Collaborative Robot Control with Hand Gestures

Ali Baccar

This thesis is done under the scope of the double degree agreement between Instituto Politécnico de Bragança and Université Libre de Tunis

Master's degree in Industrial engineering – Electrical branch

Supervisors

Paulo Leitão

Mohamed Aymen Slim

Bragança

2020

INSTITUTO POLITÉCNICO DE BRAGANÇA ipb Escola Superior de Tecnologia e de Gestão

DEPUIS 1973 ULT Université Libre de Tunis

Collaborative Robot Control with Hand Gestures

Ali Baccar

This thesis is done under the scope of the double degree agreement between Instituto Politécnico de Bragança and Université Libre de Tunis

Master's degree in Industrial engineering – Electrical branch

Supervisors

Paulo Leitão

Mohamed Aymen Slim

Bragança

2020

Abstract

This thesis focuses on hand gesture recognition by proposing an architecture to control a collaborative robot in real-time vision based on hand detection, tracking, and gesture recognition for interaction with an application via hand gestures. The first stage of our system allows detecting and tracking a bar e hand in a cluttered background using skin detection and contour comparison. The second stage allows recognizing hand gestures using a Machine learning method algorithm. Finally an interface has been developed to control the robot over.

Our hand gesture recognition system consists of two parts, in the first part for every frame captured from a camera we extract the keypoints for every training image using a machine learning algorithm, and we appoint the keypoints from every image into a keypoint map. This map is treated as an input for our processing algorithm which uses several methods to recognize the fingers in each hand.

In the second part, we use a 3D camera with Infrared capabilities to get a 3D model of the hand to implement it in our system, after that we track the fingers in each hand and recognize them which made it possible to count the extended fingers and to distinguish each finger pattern.

An interface to control the robot has been made that utilizes the previous steps that gives a real-time process and a dynamic 3D representation.

Resumo

Esta dissertação trata do reconhecimento de gestos realizados com a mão humana, propondo uma arquitetura para interagir com um robô colaborativo, baseado em visão computacional, rastreamento e reconhecimento de gestos. O primeiro estágio do sistema desenvolvido permite detectar e rastrear a presença de uma mão em um fundo desordenado usando detecção de pele e comparação de contornos. A segunda fase permite reconhecer os gestos das mãos usando um algoritmo do método de aprendizado de máquina. Finalmente, uma interface foi desenvolvida para interagir com robô.

O sistema de reconhecimento de gestos manuais está dividido em duas partes. Na primeira parte, para cada quadro capturado de uma câmera, foi extraído os pontos-chave de cada imagem de treinamento usando um algoritmo de aprendizado de máquina e nomeamos os pontos-chave de cada imagem em um mapa de pontos-chave. Este mapa é tratado como uma entrada para o algoritmo de processamento que usa vários métodos para reconhecer os dedos em cada mão.

Na segunda parte, foi utilizado uma câmera 3D com recursos de infravermelho para obter um modelo 3D da mão para implementá-lo em no sistema desenvolvido, e então, foi realizado os rastreio dos dedos de cada mão seguido pelo reconhecimento que possibilitou contabilizar os dedos estendidos e para distinguir cada padrão de dedo.

Foi elaborado uma interface para interagir com o robô manipulador que utiliza as etapas anteriores que fornece um processo em tempo real e uma representação 3D dinâmica.

Acknowledgements

First of all, I thank God for giving me the wisdom and strength for accomplishing this thesis.

I would like to express my deep gratitude, appreciation, and respect for my Professor Paulo Leitão my supervisor in Instituto Politécnico de Bragança and Professor Mohamed Ayman Slim my supervisor in Université Libre de Tunis for their endless support, encouragement, advice, patience, interest, and guidance throughout this thesis. I have gained enormously from their ideas, technical insights and profound thinking. I wish to deeply thank Professor Luis Piardi for his ideas, suggestions, cooperation and support in this project. Finally, great thanks and love to my parents for their constant understanding, encouraging, support and endless love to fulfill my goals.

Contents

List of Tables

List of Figures

List of the Acronyms

ANN	Artificial Neural Network
API	Application Programmer Interface
CNN	Convolutional Neural Network
CPS	Cyber-Physical Systems
DOF	Degree Of Freedom
FPS	Frame Per Second
HCI	Human Computer Interaction
HMI	Human Machine Interface
HSV	Hue Saturation Value
HMM	Hidden Markov Models
HRC	Human Robot Collaboration
IDE	Integrated development environment
IDLE	Integrated Development and Learning Environment
MIT	Massachusetts Institute of Technology
RF	Radio Frequency
SDK	System Development Kit
TCP	Tool Control Point
TOF	Time of Flight

Chapter 1: Introduction

1.1. Motivation and Framework

Traditional human-computer interaction devices such as the keyboard and mouse become ineffective interaction with the virtual environment applications because 3D environment need a new interaction device. An efficient human interaction with the modern virtual environment requires more natural devices. Among them the “Hand Gesture” human-computer interaction modality has become of major interest. The main objective of gesture recognition is to build a system which can recognize human gestures and utilize them to control an application.

Vision based hand gesture recognition recently became a high active research area with motivating applications such as sign language recognition , socially assistive robotics, directional indication through pointing , control through facial gestures , human-computer interaction (HCI) , immersive game technology, virtual controllers, etc.

Within the broad range of application scenarios, hand gestures are a powerful human interface components, however, their fluency and intuitiveness has not been utilized as computer interface. Recently, hand gestures applications have begun to emerge, but they are still not robust and are unable to recognize the gestures in a convenient and easy accessible manner by human. Several advanced techniques are either too fragile or too coarse grained to be of any universal use for hand gesture recognition. Especially techniques for hand gesture interfaces should be developed beyond current performance in term of speed and robustness to attain the needed interactivity and usability.

It is a difficult task to recognize hand gestures automatically from a camera input. It usually includes numerous phases such as signal processing, detection, tracking, shape description, motion analysis and pattern recognition. The general problem is quiet challenging because of several problems such as the complex nature of static and dynamic gestures, cluttered backgrounds, lightning changes and occlusions.

Trying to solve the problem in its generality needs elaborate techniques that require high performance against the issues.

Hand gesture recognition from video frames is one of the most main challenges in image processing and computer vision because it provides the computer the capability of detecting tracking, recognizing and interpreting the hand gestures to control various devices or to interact with several human machine interfaces (HMI).

1.2. Objectives

The objectives of this thesis is to develop a novel approach to the complete problem of hand gesture recognition. By proposing new algorithms to an existing problem which is a collaborative robot controlled manually.

Our work should satisfy numerous conditions:

The first requirement is a real-time performance. This is critical for complete interactivity and intuitiveness of the interface. This is measured by frames per second (fps). If there the execution time is log there will be a delay between the real event and the recognition. If the gestures are carried out in an extremely fast sequence, the event may not be recognized at all.

The second required condition is flexibility, and how well it combines with new applications and existing applications.

Third, the system should be practically precise enough to be used. The approach should be able to recognize the defined gestures at least from 90% to 100% of time to be successful and of practical use.

Fourth, is robustness which the system is able to detect, track and recognize different hand gestures successfully under different lighting conditions and cluttered backgrounds.

Fifth, the approach should be user-independent in which system must work for various persons rather than one particular person. The system must recognize hand gestures for different human hands of different scales and colors.

Finally, the system should be safe to be utilized in the industrial field, which mean it safe for human the manipulator and the surrounding workers and for objects around the system should not be harmed in any way.

1.3. Contribution

The major contributions of this thesis are the following:

We prepared a new real-time and accurate recognition for the detected hand posture detection and tracking using a Leap motion camera.

We created real-time hand tracking system from 2D and 3D videos using Intel RealSense camera.

We developed algorithms to further increase the efficiency of our system.

We developed a Human-Machine interface supported by multiple platforms which creates a 3D representation of our tracking system in Real-time.

1.4. Document Structure

This thesis includes 6 chapters:

Chapter 1 introduces the framework, vision based hand gesture processing stages and motivations of our work. The objectives to be accomplished and the contributions are also mentioned.

Chapter 2 provides the State of art which works related to gesture recognition, its applications and the potential applications in industrial environments were presented, also there is mention of works related of collaborative robots, their safety and regulations and the human usability and ergonomics.

Chapter 3 presents the methodology followed, our hardware choice, software used and different algorithms to detect hand postures in each hardware, recognition and processing, also finger tracking and recognition is mentioned

Chapter 4 proposes the architecture of the system were Leapmotion and Realsense operations for gesture detection were mentioned along with the algorithms used and the communication method between the script and the robot.

Chapter 5 presents the implementation of our methods, the results, evaluating the performance of our system and comparison between our methods.

Chapter 6 provides conclusions were brief comparison is made and future work.

Chapter 2: State of Art

2.1. Gesture Recognition

In the present world, the interaction with the computing devices has advanced to such an extent that as humans it has become necessity and we cannot live without it. The technology has become so embedded into our daily lives that we use it to work, shop, communicate and even entertain ourselves. It has been widely believed that the computing, communication and display technologies progress further, but the existing techniques may become a bottleneck in the effective utilization of the available information flow (Pantic M, 2008). To efficiently use them, most computer applications require more and more interaction. For that reason, human-computer interaction (HCI) has been a lively field of research in the last few years. Firstly based in the past on punched cards, reserved to experts, the interaction has evolved to the graphical interface paradigm. The interaction consists of the direct manipulation of graphic objects such as icons and windows using a pointing device. Even if the invention of keyboard and mouse is a great progress, there are still situations in which these devices are incompatible for HCI. This is particularly the case for the interaction with 3D objects. The 2 degrees of freedom (DOFs) of the mouse cannot properly emulate the 3 dimensions of space. The use of hand gestures provides an attractive and natural alternative to these cumbersome interface devices for human computer interaction. Using hands as a device can help people communicate with computers in a more intuitive way. When we interact with other people, our hand movements play an important role and the information they convey is very rich in many ways. We use our hands for pointing at a person or at an object, conveying information about space, shape and temporal characteristics. We constantly use our hands to interact with objects. Hand movements are thus a mean of non-verbal communication, ranging from simple actions (pointing at objects for example) to more complex ones (such as expressing feelings or communicating with others). In this sense, gestures are not only an ornament of spoken language, but are essential components of the language generation process itself. A gesture can be defined as a physical movement of the hands, arms, face and body with the intent to convey information or meaning. In particular, recognizing hand gestures for interaction can help in achieving the ease and naturalness desired for human computer interaction (Mitra S, 2007).

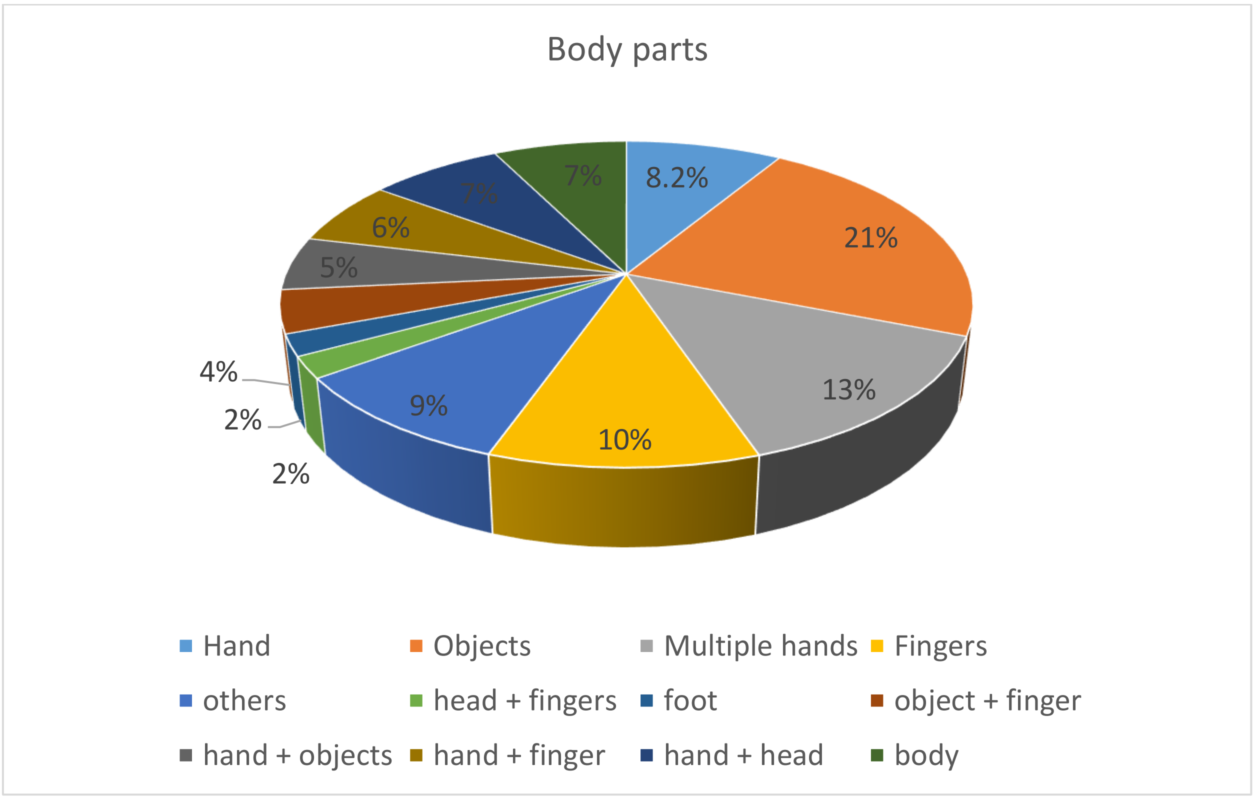


Figure 1 different body parts and objects identified in the literature employed for gesturing (Agrawal, 2012)

The graph above shows the importance of hand and finger gestures to humans as meaning of expressing their feelings and the notifications of their thoughts.

To abstract and model the human body parts motion several hand gesture representations and models have been proposed and implemented by the researchers. The two major categories of hand gesture representation are 3D model based methods and appearance based methods as shown in Figure 2.

The 3D model based hand gesture recognition has different techniques for gesture representation namely 3D textured volumetric, 3D geometric model and 3D skeleton model. Appearance based hand gesture representation include color based model, silhouette geometry model, deformable model and motion based model.

