detection. Our project has a great response on the HIWONDER robotic arm, which means the HIWONDER robotic arm can follow the gestures made by the users.

The biggest problem now are as follows:

- How to let robotic arm in VR system and real scenario follow the real hand and arm pose estimation well.
- How to synchronize robotic arm between that in VR system and in real scenario.
- How to make our project impressive. Since we decide to use the HIWONDER robotic arm, it looks not that as impressive and fancy as the DENSO robotic arm. Therefore, we should have some idea to make out project impressive enough.

10 Appendix

kinematics.py

```
#!/usr/bin/env python3
# coding:utf8

#
from math import *
import LeArm
import time
import matplotlib.pyplot as plt
import LeConf

d = []
for i in range(0, len(LeConf.Deviation), 1):
    if LeConf.Deviation[i] > 1600 or LeConf.Deviation [i]< 1400:
        print("out_of_range")
    else:
        d.append(LeConf.Deviation[i] - 1500)

LeArm.initLeArm(tuple(d))
targetX = 0
targetY = 0
targetZ = 4000
lastX = 0
lastY = 0
lastZ = 4000

l0 = 960
l1 = 1050
l2 = 880
l3 = 1650


alpha = 0


def kinematic_analysis(x, y, z, Alpha):

    global alpha
    global l0, l1, l2, l3
    if x == 0:
        theta6 = 90.0
    elif x < 0:
        theta6 = atan(y / x)
        theta6 = 180 + (theta6 * 180.0/pi)
    else:
        theta6 = atan(y / x)
        theta6 = theta6 * 180.0 / pi
    y = sqrt(x*x + y*y)
    y = y-l3 * cos(Alpha*pi/180.0)
    z = z-l0-l3*sin(Alpha*pi/180.0)
    if z < -l0:
        return False
    if sqrt(y*y + z*z) > (l1 + l2):
        return False
    aaa = -(y*y+z*z-l1*l1-l2*l2)/(2*l1*l2)
    if aaa > 1 or aaa < -1:
        return False
    theta4 = acos(aaa)
    theta4 = 180.0 - theta4 * 180.0 / pi
    if theta4 > 135.0 or theta4 < -135.0:
        return False
    alpha = acos(y / sqrt(y * y + z * z))
    bbb = (y*y+z*z+l1*l1-l2*l2)/(2*l1*sqrt(y*y+z*z))
    if bbb > 1 or bbb < -1:
        return False
    if z < 0:
        zf_flag = -1
    else:
        zf_flag = 1
    theta5 = alpha * zf_flag + acos(bbb)
    theta5 = theta5 * 180.0 / pi
    if theta5 > 180.0 or theta5 < 0:
        return False

    theta3 = Alpha - theta5 + theta4
    if theta3 > 90.0 or theta3 < -90.0:
```

```
        return False
    return theta3, theta4, theta5, theta6


def averagenum(num):
    nsum = 0
    for i in range(len(num)):
        nsum += num[i]
    return nsum / len(num)


def ki_move(x, y, z, movetime):
    alpha_list = []
    for alp in range(-25, -65, -1):
        if kinematic_analysis(x, y, z, alp):
            alpha_list.append(alp)
    if len(alpha_list) > 0:
        if y > 2150:
            best_alpha = max(alpha_list)
        else:
            best_alpha = min(alpha_list)
        theta3, theta4, theta5, theta6 = kinematic_analysis(x, y, z, best_alpha)
        pwm_6 = int(2000.0 * theta6 / 180.0 + 500.0)
        pwm_5 = int(2000.0 * (90.0 - theta5) / 180.0 + 1500.0)
        pwm_4 = int(2000.0 * (135.0 - theta4) / 270.0 + 500.0)
        pwm_3 = int(2000.0 * theta3 / 180.0 + 1500.0)
        #print(str(pwm_3)+" "+str(pwm_4)+" "+str(pwm_5)+" "+str(pwm_6)+" ")
        LeArm.setServo(3, pwm_3, movetime)
        LeArm.setServo(4, pwm_4, movetime)
        LeArm.setServo(5, pwm_5, movetime)
        LeArm.setServo(6, pwm_6, movetime)
        time.sleep(movetime / 1000.0)
        return True
    else:
        return False


def plt_image(x_list, y_list):
    #plt.plot(x, y, label = 'target')
    plt.title('Scatter_Plot')
    plt.xlabel('x-value')
    plt.ylabel('y-label')
    # plt.scatter(x, y, s, c, marker)

    plt.legend()

    plt.scatter(x_list, y_list, s=10, c="#ff1212", marker='o')

if __name__ == '__main__':
    try:

        for y in range(1250, 3250, 50):
            for x in range(1500, -1500, -50):
                ok = kinematic_analysis(x, y, 200, 0)
                if ok:
                    print(x, y)
                    plt_image(x, y)
                else:
                    for a in range(0, -60, -1):
                        ok = kinematic_analysis(x, y, 200, a)
                        if ok:
                            print('a', (x, y))
                            plt_image(x, y)
                            #print('a', a)
                            break
                # time.sleep(0.1)
        plt.show()
    except Exception as e:
        print(e)
```