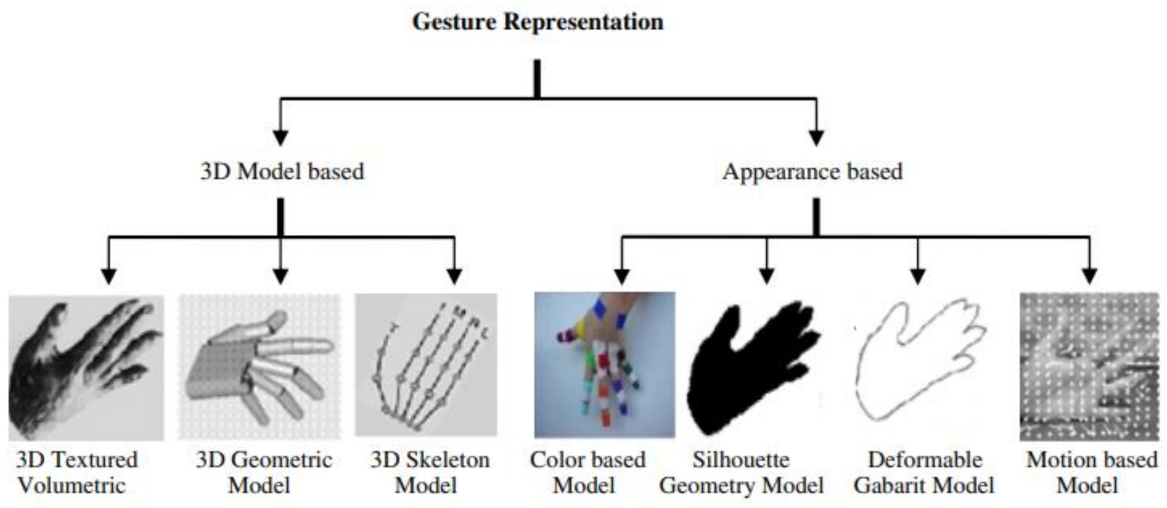


Figure 2 Vision based hand gesture representations (Bourke A, s.d.)

Most of the complete hand interactive mechanisms that act as a building block for vision based hand gesture recognition system are comprised of three fundamental phases: detection, tracking and recognition. This section of the research survey discusses some of the prominent vision based hand gesture recognition techniques used by most of the researchers by categorizing under the three verticals representing the three fundamental phases of detection tracking and recognition as shown in Figure 3.

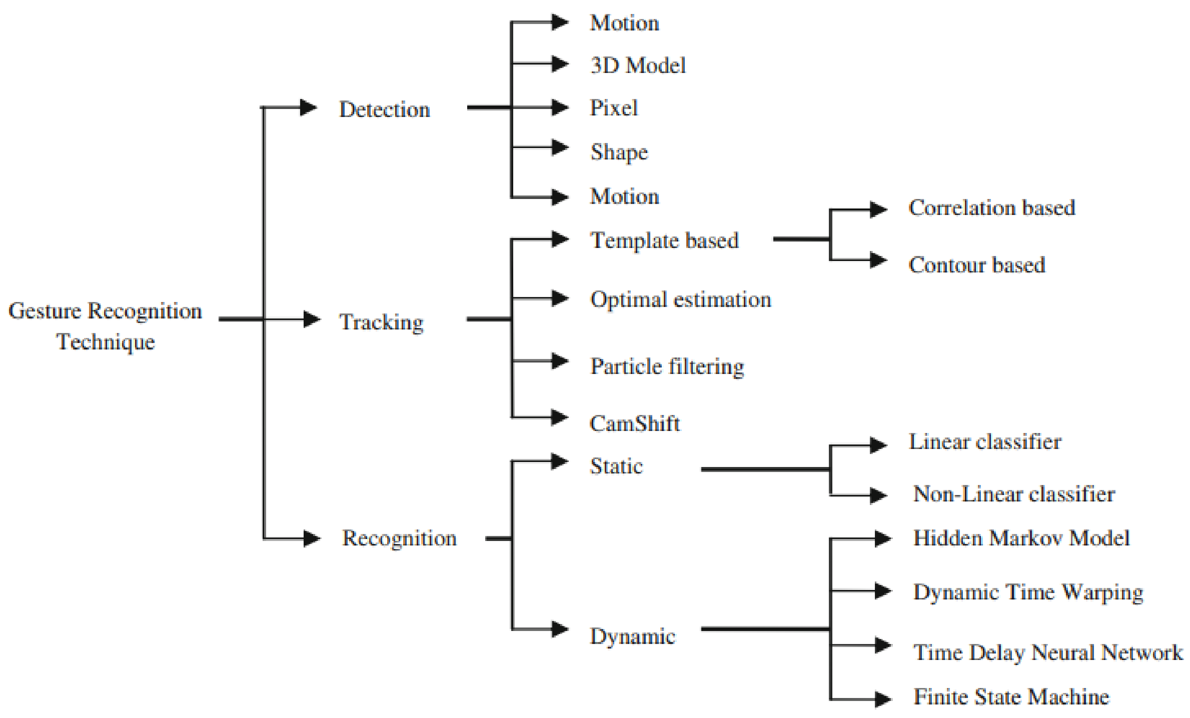


Figure 3 Vision based hand gesture recognition techniques (Agrawal, 2012)

2.1.1. Detection

The primary step in hand gesture recognition systems is the detection of hands and the segmentation of the corresponding image regions. This segmentation is crucial because it isolates the task-relevant data from the image background, before passing them to the subsequent tracking and recognition stages (Cote M, 2006) . A large number of methods have been proposed in the literature that utilize a several types of visual features and, in many cases, their combination. Such features are skin color, shape, motion and anatomical models of hands.

2.1.2. Tracking

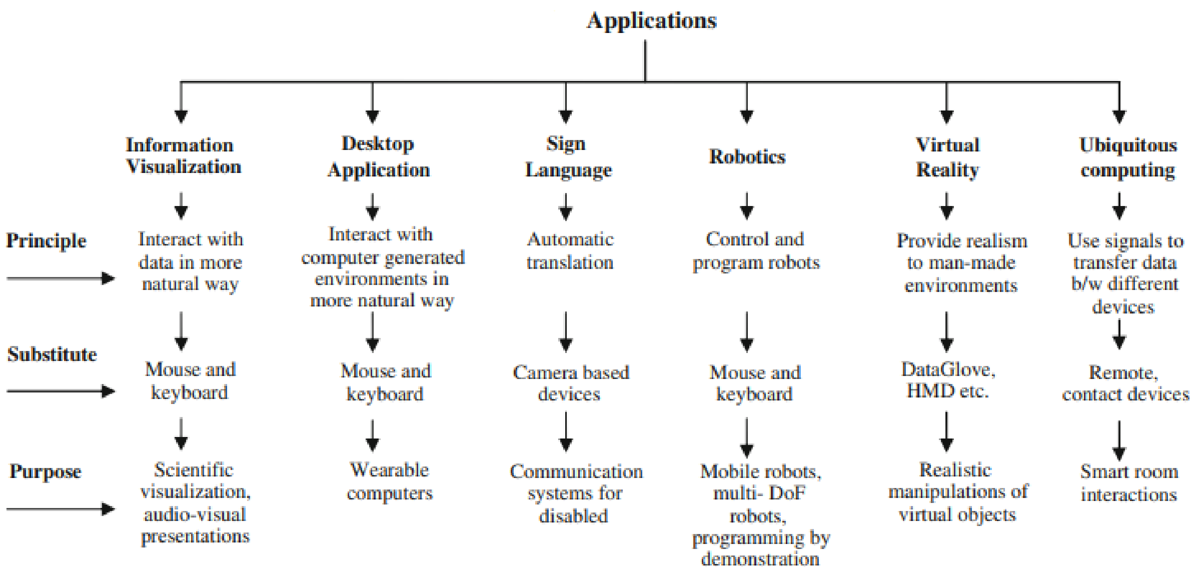
If the detection method is fast enough to operate at image acquisition frame rate, it can be used for tracking as well. However, tracking hands is notoriously difficult since they can move very fast and their appearance can change vastly within a few frames. Tracking can be defined as the frame-to-frame correspondence of the segmented hand regions or features towards understanding the observed hand movements. The importance of robust tracking is twofold. First, it provides the inter-frame linking of hand/finger appearances, giving rise to trajectories of features in time. These trajectories convey essential information regarding the gesture and might be used either in a raw form (e.g. in certain control applications like virtual drawing the tracked hand trajectory directly guides the drawing operation) or after further analysis (e.g. recognition of a certain type of hand gesture). Second, in model-based methods, tracking also provides a way to maintain estimates of model parameters variables and features that are not directly observable at a certain moment in time (Baxter, 2000).

2.1.3. Recognition

The overall goal of hand gesture recognition is the interpretation of the posture, location or gesture conveys of the hand or both hands. Vision based hand gesture recognition techniques can be further classified under static and dynamic gestures. To detect static gestures (i.e. postures), a general classifier or a template-matcher can be used. However, dynamic hand gestures have a temporal aspect and require techniques that handle this dimension like Hidden Markov Models (Wechsler, 2005).

2.1.4. Application domains

Vision based hand gestures recognition systems since its early days of exploration and research have found vital applications to a wide range of real life and real time scenarios. The evolution of human computer interaction has been paced up with the advances in pervasive computing and real time application scenarios of computing devices. These applications vary from Information Visualization, Desktop Applications, sign language, Robotics, virtual reality to games, medical environment, augmented reality and others.



Applications
Information Visualization
Desktop Application
Sign Language
Robotics
Virtual Reality
Ubiquitous computing
Principle
Interact with data in more natural way
Interact with computer generated environments in more natural way
Automatic translation
Control and program robots
Provide realism to man-made environments
Use signals to transfer data b/w different devices
Substitute
Mouse and keyboard
Mouse and keyboard
Camera based devices
Mouse and keyboard
DataGlove, HMD etc.
Remote, contact devices
Purpose
Scientific visualization, audio-visual presentations
Wearable computers
Communication systems for disabled
Mobile robots, multi- DoF robots, programming by demonstration
Realistic manipulations of virtual objects
Smart room interactions

Figure 4 Hand Gesture Application (Agrawal, 2012)

The next Chapter focuses on the robotics application as it is our main focus especially in the industrial environment.

2.2. Gesture Recognition Application and potential applications to Industrial Environments

2.2.1. Human-Robot collaboration

Robotic systems have already become essential components in various industrial sectors. Recently, the concept of Human-Robot Collaboration (HRC) has generated more interests. Human workers have unique problem-solving skills and sensory-motor capabilities, but are restricted in force and precision (Kruger, 2012). Robotic systems, on the other hand, provide better fatigue, higher speed, higher repeatability and better productivity, but are restricted in flexibility. Jointly, HRC can release human workers from heavy tasks by establishing communication channels between humans and robots for better overall performance. Ideally, a HRC team should work similarly as a human-human collaborative team in a manufacturing environment. However, time-separation or space-separation is dominant in HRC systems, which reduced productivity for both human workers and robots.

To recognize gestures in the HRC manufacturing context, it is beneficial to investigate into a generic and simplified human information processing model. Based on this generic model, we propose a specific model for gesture recognition in HRC. As shown in Figure1, there are five essential parts related to gesture recognition for HRC: sensor data collection, gesture identification, gesture tracking, gesture classification and gesture mapping, explained as follows.

- Sensor data collection: the raw data of a gesture is captured by sensors.
- Gesture identification: in each frame, a gesture is located from the raw data.
- Gesture tracking: the located gesture is tracked during the gesture movement. For static gestures, gesture tracking is unnecessary.
- Gesture classification: tracked gesture movement is classified according to pre-defined gesture types.
- Gesture mapping: gesture recognition result is translated into robot commands and sent back to workers.

HRC has many applications such as Pick and Place, packaging and palletizing, quality inspection and object manipulation.

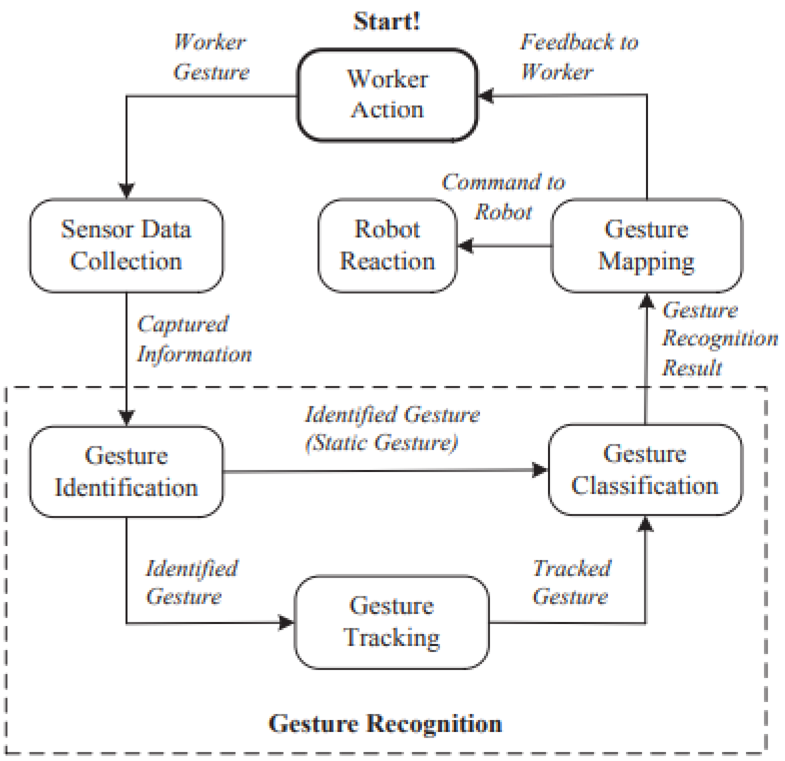


Figure 5 A process model of gesture recognition for human-robot collaboration. (Hongyi Liu, 2017)

2.2.2. Sensor technologies

Before gesture recognition process starts, raw gesture data need to be collected by sensors. In this section, different sensors in the literature are analyzed based on various sensing technologies. As shown in Figure 5, there are two basic categories of data acquisition: image based and non-image based approaches.

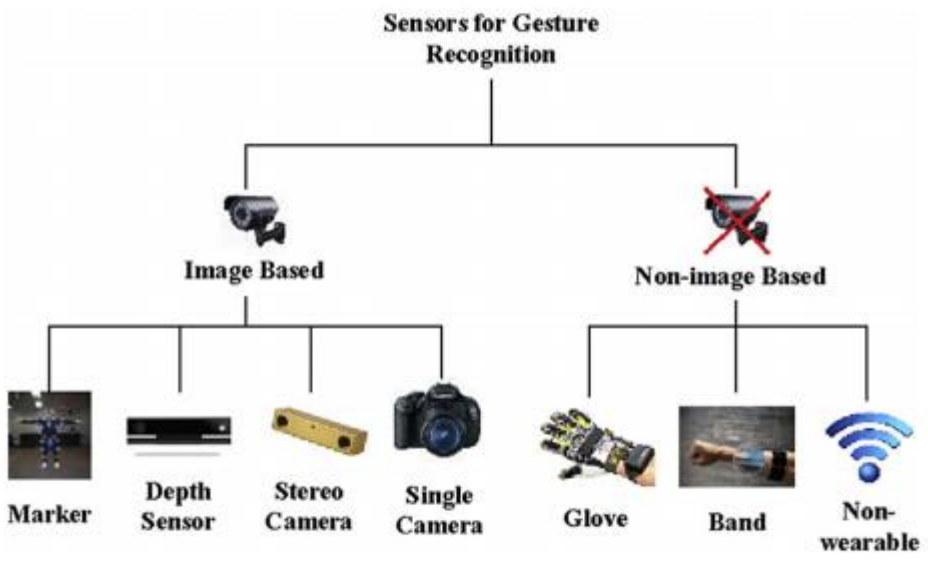


Figure 6 Different types of gesture recognition sensors. (Hongyi Liu, 2017)

2.2.2.1. Image based approaches

Technologies are often inspired by nature. As a human being, we use our eyes to recognize gestures. Therefore, for robots, it is reasonable to use cameras to "see" gestures. The image-based approaches are further divided into four categories.

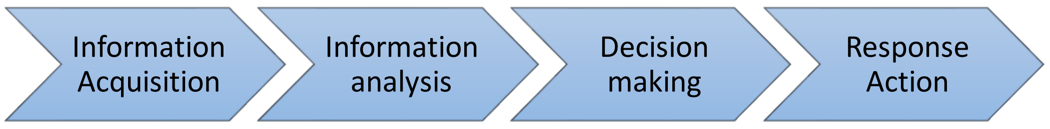


Figure 7 A four-stage model of human information processing

Marker

In marker-based approaches, a sensor is a conventional optical camera. In most marker-based solutions, users need to wear obvious. Today, we enjoy much faster graphical processing speed compared with twenty years ago. As a result, more gesture recognition sensors are available on the market.

Single camera

In the early 90th, researchers started to analyze gestures using a single camera. A drawback of single-camera-based approaches is the restriction of view angles, which affects a system's robustness. Recent research, however, applied a single camera in high-speed gesture recognition. The system utilizes the speed image sensor and specially designed visual computing processor to achieve high-speed gesture recognition.

Stereo camera

To achieve robust gesture recognition, researchers suggested stereo camera based approaches to construct 3D environment. They have been applied in applications that use two stereo cameras to construct 3D depth information. Many stereo camera based approaches follow a similar. Although stereo camera systems have improved robustness in outdoor environment, they still suffered from problems such as computational complexity and calibration difficulties (Wachs, 2011).

Depth sensor

Recently, depth sensing technologies have emerged rapidly. We define a depth sensor as a non-stereo depth sensing device. Nonstereo depth sensor enjoys several advantages compared to the traditional stereo cameras. For example, the problems of setup calibration and illumination conditions can be prevented. Moreover, the output of a depth sensor is 3D depth information. Compared with color information, the 3D depth information simplifies the problem of gesture identification. *A comparison of gesture identification accuracy by using color and depth information can be found in Time-of-Flight (ToF) technology is one of the popular depth sensing techniques. The fundamental principle of the ToF technology is to identify light travel time Recently, Microsoft Kinect 2 has applied the ToF technology. The advantage of the ToF technology is the higher frame rate. The limitation of the ToF technology is that the camera resolution highly depends on its light power and reflection.* (Hornegger, 2008)

Depth sensor provides a cheap and easy solution for gesture recognition. It is widely used in entertainment, education, and research, which has introduced a large developer community. With a large developer community, many open source tools and projects are available. Due to resolution restriction, currently, depth sensors are especially popular in body gesture recognition and close-distance hand and arm gesture recognition.

2.2.2.2 Non-image based approaches

Gesture recognition has been dominated by image-based sensors for a long time. Recent developments in MEMS and sensors have significantly boosted non-image based gesture recognition technologies.

Glove

Glove-based gestural interfaces are commonly used for gesture recognition. Usually, glove-based approaches require wire connection, accelerometers, and gyroscopes. However, a cumbersome glove with a load of cables can potentially cause problems in HRC manufacturing environment. Glove-based approaches also introduced complex calibration and setup procedures.

Band

Another contactless technology uses band-based sensors. Band based sensors rely on a wristband or similar wearable devices. Band-based sensors adopt wireless technology and electromyogram sensors, which avoid connecting cables. The sensors only need to contact with wrist; user's hand and fingers are released. One example is Myo gesture control armband (Labs, 2015) (Zhang, 2015). Recently, several band-based sensor gesture control systems have been reported.

Non-wearable

The third type of non-image based technologies adopts non wearable sensors. Non-wearable *sensors* can detect gestures without contacting human body.

Google introduced Project Soli, a radio frequency (RF) signal based hand gesture tracking and recognition system (Google, 2015).The device has an RF signal sender and a receiver. It is capable of recognizing different hand gestures within a short distance. MIT has been leading non-wearable gesture recognition technology for years. Electric Field Sensing technology was pioneered by MIT (Smith et al., 1998). A recent discovery from MIT introduced WiTrack and RF-Capture system that captures user motion by radio frequency signals reflected from human body. (Hongyi Liu, 2017)

As shown in Fig. 8(b), the RF-Capture system selects particular RF signals that can traverse through walls and reflect off the human body. The system can capture human motion even from another room with a precision of 20 cm. Although the precision is not acceptable in HRC manufacturing, non-wearable based technologies are promising and fast-growing sensor technologies for gesture recognition.

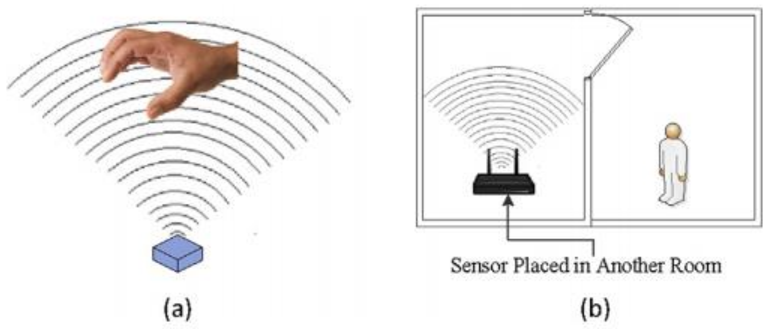


Figure 8 Project Soli and RF-Capture system: (a) concept of Project Soli; (b) concept of RF-Capture gesture capturing system. (Hongyi Liu, 2017)

2.2.2.3. Comparison of sensor technologies

Summarizing the advantages and disadvantages of different technologies. It is clear that there is no sensor that fits all HRC applications. Two observations of the sensor technologies are provided based on the above analyses: In indoor HRC manufacturing environment, depth sensors are the most promising image-based techniques. Depth sensors possess advantages of easy calibration and accurate data processing. A large application developer community exists, which provides immediate solutions. Non-wearable approaches are the most promising technology among non-image based approaches. They can avoid direct contact with users, which provide advantages in an HRC manufacturing environment. Non-wearable sensing is also a fast-growing field.

2.3. Collaborative robots: Safety and Regulations

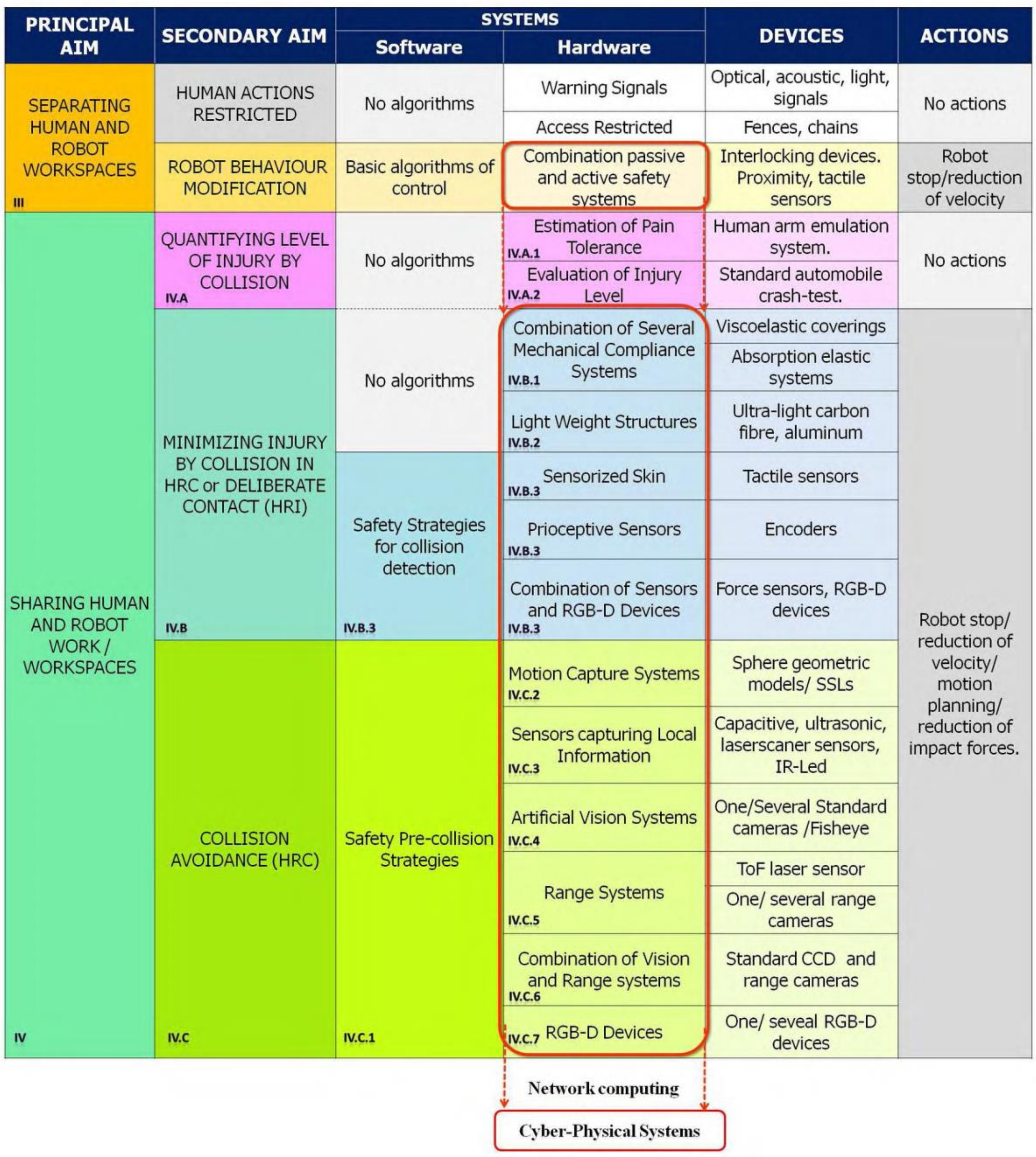
After many years of rigid conventional procedures of production, industrial manufacturing is going through a process of change toward flexible and intelligent manufacturing, the so-called Industry 4.0. In this paper, human–robot collaboration has an important role in smart factories since it contributes to the achievement of higher productivity and greater efficiency. However, this evolution means breaking with the established safety procedures as the separation of workspaces between robot and human is removed. These changes are reflected in safety standards related to industrial robotics since the last decade, and have led to the development of a wide field of research focusing on the prevention of human–robot impacts and/or the minimization of related risks or their consequences. This paper presents a review of the main safety systems that have been proposed and applied in industrial robotic environments that contribute to the achievement of safe collaborative human–robot work. Additionally, a review is provided of the current regulations along with new concepts that have been introduced in them. The discussion presented in this paper includes multidisciplinary approaches, such as techniques for estimation and the evaluation of injuries in human–robot collisions, mechanical and software devices designed to minimize the consequences of human–robot impact, impact detection systems, and strategies to prevent collisions or minimize their consequences when they occur.

2.3.1. A framework for safety in industrial robotic environments.

To provide a structured framework, a classification of the main safety systems in robotic environments is provided in Table 1, including the aims pursued by the safety systems, hardware and software systems that are employed, devices that are used, and the actions involved in each type safety system. Table 1 indicates the sections of the paper where each subject is covered.

The term of Cyber-Physical Systems (CPS) has been incorporated because of the ongoing improvements in intelligent manufacturing have significant implications on the usage of robot safety systems. In this way, the incorporation of network computing, connected devices and data management systems in manufacturing processes, including active safety systems, resulted in CPS.
Cyber-Physical System are defined as physical devices which are provided with technologies to collect data about themselves and their surroundings, process and evaluate these data, connect and communicate with other systems and initiate actions to achieve their goals (S. ROBLA-GÓMEZ 1, 2017).

Table 1 Classification of safety in industrial robot collaborative (S. ROBLA-GÓMEZ 1, 2017)



PRINCIPAL AIM	SECONDARY AIM	SYSTEMS Software	SYSTEMS Hardware	DEVICES	ACTIONS
SEPARATING HUMAN AND ROBOT WORKSPACES III	HUMAN ACTIONS RESTRICTED	No algorithms	Warning Signals	Optical, acoustic, light, signals	No actions
			Access Restricted	Fences, chains	
	ROBOT BEHAVIOUR MODIFICATION	Basic algorithms of control	Combination passive and active safety systems	Interlocking devices. Proximity, tactile sensors	Robot stop/reduction of velocity
SHARING HUMAN AND ROBOT WORK / WORKSPACES IV	QUANTIFYING LEVEL OF INJURY BY COLLISION IV.A	No algorithms	IV.A.1 Estimation of Pain Tolerance	Human arm emulation system.	No actions
			IV.A.2 Evaluation of Injury Level	Standard automobile crash-test.	
	MINIMIZING INJURY BY COLLISION IN HRC or DELIBERATE CONTACT (HRI) IV.B	No algorithms	IV.B.1 Combination of Several Mechanical Compliance Systems	Viscoelastic coverings	Robot stop/ reduction of velocity/ motion planning/ reduction of impact forces.
				Absorption elastic systems	
			IV.B.2 Light Weight Structures	Ultra-light carbon fibre, aluminum	
		Safety Strategies for collision detection IV.B.3	IV.B.3 Sensorized Skin	Tactile sensors	
			IV.B.3 Prioceptive Sensors	Encoders	
			IV.B.3 Combination of Sensors and RGB-D Devices	Force sensors, RGB-D devices	
	COLLISION AVOIDANCE (HRC) IV.C	Safety Pre-collision Strategies IV.C.1	IV.C.2 Motion Capture Systems	Sphere geometric models/ SSLs	
			IV.C.3 Sensors capturing Local Information	Capacitive, ultrasonic, laserscaner sensors, IR-Led	
			IV.C.4 Artificial Vision Systems	One/Several Standard cameras /Fisheye	
			IV.C.5 Range Systems	ToF laser sensor	
				One/ several range cameras	
			IV.C.6 Combination of Vision and Range systems	Standard CCD and range cameras	
			IV.C.7 RGB-D Devices	One/ seveal RGB-D devices	

2.3.2. Separating Human and Robot workspaces

Common mechanical robots are enormous, heavy and move at high speeds. These conditions make it important to forestall impacts between the robot and the humans who may enter the robot workspace, in order to avoid harm to the human. The methodology endorsed by the norm ISO 10218:1992 or its equivalent UNE-EN 775 is set to prevent such crashes or different occurrences that may result in any harm, this norm established a mandatory separation among human and robot workspaces, by distinguishing human interruptions in robot workspaces, and changing the robot behavior.