main.py

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-

import cv2
import numpy as np
import time
import threading
```

```
import signal
import LeArm
import kinematics as kin
import RPi.GPIO as GPIO
import time
import socket

def move2right(step):
    Xa = Xa + step
    return kin.ki_move(Xa, Ya, Za, 1500)

def move2up(step):
    Za= Za + step
    return kin.ki_move(Xa, Ya, Za, 1500)

def moveForward(step):
    Ya = Ya + step
    return kin.ki_move(Xa, Ya, Za, 1500)

def handstate0():
    LeArm.setServo(1, 2500, 200)

def handstate1():
    LeArm.setServo(1, 1800, 200)

def handstate2():
    LeArm.setServo(1, 1200, 200)

def handstate3():
    LeArm.setServo(1, 500, 200)

def move2pos(rightStep, upStep, forwardStep):
    #print (hand_state+"|",rightStep,"|",upStep,"|",forwardStep)
    global Xa, Ya, Za
    Xa_temp = Xa+rightStep
    Ya_temp = Ya+forwardStep
    Za_temp = Za+upStep
    print (hand_state+"|",Xa_temp,"|",Ya_temp,"|",Za_temp)
    if kin.ki_move(Xa, Ya, Za, 50):
        Xa = Xa_temp
        Ya = Ya_temp
        Za = Za_temp
        return True
    else:
        return False

def changeHandstate (hand_state):
    if hand_state == "0":
        LeArm.setServo(1, 2500, 50)
    elif hand_state == "1":
        LeArm.setServo(1, 1200, 5)
    elif hand_state == "2":
        LeArm.setServo(1, 800, 50)
    elif hand_state == "3":
        LeArm.setServo(1, 500, 50)
    #else:
        #print("fingers do not change.")

#Initial position
LeArm.runActionGroup('rest', 1)

Xa = 0
Ya = 1250
Za = 1500

#LeArm.setServo(3, 1000, 500)
#LeArm.setServo(4, 1000, 500)
#LeArm.setServo(5, 1000, 500)
#LeArm.setServo(1, 500, 500)
#kin.ki_move(2000, 2000, 2000.0, 1500)
#LeArm.setServo(1, 1200, 500)
#time.sleep(1)
#print("start to rotate")
#print(kin.ki_move(1500, 1100, 4000, 3000))
time.sleep(3)
#print("start to rotate")
#print(kin.ki_move(0, 1250, 1500, 3000))

#time.sleep(3)
#print("start to change")
#print(kin.ki_move(200, 0, 0, 3000))
#time.sleep(5)

```

```
hand_state = "0"
right = 0
up = 0
forward = 0

rightStep = 0
upStep = 0
forwardStep = 0

def updateArm (hand_state, right, up, forward):
    '''
    if right == 1:
        rightStep = 100
    elif right == -1:
        rightStep = -100
    else:
        rightStep = 0

    rightStep = right * 100

    if up == 1:
        upStep = 100
    elif up == -1:
        upStep = -100
    else:
        upStep = 0

    if forward == 1:
        forwardStep = 100
    elif up == -1:
        forwardStep = -100
    else:
        forwardStep = 0
'''


    rightStep = right * 35
    upStep = up* 35
    forwardStep = forward*10
    #print (hand_state+"|",right,"|",up,"|",forward)

    time_start = time.time()
    b = move2pos(rightStep, upStep, forwardStep)
    if b == False:
        global Xa, Ya
        if Xa > 0:
            Xa -= 100
        else:
            Xa += 100
        if Ya > 2250:
            Ya -= 100
        else:
            Ya += 100
        print("out_of_range!!!!!!!!!!!!!!!!!!!")
    time_end = time.time()
    #print("moving hand costs ", time_end-time_start)
    time_start = time.time()
    changeHandstate (hand_state)
    time_end = time.time()
    #print("opening hand costs ", time_end-time_start)
'''
import socket
SERVER = "192.168.1.110"
#SERVER = "127.0.0.1"
PORT = 10000
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER, PORT))
client.sendall(bytes("This is from Client",'UTF-8'))
while True:
  in_data =  client.recv(1024)
  #print("From Server :" ,in_data.decode())
  message = in_data.decode()
  index = 0
  str_count = 0
  hand_state=""
  X=""
  Y=""
  Z=""
  while True:
    if message[index] == "a":
      index = index + 1
      while str_count < 4:
        if message[index] == " ":
```