In view of these restrictions, when an intrusion comes in the robot workspace the robot speed is diminished in relation to the distinguished danger level, with the robot halting is the highest one. Three degrees of risk detection are proposed alongside control methodologies, passive and active safety devices. Such devices incorporate for example acoustic signs, proximity sensors, pressure mats, and ultrasonic sensors.

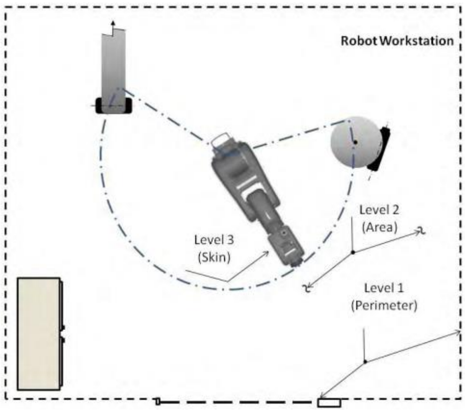


Figure 9 Separating human-robot workspace. (S. ROBLA-GÓMEZ 1, 2017)

2.3.3. Shared human and robot work/workspaces

Collaborative tasks involving human and robot make it necessary to remove the separating elements between them, and therefore new risks emerge that need to be addressed. In the following sections the main approaches to mitigate these risks are presented, including the quantification level of injury by collision. The information about the consequences to the human body of having a collision with a robot is key in taking the necessary steps to minimize injuries to the human and can be used for testing new robot safety systems.

2.3.3.1. Quantifying level of injury by collision

Focusing on systems whose principal aim is to enable safe human robot collaboration, several researchers have analyzed the consequences of human-robot collisions on the human body. This question may be approached from two different points of view. The first one is to estimate the pain tolerance, and the second one is to quantify the level of injury following a collision.

2.3.3.2. Minimizing injury in human-robot collision.

As in some cases a robot-human collision during the execution of collaborative tasks can be unavoidable, an important line of research focusses on the minimization of injuries in humans caused by such collisions. The methods that have been proposed to reduce the effects of collisions can be broadly classified as mechanical compliance systems and safety strategies involving collision/contact detection.

2.3.3.3. Collision avoidance

Although minimizing injuries in case of human-robot collision is very important, the prevention of impacts between robot and human is highly desirable. Therefore, a second key aim in human robot collaboration is to enhance safety through the implementation of collision avoidance systems. For this purpose, several solutions have been tried, which may be in many cases complementary to the previously discussed safety systems.

Chapter 3: Methodology

3.1. Hardware choice

For intelligent robots, tabletop object manipulation is quite possibly the most widely recognized task. It joins the capacities of the robot in vision, image procession, object recognition, hand arm manipulation, however, the real indoor environment is much more complicated than experimental scenarios. The vision of the robot sometimes can hardly provide enough information for successfully executing some difficult tasks, such as pick, place or assemble some small objects (Haiyang Jin, 2016) .In this case the implementing of the human gestures is a problem solving idea.

Interpretation of human gestures by a computer is used for human-machine interaction in the area of computer vision. The main purpose of gesture recognition research is to identify a particular human gesture and convey information to the user pertaining to individual gesture. From the corpus of gestures, specific gesture of interest can be identified, and on the basis of that, specific command for execution of action can be given to robotic system. Overall aim is to make the computer to understand human body language, thereby bridging the gap between machine and human. Hand gesture recognition can be used to enhance human–computer interaction without depending on traditional input devices such as keyboard and mouse.

Hand gestures are extensively used for telerobotic control and applications. Robotic systems can be controlled naturally and intuitively with such telerobotic communication. A prominent benefit of such a system is that it presents a natural way to send geometrical information to the robot such as: left, right. Robotic hand can be controlled remotely by hand gestures. Several approaches have been developed for sensing hand movements and corresponding by controlling robotic hand.

For hand gesture recognition, a highly efficient way is using data glove that can record the motion of each finger; some kinds of data glove can even measure the contact force of a grasping or pinching action. However, beside the high cost of data glove, they lack the capability to track position of the hand and even thought the glove-based gestural interfaces give more precision, it limits freedom as it requires users to wear cumbersome patch of devices.

Therefore, extra approaches are added to track hand positions, such as inferred optical tracking, which also increases the complexity of the system.

In this work we decided to only use the vision based method for both the hand tracking and gesture recognition as it lowers the costs and the hardware complexity significantly. But the performance of the gesture recognition is much effected by the lighting and background conditions. Thus, some aiding methods like skin color and pure color background are used to improve the recognition accuracy.

Other scholars use RGB-D data from Kinect for gesture recognition. However, the Kinect sensor is developed for body motion tracking, in the research of Kim et al., it has been proved that the accuracy of hand motion tracking using Kinect is much lower than LeapMotion sensor, which is particularly designed for hand motion tracking. (Yonjae Kim, 2014)

"The LeapMotion sensor, developed by Leap Motion Inc., is a new non-contact finger/hand tracking sensor. It has a high tracking accuracy and provides plenty of software interface for pose and gesture recognition. Some preliminary studies have been carried out for robot manipulation. Zubrycki et al. use a LeapMotion sensor to control a 3-finger gripper, GuerreroRincon et al. developed an interface to control a robotic arm, Marin et al. report the first attempt to detect gestures from the data combination of LeapMotion and Kinect. These use single LeapMotion for hand tracking and gesture recognition, however, due to the occlusion problem between fingers, single sensor can perform well only when the palm is with an ideal orientation." (Haiyang Jin, 2016)

This solution offers using multiple LeapMotion sensors, but I found a better solution opting toward a different type of system as difficulties of accuracy and precision were faced.

To overcome these difficulties a multi-sensors hand tracking system is developed to overcome the limitation of the aforementioned drawback of a single LeapMotion. The tracking space and working area are analyzed to gain an appropriate setup for one LeapMotion sensor alongside an Intel Realsense Camera. With self-registration, a coordinate system are established. Based on the definition of the element actions, an algorithm to calibrate the delay and combine the data from both 3D sensors is proposed to improve the stability for both the hand tracking and gesture recognition. To develop an operative demonstration system, a 6-DoFs (Degree of Freedoms) robotic arm with a 2-finger gripper are combined with the development of a 3D Cameras hand tracking system in ROS (Robot Operation System). Functional experiments are performed to indicate the results of combined hand tracking and gesture recognition. At the end, a scenario experiment is performed to show how this proposed system is used in a robotic system.

3.1.1. Leap Motion Camera

In the last few years, different optical sensors, which allow the acquisition of 3D objects, have been developed. Concurrently with the appearance of the new sensors, the number of potential applications vastly increases. Applications benefit especially from the increasing accuracy and robustness of 3D sensors and a drop in prices. Applications for 3D sensors include industrial tasks and many others.

Object tracking, motion analysis, character animation, 3D scene reconstruction and gesture-based user interfaces. These applications have different requirements in terms of resolution, speed, distance and target characteristics. Particularly with regard to gesture-based user interfaces, the accuracy of the sensor is a challenging task. Consumer-grade sensors offer only limited positioning accuracy. An analysis of the sensor shows a standard deviation in depth accuracy of approximately 1.5cm. The evaluation of the accuracy of optical sensors is the subject of current research and scientific discussion. The Leap Motion controller introduces a new gesture and position tracking system with sub-millimeter accuracy. In contrast to standard multi-touch solutions, this above-surface sensor is discussed for use in realistic stereo 3D interaction systems, especially concerning direct selection of stereoscopically displayed objects.

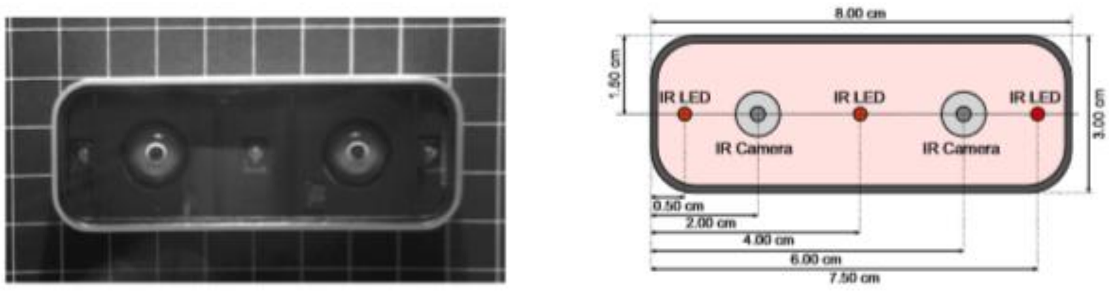


Figure 10 Schematic View of Leap Motion Controller. (FrankWeichert, 2013)*

The Leap Motion controller in conjunction with the current API (Application Programmer Interface) delivers positions in Cartesian space of predefined objects like finger tips, pen tip, etc. The delivered positions are relative to the Leap Motion controller's center point, which is located at the position of the second, centered infrared emitter. As illustrated in Figure 1, the controller consists of three IR (Infrared Light) emitters and two IR cameras. Hence, the Leap Motion can be categorized into optical tracking systems based on Stereo Vision. Because of the missing point cloud of the scene and the predefined detectable objects, traditional calibration techniques are not suitable for the LeapMotion. Nevertheless, a precise reference system is needed in order to evaluate the accuracy and repeatability of the Leap Motion controller. Industrial Robots support the ability of fixing different tools to their TCP (Tool Control Point) and exhibit high precision in sub-millimeter range. Consequently, industrial robots can act as high precision reference systems during the evaluation of the Leap Motion.

The robot cell builds the metrology system of the mandatory measurements. The analyzed parameters related to the sensors are accuracy and repeatability. Accuracy is the ability of a 3D sensor to determine a desired position in 3D space. Repeatability is the ability of a sensor to locate the same position in every measurement repetition. The analysis of the accuracy and repeatability tests was performed in accordance to ISO 9283 standard, which is primarily used for industrial robots.

According to the paper mentioned in the bibliography the LeapMotion sensor went through many experiments to assure its accuracy, Robustness and precision.

The following tests were conducted considering the metrology calibration approaches:

- Positioning test probe methods (static cases)
- Path drawing methods (dynamic cases)

The basic test cases focus on the evaluation of the accuracy of the reference pen tip moving to positions on a regular grid of a plane (xy-, xz- and yz-plane) and moving to discrete positions on a path, for example along the particular axes (x-, y- and z-axis) of the sensor coordinate system and on a sinus function within the xy-plane, as illustrated in Figure2.

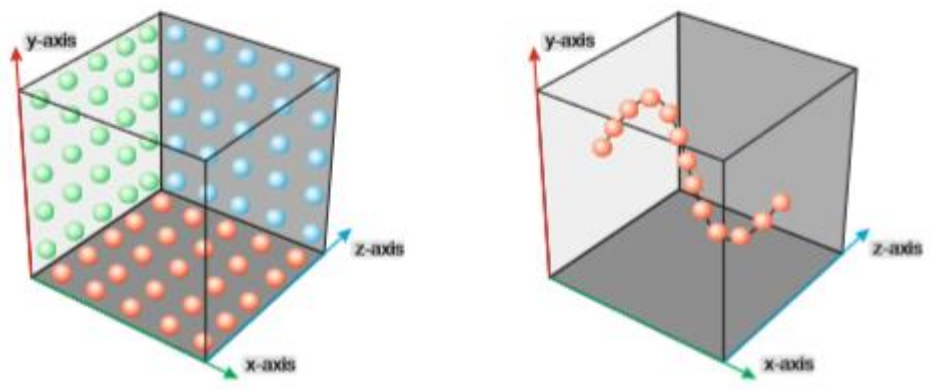


Figure 11 Visualization of Sinus Function within the x-y plane (FrankWeichert, 2013)*

3.1.1.1. Static test cases:

The particular diagrams visualize the deviation between the desired static Cartesian position and the measured positions relative to the xy-plane (a), xz-plane (b) and yz-plane (c).

Independent from the axis, the deviation between the desired position and the measured positions is less than 0.20mm. In the case of the x-axis the deviation is less than 0.17mm.

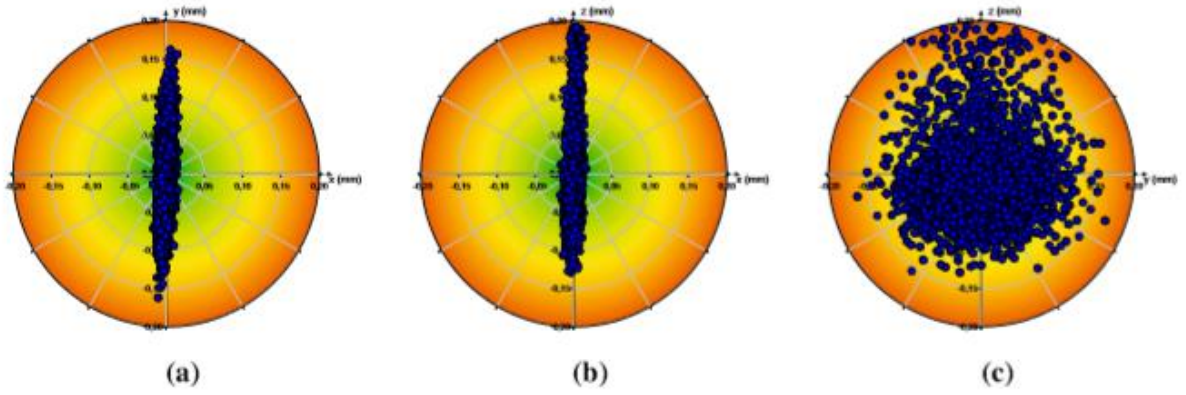


Figure 12 eviation between a desired 3D Position and the Measured Positions for a Static Position. (a)xy-Variation;(b)xz-Variation; (c) yz-Variation. (FrankWeichert, 2013)*

3.1.1.2. Dynamic Test Cases:

Next, the dynamic scenarios are analyzed by positioning the reference pen tip in different positions using the robot. Figure 13 illustrates the analysis of the accuracy when positioning the reference tip on different positions on a regular grid in the xy-, xz- and yz-plane by the robot. The diagrams show the deviation between a desired 3D position and the median of the corresponding measured positions respectively. Independent from the axis the deviation is below 1mm and on average under 0.4mm.

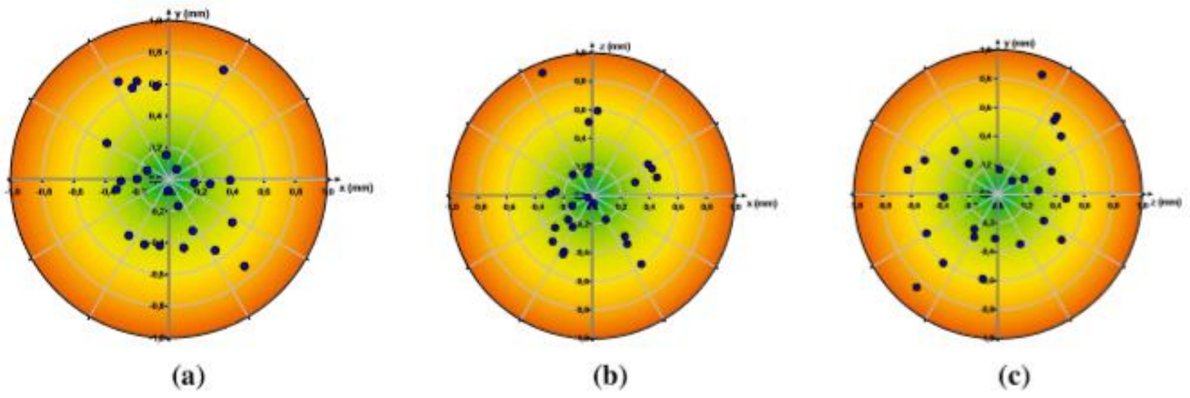


Figure 13 Deviation between a Desired 3D Position and the Median of the Measured Positions. (a) xy-Plane; (b) xz-Plane; (c) yz-Plane. (FrankWeichert, 2013)*

As the results of these experiments were within our margin of acceptance we decided to opt for this Camera as we found that its capabilities suits the tasks and objectives in our work.

3.1.2. Intel Realsense D415

The Intel RealSense Depth Camera has been designed to equip devices with the ability to see, understand, interact with, and learn from their environment. The D415 features the Intel RealSense Vision D4 processor with a high-resolution depth (up to 1280 × 720 pixels at 30 frames per second), long-range capability (up to about 10 m), rolling shutter technology and, as noted, a narrow field of view ideal for precise measurements. The RealSense Vision D4 is a vision processor based on 28-nanometer (nm) process technology for real-time calculation of stereo depth data. The device has a very compact depth camera (dimensions: 99 × 20 × 23 mm3, weight: 72 g) that can either be integrated into computers and mobile devices or used as a self-standing device. Moreover, it comes with a color camera and a depth camera system, comprising two IR cameras and an IR projector.

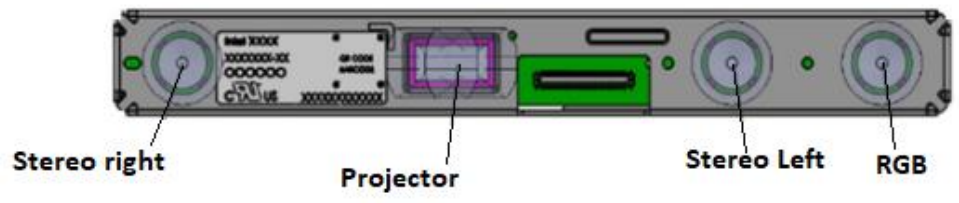


Figure 14 RealSense D415 (Monica Carfagni, 2018)

The infrared projector improves the ability of the stereo camera system to determine depth by projecting a static infrared pattern on the scene to increase the texture of low-texture scenes. The D415 is equipped with a color camera with a resolution up to 1920 × 1080 pixels, and provides texture information to be superposed on the depth data. The spatial resolution of the depth map of the Intel RealSense D415 is up to HD (1280 × 720 pixels), in a working depth range declared by the manufacturer equal to ~160–10,000 mm. The camera works at different pixel resolutions corresponding to different minimum depth values, point densities and framed areas.

Table 2 Technical specifications of Intel RealSense D415.

Environment	**Indoor and Outdoor**
Depth Technology	Active Infrared(IR) stereo
Image Sensor Technology	Rolling shutter : 1.4μm x 1.4μm pixel size
Depth of view (Horizontal x Vertical)	69.4 x 42.5° (+-3°)
Depth stream output Resolution	Up to 1280 x720 pixels
Depth Stream Output Frame rate	Up to 90 fps
Minimum Depth Distance	0.16m
Maximum range	~10m
RGB sensor Resolution and frame Rate	Up to 1920 x 1080 pixels at 30 fps
RGB Sensor Field of View (Horizontal x Vertical)	69.4 x 42.5° (+-3°)
Camera dimension (Length x Depth x Height)	99mm x 20 mm x 23mm
Connector	USB type C

Open library RealSense SDK 2.0 has standard functions for camera initialization, parameters setting, functions and methods for reading frames from the video stream, calculating the distance from the hand to the depth camera, RGB images and depth maps saving methods. It is possible to modify the algorithms available in the source code of the RealSence SDK 2.0. The methods and functions from RealSence SDK 2.0 were used to implement gestures image. Some additional functions for gesture capture were coded by the authors.

3.2. Hand Detection

The LeapMotion detector provides a high accuracy system to detect each hand. It's based on an infra-red detector and is able to detect the hands in a half sphere centered on the Leap Motion of about 30 cm in radius (in reality it is not a real half sphere because the angle of view of the Leap Motion is 150 degrees). Using the SDK from the Leapmotion made this task relatively simpler than the Realsense camera.

As for the realsense camera we had to develop a system to perform a complete process from pre-processing to detection & tracking to feature extraction and finally training and testing by HMM algorithm. For such task we opted toward OpenCV which is a library that made the system easy to create due to the large amount of inbuilt functions of various image processing tasks like edge detection, feature tracking etc.

First, we have Detect and track the hand. So we decided to create a combination of two methods for hand localization the Thresholding and skin color segmentation.

3.2.1. Thresholding

To detect the moving object region, we applied the thresholding on the frame to detect the possible moving area in a complex background.

$$D_i(x,y) = \begin{Bmatrix} 1, |(x,y) - F_{i+1}(x,y)|\ threshold \\ 0, otherwise \end{Bmatrix}$$

3.2.2. Skin color segmentation

The second method is the skin color segmentation which can be detected by color constraint. Skin color is usually detected using HSV (Hue Saturation Value) color space.

Using these two methods we were able to get hand movement information. Afterwards we used 'AND' logic to combine both extracted information together.

$$C_i(x,y) = D(x,y) and\ E_i(x,y)$$

This method is divided in several steps, in the first step the image is passed through baseline CNN (Convolutional Neural Network) to extract the feature maps of the input. Next the feature map is then processed in a multi stage CNN pipeline to create a Confidence Map and a Part Affinity Field. In the last step, the Confidence Maps and Part Affinity Fields that are generated are processed by a matching algorithm to obtain the poses for each person in the image.

A Confidence map is a 2D representation on where the body part can be located in any given pixel while a Part Affinity Fields is a set of 2D vector fields that encodes location and orientation of limbs of different people in the image.

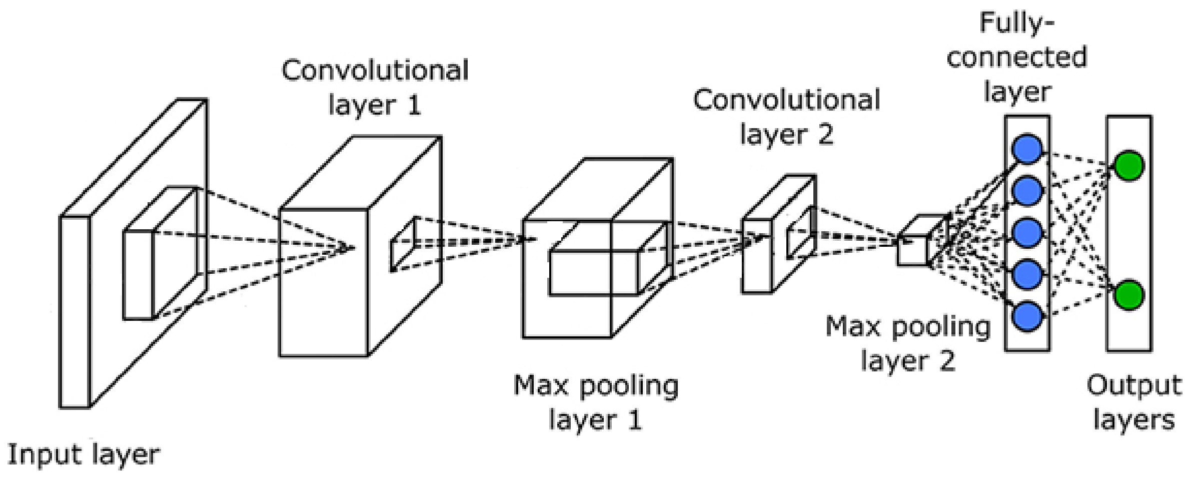


Figure 15 Convolutional Neural Network (CNN) (LeCun, 1998)

3.3. Finger Detection

Finger detection is split between static gestures and dynamic gestures. Features for static gestures are mainly built based on palm and fingers relative distances. We calculated two types of distances. One type is distances between fingertips Fpos and palm center Ppos. The other type is distances between two fingers which are adjacent. For example distance between thumb and index, distance between index and middle denoted the figure below shows examples of static gestures we could effectively recognize.

The other gesture features are built based on distances between fingers and palms. The distance between thumb and index is used to identify the OK gestures. The distance between index and middle finger is used to distinguish V gesture and Index and Middle pointing gesture. The rest gestures simply combined these two standard gestures. For example the index L gestures on the top most in the figure below are index and thumb extended and the rest fingers bent.

As for the dynamic gestures, features are easily distinguished from static gestures features. We calculate the total value of velocity magnitude among fingers and palm. If the total movement value is greater than a user-defined threshold, we believe the hand is moving. Otherwise, we starts to recognize the static hand gestures. Dynamic Gesture features mainly use the velocity of fingertips and palm to detect the movement patterns. Compared with the static gestures, dynamic gestures are much more complicated. We starts from the global movement and then go through the details of the fingers' movement. From the global movement, we try to detect hand translation movement, hand rotation movement, hand circle movement. Then we consider the fingers' movement. Since there are so many possible movement and will focus on the movement of index finger which is very useful in communication and interactions.

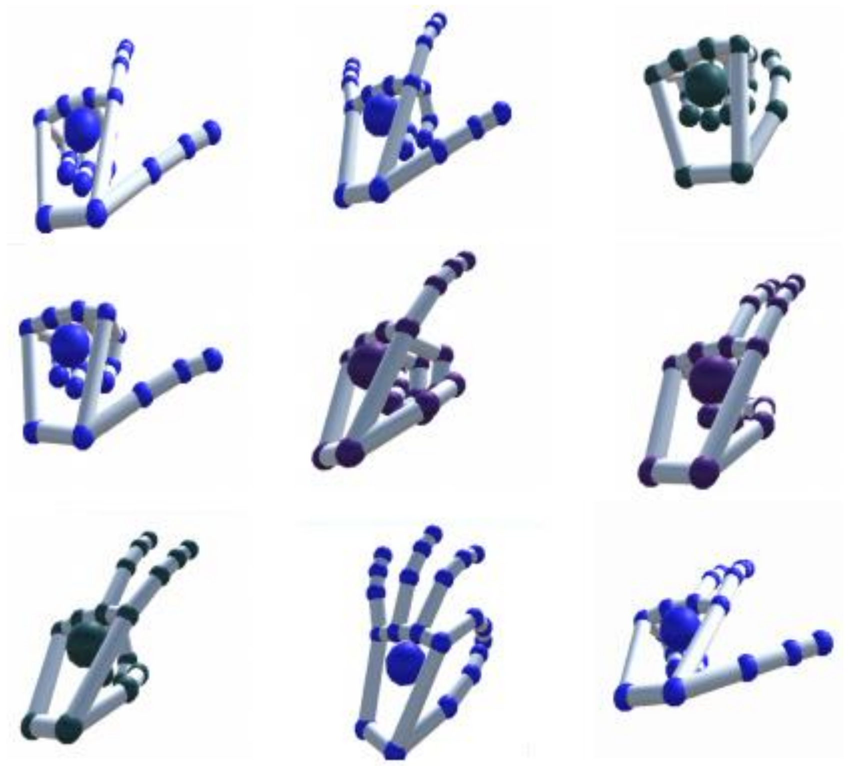


Figure 16 Examples of static gestures

Also for dynamic gestures, we have other features as hand translation which indicates fingers and palm are moving together straightly without rotation. We calculate the cross correlation of velocity vectors between fingers Fv and palm Pv for all fingers. If the absolute values of these cross correlations are greater than 0.95, we recognize that the hands are moving straightly. Also Hand rotation which contains two parts. One is the difference of current palm normal Pn and previous palm normal Pn-1. The other parts is the angle between difference of current palm and hand direction PD. We then calculate the cross correlation of the current palm and the hand direction. As for Hand circle it indicates the palm is drawing a great circle. Same to hand rotation features, we calculate the first order difference between palm normals.

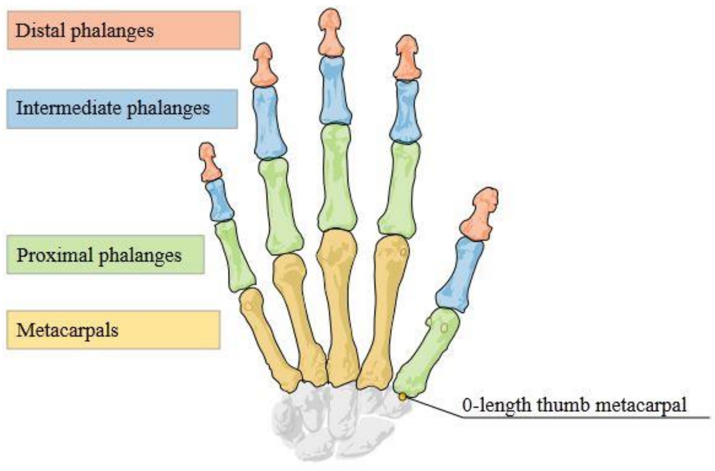


Figure 17 Hand Skeleton (Haiyang Jin, 2016)

Skeletal Tracking Model is a standard hand model provided by Leap Motion. It simulates the actual hand skeleton.

3.4. Software

As for Software we used Python as our main programming language for both LeapMotion and Realsense operating codes also we used Unity software to elaborate an Interface of command.