```
            str_count = str_count + 1
            index = index + 1
          if str_count == 0:
            hand_state += message[index]
          elif str_count == 1:
            Z += message[index]
          elif str_count == 2:
            X += message[index]
          elif str_count == 3:
            if message[index] == "b":
              break
            Y += message[index]
          index = index + 1
      if message[index] == "b":
        break
      index = index + 1

    print (hand_state+"|"+X+"|"+Y+"|"+Z)
    updateArm (hand_state, X, Y, Z)
    #out_data = input()
    #client.sendall(bytes(out_data,'UTF-8'))
    #if out_data=='bye':
      #break

client.close()

'''
X = 0
Y = 0
Z = 0
lock = threading.Lock()
s1 = threading.Semaphore(0)

def control_arm():
    while True:
        global X, Y, Z
        #print(X)
        #print(Y)
        #print(Z)
        s1.acquire()
        lock.acquire()
        #print("in")
        x = X
        y = Y
        z = Z
        print (hand_state+"|",x,"|",y,"|",z)
        X = 0
        Y = 0
        Z = 0
        #print (hand_state+"|",x,"|",y,"|",z)
        lock.release()
        updateArm (hand_state, x, y, z)

SERVER = "192.168.1.110"
#SERVER = "127.0.0.1"
PORT = 10000
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER, PORT))
client.sendall(bytes("This_is_from_Client",'UTF-8'))
op_thread = threading.Thread(target=control_arm)
op_thread.start()
while True:
    #print("come in")
    #print(str(X)+" "+str(Y))
    in_data =  client.recv(1024)
    #print("From Server :" ,in_data.decode())
    message = in_data.decode()
    index = 0
    str_count = 0
    hand_state=""
    #global X, Y, Z
    size = len(message)
    while True:
        if index >= size:
            break
        changed = False
        #if len(message) > 15:
        #    break
        #print(type(message))
        #info_list = message.split(" ")
        #print(info_list)
        #print("come in 2")
        if message[index] == "a":
```

```
257                 index = index + 1
258                 while str_count < 4:
259                     if message[index] == "_":
260                         str_count = str_count + 1
261                         index = index + 1
262                     if str_count == 0:
263                         hand_state = message[index]
264                     elif str_count == 1:
265                         if message[index] == "-":
266                             lock.acquire()
267                             X -= 2
268                             lock.release()
269                             changed = True
270                             index += 1
271                             continue
272                         lock.acquire()
273                         X += int(message[index])
274                         if int(message[index]) != 0:
275                             changed = True
276                         lock.release()
277                     elif str_count == 2:
278                         if message[index] == "-":
279                             lock.acquire()
280                             Y -= 2
281                             lock.release()
282                             changed = True
283                             index += 1
284                             continue
285                         lock.acquire()
286                         Y += int(message[index])
287                         if int(message[index]) != 0:
288                             changed = True
289                         lock.release()
290                     elif str_count == 3:
291                         if message[index] == "b":
292                             break
293                         if message[index] == "-":
294                             lock.acquire()
295                             Z -= 2
296                             lock.release()
297                             changed = True
298                             index += 1
299                             continue
300                         lock.acquire()
301                         Z += int(message[index])
302                         if int(message[index]) != 0:
303                             changed = True
304                         lock.release()
305                     index = index + 1
306             if message[index] == "b":
307                 if changed:
308                     s1.release()
309                 break
310             index = index + 1
311             print (hand_state+"|",X,"|",Y,"|",Z)
```

communication TCP-server.py

```
1  import socket, threading
2  class ClientThread(threading.Thread):
3      def __init__(self,clientAddress,clientsocket):
4          threading.Thread.__init__(self)
5          self.csocket = clientsocket
6          print ("New connection added: ", clientAddress)
7      def run(self):
8          print ("Connection from : ", clientAddress)
9          #self.csocket.send(bytes("Hi, This is from Server..",'utf-8'))
10         msg = ''
11         while True:
12             # data = self.csocket.recv(2048)
13             # msg = data.decode()
14             msg="a1_1_1_1b"
15             if msg=='bye':
16               break
17             print ("from client", msg)
18             self.csocket.send(bytes(msg,'UTF-8'))
19         print ("Client at ", clientAddress , " disconnected...")
20 LOCALHOST = "192.168.1.110"
21 #LOCALHOST = "127.0.0.1"
```