3.4.1. Python

Python is a high-level and general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

3.4.2. Python Idle

IDLE (Integrated Development and Learning Environment) is an integrated development environment (IDE) for Python. . IDLE can be used to execute a single statement just like Python Shell and also to create, modify, and execute Python scripts.

3.4.3. Unity

Unity is a cross-platform game engine developed by Unity Technologies, the engine is able to support more than 25 platforms also it can be used to create three-dimensional, two-dimensional, virtual reality, and augmented reality software, as well as simulations and other experiences. The engine has been adopted by industries outside video gaming, such as automotive, architecture, engineering and construction.

Unity is originally a game engine development software, but we saw the potential of this software to create our Commanding interface, as it provide interesting assets for the LeapMotion camera which makes coding simpler and smoother in comparison to other software.

3.4.4. C# Programming language

C# is a general-purpose, multi-paradigm programming language encompassing static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented, and component-oriented programming disciplines. We used C# as our programming language for the Unity Interface.

3.4.5. Visual Studio

To ensure the functionality of our interface and to increase the functions provided by it we wrote our scripts in Visual Studio platform in C# language.

> Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Chapter4: System Architecture

In this chapter we will present the overall architecture of our system, and explain each block.

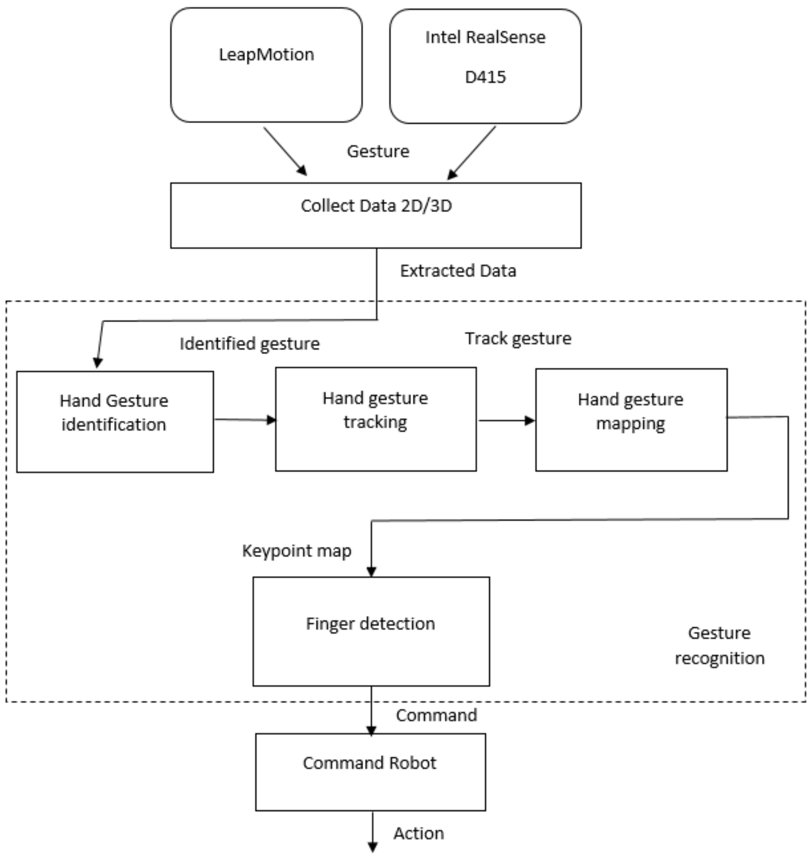


Figure 18 Overall system architecture

Our two inputs are the LeapMotion sensor and the Intel RealSense camera D415, they will provide 2D/3D data, this extracted data will be identified for a hand gesture, track it for a dynamic gesture and generate a keypoint map, this map will allow us to detect the fingers to be able to command the Robot.

4.1. Leap Motion operation for gesture detection

For Leap motion camera, the detection of hand gestures is done through the Leap motion SDK which has multiple functions and methods for hand tracking allowing the detection of gestures like circles, tapping, swiping and many others. These functions have their own library files, the System Development Kit for Leap motion camera is very rich and made it relatively simple to manipulate these functions optimally as the recognition system is built in the SDK allowing an optimal and relatively precise detection. To tap further into this system, we have to detect fingers, so we have to recognize the fingers in a hand on two hands and the motion of these fingers which is the main point in using such technology, luckily the SDK offers other functions allowing the recognition of each finger in each hand.

In our work, we developed a code to firstly recognize each hand named ‘left’ and ‘right’, this is done through a method that detect the hand type measuring the ergonometric in each hand by measuring the position of the thumb relatively to the other fingers, such method will help us know if we are using left or right hand in the frame. Next we had to distinguish each finger on each hand and name them, we made an array of fingers names and allocated each finger to its name in the array, and by doing this we made things relatively easier to manipulate each finger on its own, we used a built in method which uses machine learning to recognize each finger. To perform the tasks above we developed a python code importing all the required libraries in the SDK.

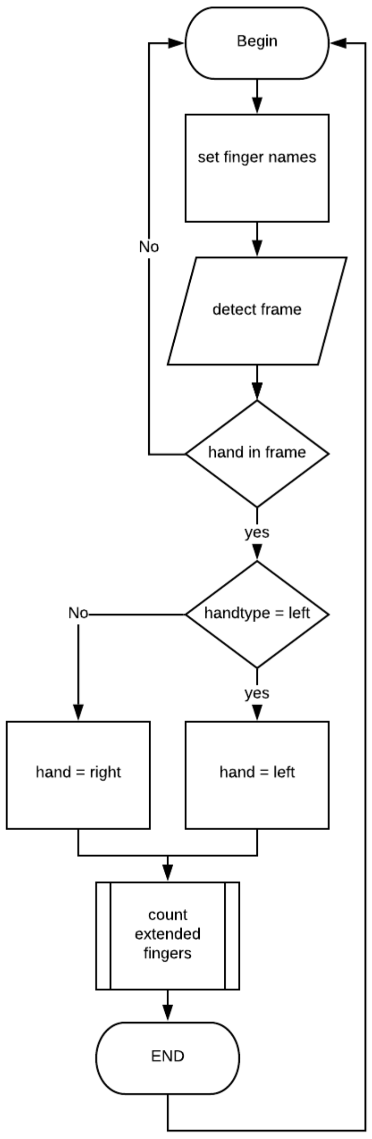


Figure 19 Flowchart Leapmotion

```python
import Leap, sys, thread, time
from Leap import CircleGesture, KeyTapGesture, ScreenTapGesture, SwipeGesture
class LeapMotionListener(Leap.Listener):
    finger_names = ['Thumb', 'Index', 'Middle', 'Ring', 'Pinky']
    bone_names = ['Metacarpal', 'Proximal', 'Intermediate', 'Distal']
    state_names = ['STATE_INVALID', 'STATE_START', 'STATE_UPDATE', 'STATE_END']

    def on_init(self,controller):
        print("Motion Sensor Connected !")

        controller.enable_gesture(Leap.Gesture.TYPE_CIRCLE);
        controller.enable_gesture(Leap.Gesture.TYPE_KEY_TAP);
        controller.enable_gesture(Leap.Gesture.TYPE_SCREEN_TAP);
        controller.enable_gesture(Leap.Gesture.TYPE_SWIPE);

    def on_disconnect(self, controller):
        print("Motion Sensor Disconnected !")

    def on_exit(self, controller):
        print("Exit")

    def on_frame(self, controller):
        frame = controller.frame()
        for hand in frame.hands:
            handType = "Left Hand" if hand.is_left else "Right Hand"
            extended_finger_list = frame.fingers.extended()
            print(str(handType) + "  # fingers : " + str(len(extended_finger_list)))
    def main():

        listener = LeapMotionListener()
        controller = Leap.Controller()

        controller.add_listener(listener)

        print("Press enter to quit")
        try:
            sys.stdin.readline()
        except KeyboardInterrupt:
            pass
        finally:
            controller.remove_listener(listener)
    if __name__ == "__main__":
        main()
```

To allow the robot to recognize the hand patterns we used hand pattern examples and we assigned each pattern to an action, we chose the number of fingers as the main pattern distinguish for this task varying from 0 to 5 allowing multiple finger combinations. This task is done by calculating the angle of each finger to know which ones are the extended ones.

4.2. Real Sense operation for gesture detection

For Real Sense the gesture recognition is more complicated as the developers didn't come with a specific library to perform such tasks, so we opted toward OpenCV (Open Source Computer Vision Library) which is an open source computer vision and machine learning software library.

First we made sure to eliminate as much as noise in the taken image by saturating the colors of the skin to make it distinguishable from the background, second we detected the contours of the hand which is very important for the calculation of distance and angles later on.

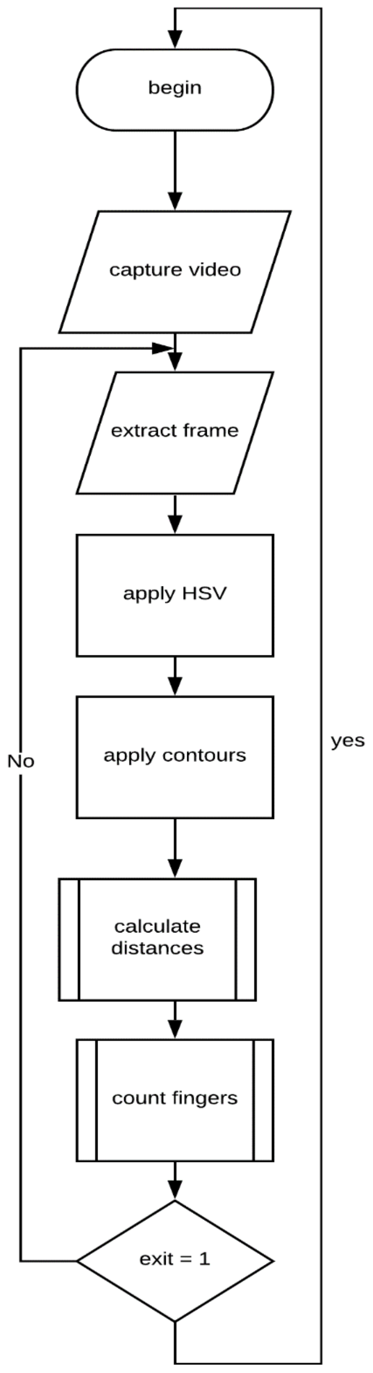


Figure 20 Flowchart RealSense Hand Gesture

```
import cv2

vidcap = cv2.VideoCapture('video.mp4')
def getFrame(sec):

  vidcap.set(cv2.CAP_PROP_POS_MSEC, sec*1000)
  hasFrames,image = vidcap.read()
  if hasFrames:
        cv2.imwrite("image"+str(count)+".jpg",image)
  return hasFrames

sec = 0
FrameRate = 0.5
count=1
success = getFrame(sec)
while success:
        count = count + 1
        sec = sec + frameRate
        sec = round(sec,2)
        success = getFrame(sec)
```

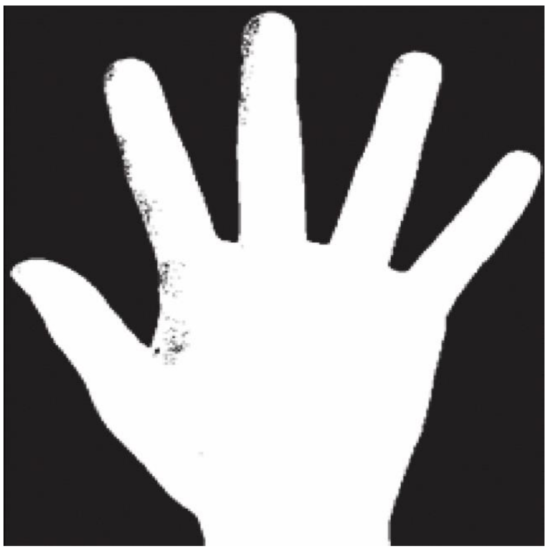


Figure 21 Hand detection using HSV

To perform these methods above we need first to train the system using an estimation method, we used ‘OpenPose’ libraries which is a real-time multi-person system to jointly detect human body, hand, facial, and foot key-points (in total 135 key-points) on single images.

‘OpenPose’ libraries made us integrate these algorithms easier, first we train the system using different hand position maps. On each hand image we mapped the position of joints up to 20 joints. In that way the system will estimate the position of hand joints in real-time.

Next we made a Python algorithm that takes a video, detect the location of each hand in the frame and estimate the position of joints on each hand numbering them and generating a skeleton map which made it easier to locate each point to further calculate the distance between each point, calculate angles, directions in a 2D or a 3D environment.

We considered each pair of points as a joint, and by calculating the distance between each point we made sure to distinguish the extended fingers from the not extended ones which allowed us to count the number of extended fingers and generating a both skeleton and keypoint maps saving them as .jpg file for further calculation.

This method made us not only count the number of extended fingers but also detect which finger exactly is extended and organizing them in an array table.

```
from __future__ import division
import cv2
import time
import numpy as np

def x_axis(s):
        lis = s.split(',')
        temp = str(lis[0])
        lis0 =  temp.split('(')
        temp0 = lis0[1]
        return temp0


def y_axis(s):
        lis = s.split(',')
        lis0 =  str(lis[1]).split(')')
        return lis0[0]

def distance(x1,x2):
        p = int(x2) - int(x1)
        return p



protoFile = "hand/pose_deploy.prototxt"
weightsFile = "hand/pose_iter_102000.caffemodel"
nPoints = 22
POSE_PAIRS = [ [0,1],[1,2],[2,3],[3,4],[0,5],[5,6],[6,7],[7,8],[0,9],[9,10],[10,11],[11,12],
               [0,13],[13,14],[14,15],[15,16],[0,17],[17,18],[18,19],[19,20] ]
net = cv2.dnn.readNetFromCaffe(protoFile, weightsFile)

frame = cv2.imread("image1.jpg")
frameCopy = np.copy(frame)
frameWidth = frame.shape[1]
frameHeight = frame.shape[0]
aspect_ratio = frameWidth/frameHeight

threshold = 0.1

t = time.time()
```

In the script above we used the open Pose “.caffemodel” to train our system and we saved every frame captured. Also numpy library helped us to further develop and manipulate our system.

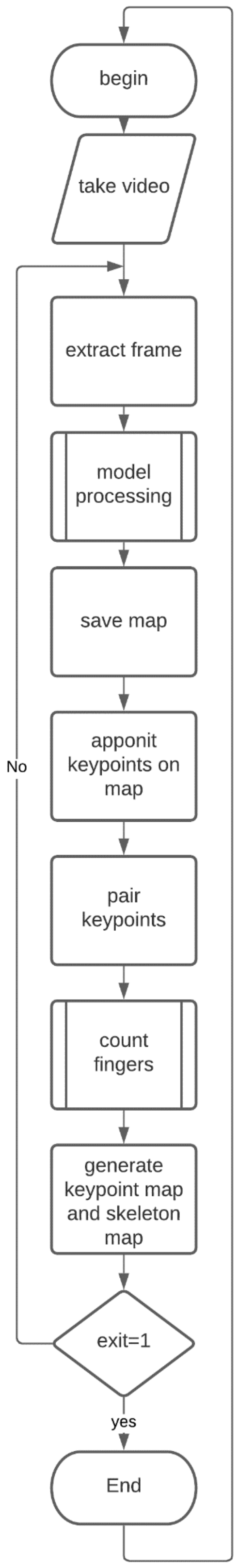


Figure 23 Flowchart RealSense using OpenPose

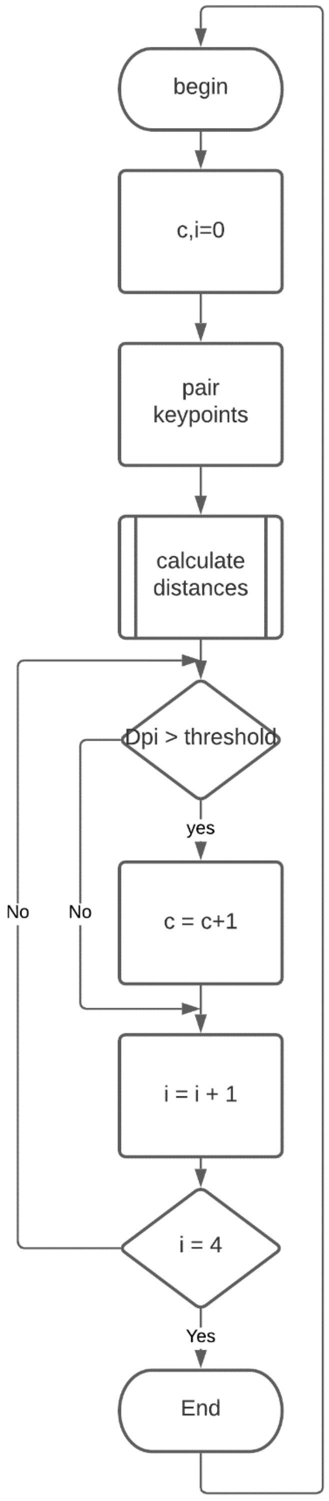


Figure 22 Flowchart finger counting

```
# input image dimensions for the network
inHeight = 368
inWidth = int(((aspect_ratio*inHeight)*8)//8)
inpBlob = cv2.dnn.blobFromImage(frame, 1.0 / 255, (inWidth, inHeight),
                                (0, 0, 0),swapRB=False, crop=False)

net.setInput(inpBlob)
output = net.forward()
# Empty list to store the detected keypoints
points = []

for i in range(nPoints):
    # confidence map of corresponding body's part.
    probMap = output[0, i, :, :]
    probMap = cv2.resize(probMap, (frameWidth, frameHeight))
    minVal, prob, minLoc, point = cv2.minMaxLoc(probMap)

    if prob > threshold :
        cv2.circle(frameCopy, (int(point[0]), int(point[1])), 8, (0, 255, 255),
                   thickness=-1, lineType=cv2.FILLED)
        cv2.putText(frameCopy, "{}".format(i), (int(point[0]), int(point[1])),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2, lineType=cv2.LINE_AA)

        # Add the point to the list if the probability is greater than the threshold
        points.append((int(point[0]), int(point[1])))
    else :
        points.append(None)
```

Next we modify the frame that we captured and detect the points using OpenCV machine learning methods.

```
# Draw Skeleton
fingers = [0,0,0,0,0]
for pair in POSE_PAIRS:
    partA = pair[0]
    partB = pair[1]
    f = 0

    if points[partA] and points[partB]:
        cv2.line(frame, points[partA], points[partB], (0, 255, 255), 2)
        cv2.circle(frame, points[partA], 8, (0, 0, 255), thickness=-1, lineType=cv2.FILLED)
        cv2.circle(frame, points[partB], 8, (0, 0, 255), thickness=-1, lineType=cv2.FILLED)
        if(partB == 12): fingers[2] = "middle"
        if(partB == 4): fingers[0] = "thumb"
        if(partB == 20): fingers[4] = "pinky"
        if(partB == 16): fingers[3] = "ring"
        if(partB == 8): fingers[1] = "index"

    if(fingers[0] == "thumb"): f = f+1
    if(fingers[2] == "middle"): f = f+1
    if(fingers[3] == "ring"): f = f+1
    if(fingers[1] == "index"): f = f+1
    if(fingers[4] == "pinky"): f = f+1
```

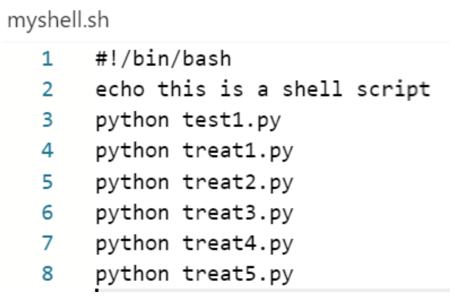
After that we assign each point to a pair of points as an array, next we distinguish each pair of points to determine which finger and assign each pair in the finger names array. This allowed us to count how many fingers are used by increasing the f counter each time we detect the pair of points assigned to one finger.

```
print(f)
pos_8 = x_axis(str(points[8]))
pos_5 = x_axis(str(points[5]))
d_index = distance(pos_8 , pos_5)
pos_12 = x_axis(str(points[12]))
pos_10 = x_axis(str(points[10]))
d_midd = distance(pos_12 , pos_10)
pos_16 = x_axis(str(points[16]))
pos_14 = x_axis(str(points[14]))
d_ring = distance(pos_16 , pos_14)
pos_20 = x_axis(str(points[20]))
pos_18 = x_axis(str(points[18]))
d_pinky = distance(pos_20 , pos_18)


print(d_index)
print(d_midd)
print(d_ring)
print(d_pinky)
print(calc())

cv2.putText(frame, str(f),(50,50),cv2.FONT_HERSHEY_SIMPLEX ,1,(255, 0, 0), 2, cv2.LINE_AA)

cv2.imwrite('My_keypoints1.jpg', frameCopy)
cv2.imwrite('My_Skeleton1.jpg', frame)

cv2.waitKey(0)
```

Above we calculated the distance between each point to help us get more information before drawing the keypoint and the skeleton maps.

At this point we got our keypoint and skeleton maps, so the next step is the process these information but we ran into a problem that processing a whole video takes too much time, so we opted toward taking multiple images processing each one of them separately which made the system runs a lot faster and smoother.

Using a Shell script allowed us to run all the Python Codes at once.



```
myshell.sh
#!/bin/bash
echo this is a shell script
python test1.py
python treat1.py
python treat2.py
python treat3.py
python treat4.py
python treat5.py
```

Figure 24 Shell script

4.3. Communication method between scripts and the robot

To communicate with the robot we used unity interface, unity allowed us to have a functional multi-platform interface that works on Windows, IOS, Android and other platforms.

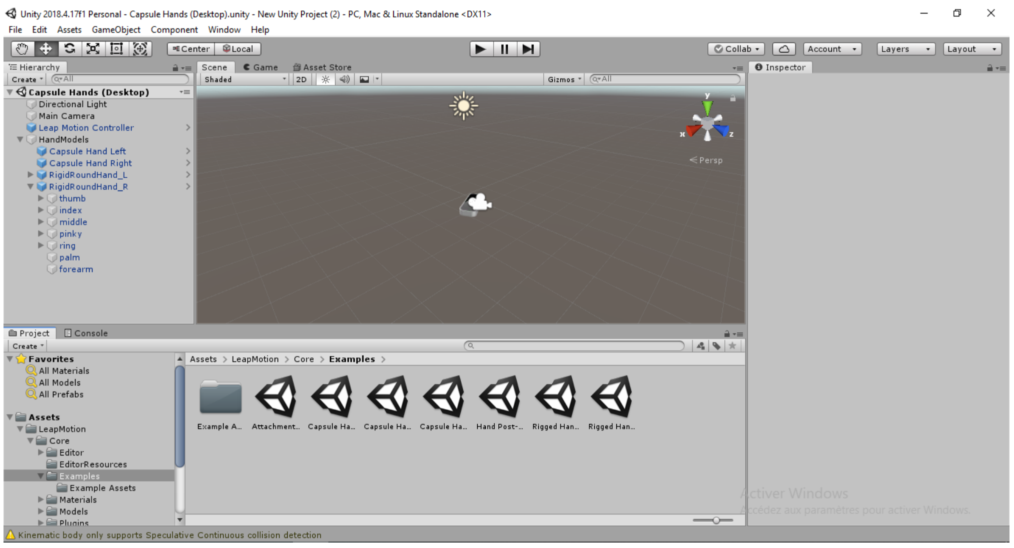


Figure 25 Unity designing interface

Leapmotion provided us with free graphical assets downloadable from their official website, these 3D graphical assets are based on the keypoints on each hand to create a 3D dynamic skeleton. We download the asset package and then we import it in the Project window, later we can drag in every model we need and attach our script to it. Every 3d model is composed of sub models which represent every bone in our hands.

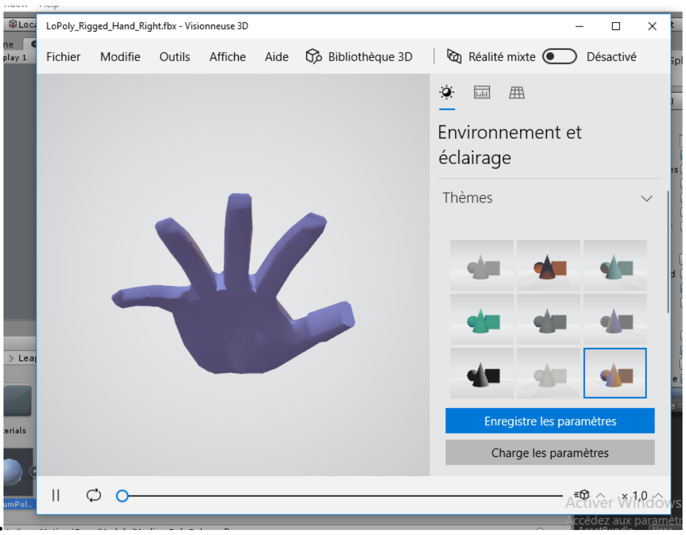
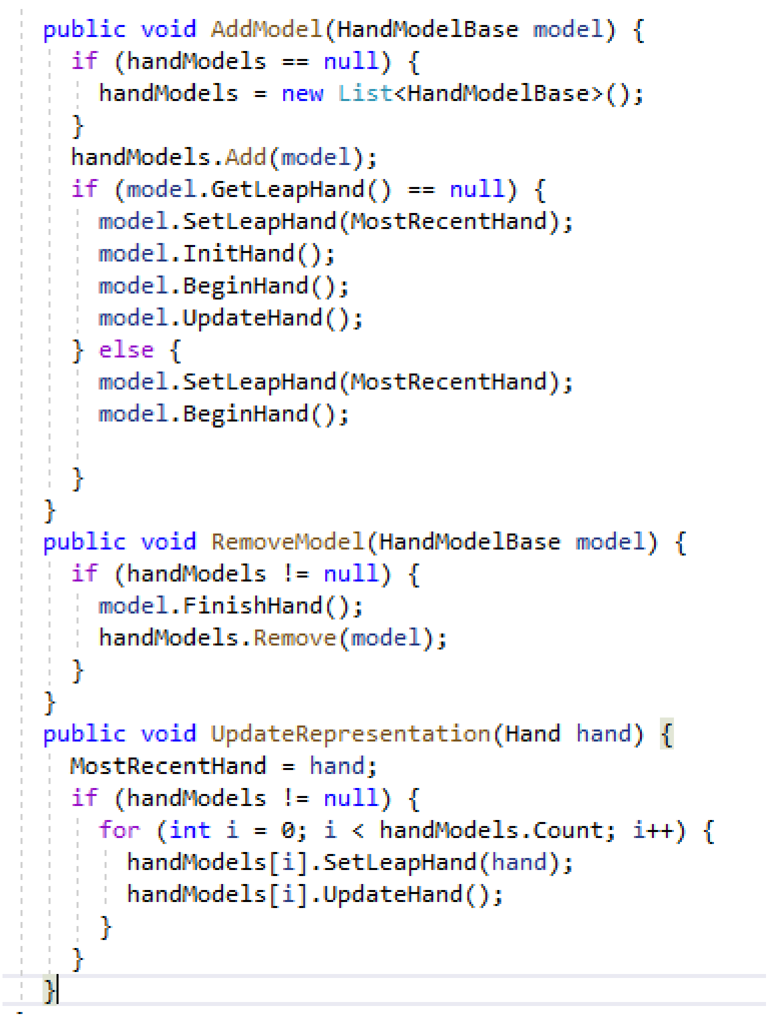

LoPoly_Rigged_Hand_Right.fbx - Visionneuse 3D
Fichier Modifie Outils Affiche Aide Bibliothèque 3D Réalité mixte Désactivé
Environnement et éclairage
Thèmes
Enregistre les paramètres
Charge les paramètres

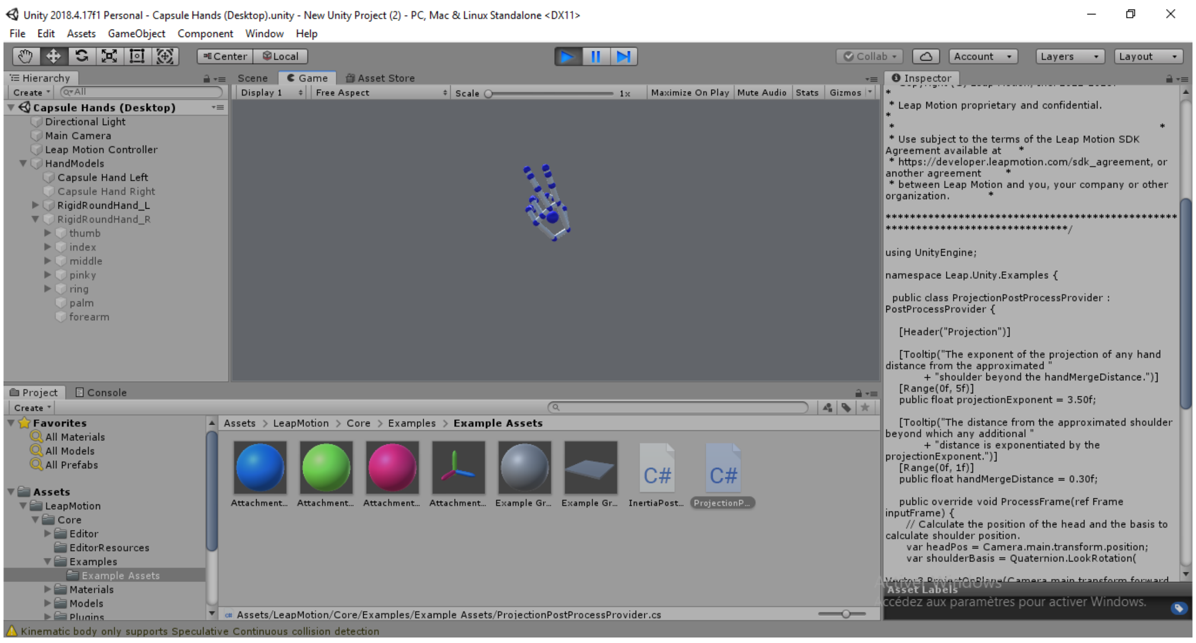
Figure 26 3d hand example from unity assets

These assets provided us with scripts to control the 3D graphics, these scripts are written in C# so we had to modify them to fulfill our requirements.



```
public void AddModel(HandModelBase model) {
  if (handModels == null) {
    handModels = new List<HandModelBase>();
  }
  handModels.Add(model);
  if (model.GetLeapHand() == null) {
    model.SetLeapHand(MostRecentHand);
    model.InitHand();
    model.BeginHand();
    model.UpdateHand();
  } else {
    model.SetLeapHand(MostRecentHand);
    model.BeginHand();

  }
}
public void RemoveModel(HandModelBase model) {
  if (handModels != null) {
    model.FinishHand();
    handModels.Remove(model);
  }
}
public void UpdateRepresentation(Hand hand) {
  MostRecentHand = hand;
  if (handModels != null) {
    for (int i = 0; i < handModels.Count; i++) {
      handModels[i].SetLeapHand(hand);
      handModels[i].UpdateHand();
    }
  }
}
```

the script above is written is C# in Visual studio platform to add the 3D models to the interface, link them to our controlling script, remove them if no hand is detected and update them every frame.