```
PORT = 10000
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((LOCALHOST, PORT))
print("Server started")
print("Waiting for client request..")
while True:
    server.listen(1)
    clientsock, clientAddress = server.accept()
    newthread = ClientThread(clientAddress, clientsock)
    newthread.start()
```

communication TCP-client.py

```
import socket
#SERVER = "192.168.1.110"
SERVER = "127.0.0.1"
PORT = 12345
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect((SERVER, PORT))
client.sendall(bytes("This is from Client",'UTF-8'))
while True:
  in_data =  client.recv(1024)
  print("From Server :" ,in_data.decode())

  #out_data = input()
  #client.sendall(bytes(out_data,'UTF-8'))
  #if out_data=='bye':
    #break
client.close()
```

communication UDP.py

```
import socket
import logging

def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    addr = ('192.168.1.104', 9999)
    s.bind(addr)
    logging.info('UDP Server on %s:%s...', addr[0], addr[1])

    user = {}  # {addr:name}
    while True:
        try:
            data, addr = s.recvfrom(1024)
            if not addr in user:
                for address in user:
                    s.sendto(data + ' enter chatting room...'.encode(), address)
                user[addr] = data.decode('utf-8')
                continue

            if 'EXIT' in data.decode('utf-8'):
                name = user[addr]
                user.pop(addr)
                for address in user:
                    s.sendto((name + ' leave chatting...').encode(), address)
            else:
                print('"%s" from %s:%s' %
                      (data.decode('utf-8'), addr[0], addr[1]))
                for address in user:
                    if address != addr:
                        s.sendto(data, address)

        except ConnectionResetError:
            logging.warning('Someone left unexcept.')

if __name__ == '__main__':
    main()
```

communication TCP-client.py

```
import threading
import socket
def recv(sock, addr):
    '''“
       data=sock.recv(1024)
        OSError: [WinError 10022] offer unknown parameters ”
    '''
    sock.sendto(name.encode('utf-8'), addr)
    while True:
        data = sock.recv(1024)
        print(data.decode('utf-8'))


def send(sock, addr):
    while True:
        string = input()
        message = name + ' : ' + string
        data = message.encode('utf-8')
        sock.sendto(data, addr)
        if string == 'EXIT':
            break

def main():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server = ('192.168.1.104', 9999)

    tr = threading.Thread(target=recv, args=(s, server), daemon=True)
    ts = threading.Thread(target=send, args=(s, server))
    tr.start()
    ts.start()
    ts.join()
    s.close()

if __name__ == '__main__':
    print(”-----Welcome to chatting room, exist is 'EXIT'-----”)
    name = input('enter your room:')
    print('------------------%s--------------------' % name)
    main()
```

Reference

[1] M. Hermann, T. Pentek and B. Otto, "Design Principles for Industrie 4.0 Scenarios," 2016 49th Hawaii International Conference on System Sciences (HICSS), Koloa, HI, 2016, pp. 3928-3937. doi: 10.1109/HICSS.2016.488

[2] BMBF-Internetredaktion. "Industrie 4.0." BMBF, 2 Dec. 2018, www.bmbf.de/de/zukunftsprojekt-industrie-4-0-848.html.

[3] "Industrie 4.0." Lernfabrik - Globale Produktion, globallearningfactory.com/industrie-4-0/.

[4] "Digital Twin" https://www.plm.automation.siemens.com/global/en/our-story/glossary/digital-twin/24465

[5] "Digital twin technology" https://www.i-scoop.eu/digital-twin-technology-benefits-usage-predictions/.

[6] "Robotic Standards for Collborative Robots" https://www.robotics.org/userAssets/riaUploads/file/6-Pat.pdf

[7] "Formula 1 Digital Twin" // https://www.ice.org.uk/news-and-insight/the-civil-engineer/july-2019/digital-twins-in-f1-built-environment

[8] Shih-Enal Wei, Varun Ramakrishna, Takeo Kanade, Yaser Sheikh; Convolutional Pose Machines, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4724-4732

[9] Tomas Simon, Hanbyul Joo, Iain Matthews, Yaser Sheikh; Hand Keypoint Detection in Single Images Using Multiview Bootstrapping, The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 1145-1153

[10] Zhe Cao and Gines Hidalgo and Tomas Simon and Shih-En Wei and Yaser Sheikh, OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Af[U+FB01]nity Fields

[11] Franklin, G., Powell, J. and Emami-Naeini, A. Feedback Control of Dynamic Systems (8th Edition)

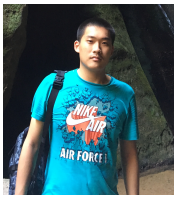
[12] DENSO WAVE robot manual - "https://www.denso-wave.com/authupd/21032/20085_contents4.zip"

[13] Wang Q , Zhang L , Bertinetto L , et al. Fast Online Object Tracking and Segmentation: A Unifying Approach[J]. 2018.

[14] Shafiee M J , Chywl B , Li F , et al. Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video[J]. 2017.

Bios

I am Jingying Wang, a senior student of UM-SJTU Joint Institute (JI) major in Electrical and Computer Engineering. In 2017, the first year of college, I worked as an trainee engineer at Nokia Shanghai Bell for one month, designing an integrated interface of code checking tools. In the second year of college, I won the Honorable Mention in 2018 Mathematical Contest in Modeling. In the same year, I joined System, Control, and Optimization Laboratory as a research assistant, developing a VR medical operation learning and practicing system. In 2019, I worked at Plug and Play China as operation intern and at JI as the teaching assistant of VE445 Intro to Machine Learning. I am interested and used to work in Image Assessment, Crowd Counting, Hand Pose Estimation, Action Quality Assessment, and Computer Graphics. My future career plan is to first pursue master degree and PhD degree and then work in industry. If possible, I will consider starting my own company.



My name is Zhiyuan Xiang, a senior student of UM-SJTU Joint Institute major in Electrical and Computer Engineering, I'm very interested in computer programming. In my sophomore year, I joined the VEX robotic team of SJTU, and won the second place in VEX national competition and second place in Asian Championships with my team. In the same year, I joined the Emerging Computing Technology Laboratory to do some research of approximate computing. In my junior year, I joined professor Yuan Bo's research group, did some research about the application of deep learning in high frequency trading. During the summer vacation, I went to Wenbo investment management Inc. for a one-month internship, I helped to build a market model for the company. Influenced by my personal experience, I'm extremly interested in computer science, especially for the artificial intelligence and machine learning, in the future, I plan to get a master degree first and then a PhD to further my study in this field.

My name is Minhao Jin, a senior student of UM-SJTU Joint Institute major in Electrical and Computer Engineering, I'm very interested in computer science and electrical and computer engineering. In 2018-2019, I worked in VEX robotic team. I'm mainly charged for wiring and programming and we win the championship in the campus competition, and the second prize in the national competition. In 2019, I began to work on indoor localization under Professor Han's supervision. I try to fuse Radio Frequency (RF) and Computer Vision (CV) to increase the flexibility of camera and keep the localization accuracy as well. This is a very challenging work since there is

not much work on it. I designed a system and fortunately, it has a nice performance. I really like this field.

My name is Wentao Yang, a senior student of UM-SJTU Joint Institute major in Mechanical Engineering. I like playing basketball, billiard, baseball and rugby. Besides, I like playing the piano. I like the sound of engine, the structure of gear matching with each other. In my freshman year, during the winter program holding by our institute, I went to Germany and learn more about Mechanical Engineering. In my junior year, I worked in Professor Hua Bao's laboratory about Solar Energy and worked with his graduate student publishing an essay in China. In my senior year, I took VM490 course of Professor David.Hung and began to learn about how to use Openfoam to simulate the fluid and watch the animation in Paraview. I really like the field of Mechanical Engineering, I believe whatever your idea is, only when you make it comes true, you have the hardware to run it or work on, then that makes sense. I plan to get my master and PhD degree abroad.

My name is Niall Halloran, currently on exchange for both fall and summer semesters for my senior year at UM-SJTU Joint Institute. At home, I study Mechanical Manufacturing engineering at Trinity college Dublin. This is my second design project – my first involved innovating for cyclists. My idea for vibrating handlebars for deaf cyclists achieved shortlist in two design competitions, 2019 Smarter travel campus awards and the Universal grand challenge; both based in Ireland. As a result – I really enjoy working with new ideas and emerging technologies. In the future I hope to work in a start-up or in an industry based around product innovation. Outside of engineering my interests include music, going to the gym and living a sustainable life.