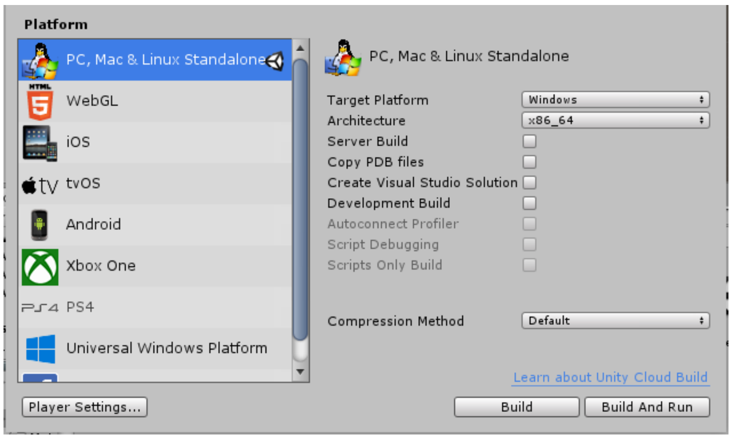

Unity 2018.4.17f1 Personal - Capsule Hands (Desktop).unity - New Unity Project (2) - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Window Help
Hierarchy
Capsule Hands (Desktop)
Directional Light
Main Camera
Leap Motion Controller
HandModels
Capsule Hand Left
Capsule Hand Right
RigidRoundHand_L
RigidRoundHand_R
thumb
index
middle
pinky
ring
palm
forearm
Scene
Game
Asset Store
Inspector
Project
Console
Favorites
All Materials
All Models
All Prefabs
Assets
LeapMotion
Core
Editor
EditorResources
Examples
Example Assets
Materials
Models
Plugins
Assets > LeapMotion > Core > Examples > Example Assets
Assets/LeapMotion/Core/Examples/Example Assets/ProjectionPostProcessProvider.cs
Kinematic body only supports Speculative Continuous collision detection

```
*
 * Leap Motion proprietary and confidential.
*
 *                                                  *
 * Use subject to the terms of the Leap Motion SDK
Agreement available at     *
 * https://developer.leapmotion.com/sdk_agreement, or
another agreement      *
 * between Leap Motion and you, your company or other
organization.        *

*************************************************
*****************************/

using UnityEngine;

namespace Leap.Unity.Examples {

  public class ProjectionPostProcessProvider :
PostProcessProvider {

    [Header("Projection")]

    [Tooltip("The exponent of the projection of any hand
distance from the approximated "
           + "shoulder beyond the handMergeDistance.")]
    [Range(0f, 5f)]
    public float projectionExponent = 3.50f;

    [Tooltip("The distance from the approximated shoulder
beyond which any additional "
           + "distance is exponentiated by the
projectionExponent.")]
    [Range(0f, 1f)]
    public float handMergeDistance = 0.30f;

    public override void ProcessFrame(ref Frame
inputFrame) {
      // Calculate the position of the head and the basis to
calculate shoulder position.
      var headPos = Camera.main.transform.position;
      var shoulderBasis = Quaternion.LookRotation(
```

Figure 27 simulation of hand sign

The script allows to track hand or hands and represent a 3D hand simulating gestures.

Unity allows us to export our application in different platforms using Build settings.


Platform
PC, Mac & Linux Standalone
WebGL
iOS
tvOS
Android
Xbox One
PS4
Universal Windows Platform
Learn about Unity Cloud Build
Player Settings...
Build
Build And Run

PC, Mac & Linux Standalone

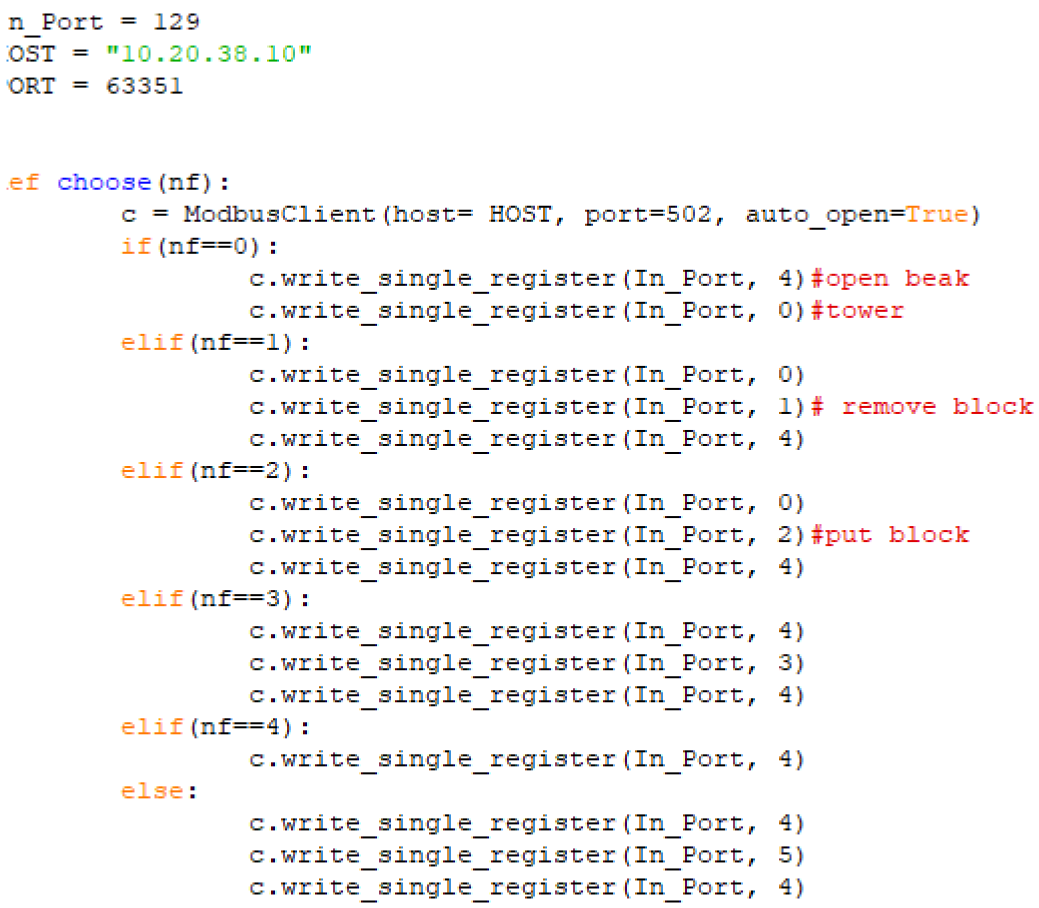
Target Platform	Windows
Architecture	x86_64
Server Build	☐
Copy PDB files	☐
Create Visual Studio Solution	☐
Development Build	☐
Autoconnect Profiler	☐
Script Debugging	☐
Scripts Only Build	☐
Compression Method	Default

Figure 28 Build settings

Next we had to communicate to the robot using our python script

```
import time
from pyModbusTCP.client import ModbusClient

In_Port = 129
HOST = "10.20.38.10"
PORT = 63351

c = ModbusClient(host= HOST, port=502, auto_open=True)
```

First we had to connect to the robot Host IP and the PORT using ModbusTCP library, then we assign each finger gesture to a different Robot action.



```
n_Port = 129
OST = "10.20.38.10"
ORT = 63351


ef choose(nf):
        c = ModbusClient(host= HOST, port=502, auto_open=True)
        if(nf==0):
                c.write_single_register(In_Port, 4)#open beak
                c.write_single_register(In_Port, 0)#tower
        elif(nf==1):
                c.write_single_register(In_Port, 0)
                c.write_single_register(In_Port, 1)# remove block
                c.write_single_register(In_Port, 4)
        elif(nf==2):
                c.write_single_register(In_Port, 0)
                c.write_single_register(In_Port, 2)#put block
                c.write_single_register(In_Port, 4)
        elif(nf==3):
                c.write_single_register(In_Port, 4)
                c.write_single_register(In_Port, 3)
                c.write_single_register(In_Port, 4)
        elif(nf==4):
                c.write_single_register(In_Port, 4)
        else:
                c.write_single_register(In_Port, 4)
                c.write_single_register(In_Port, 5)
                c.write_single_register(In_Port, 4)
```

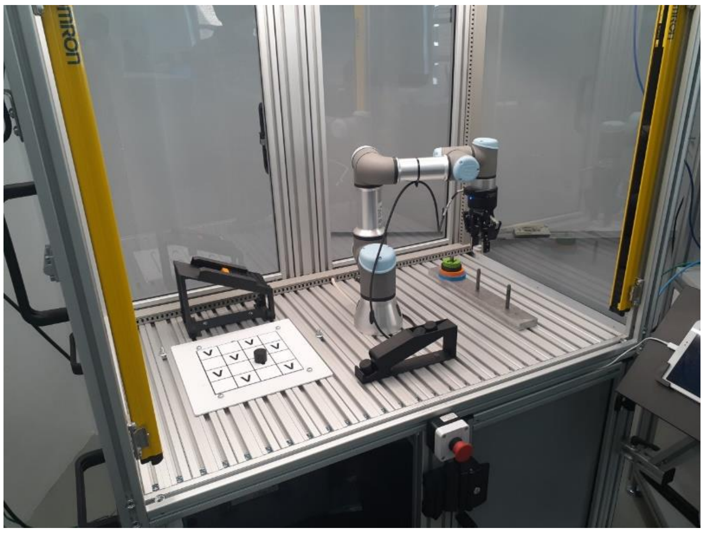


Figure 29 Robotic workbench

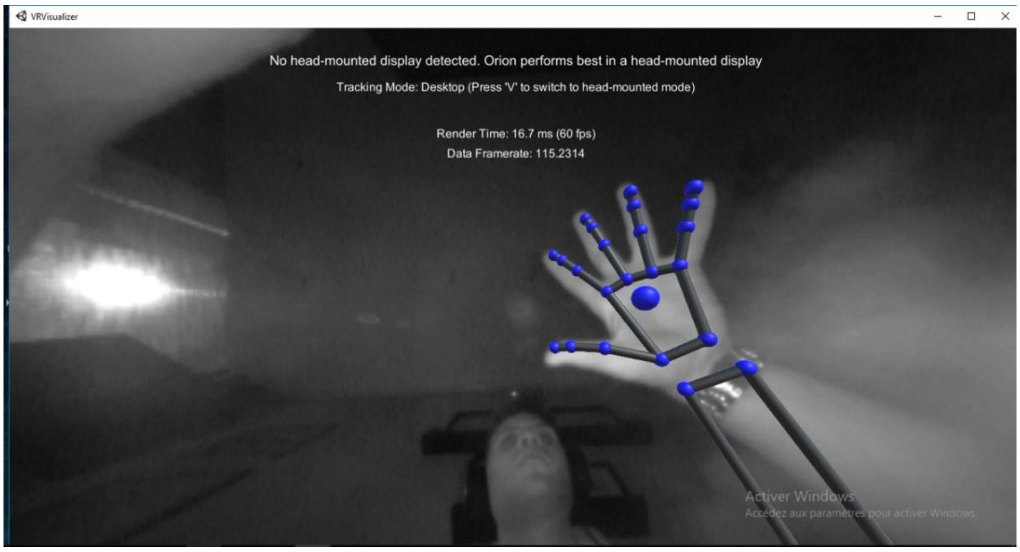
Chapter 5: Results and Comparison

In this chapter we will present the results and we will provide some pictures ensuring that the system is operating in a smooth manner.

5.1. LeapMotion

Below we present a 3D hand detection in real-time using the leap motion sensor, we found that putting the sensor below the arm gives the best results also it works better in short distances, if the hand gets too close to the sensor the system will stop detecting the hand so keeping 400 mm between the sensor and the hand is optimal.

After several tests the Leap motion sensor will work optimally and provides a high accuracy gesture recognition when the palm rotates less than 60°, however when the rotation angle of the palm reaches 90°, the fingers might get occluded by the palm this is called finger occlusion, and when the rotation angle closes to 180°, we get finger occlusion again which makes it harder to perform hand gestures. A second sensor could be used here to cover the blind spots in the first sensor, this solution is far more reliable and accurate than all the others but it increases the complexity and the costs of our system, the Leap motion manufactures developed a prediction algorithm that will predict the position of hand when it reaches the blind spots, this solution is easier and cheaper but it is less reliable and accurate. Also we faced another minor problem it's when the hand is too close to the background, the tracking and the accuracy will be reduced. Thankfully in our Robot controlling system the tracking is done far from the background, so it will performs its task smoothly



No head-mounted display detected. Orion performs best in a head-mounted display

Tracking Mode: Desktop (Press 'V' to switch to head-mounted mode)

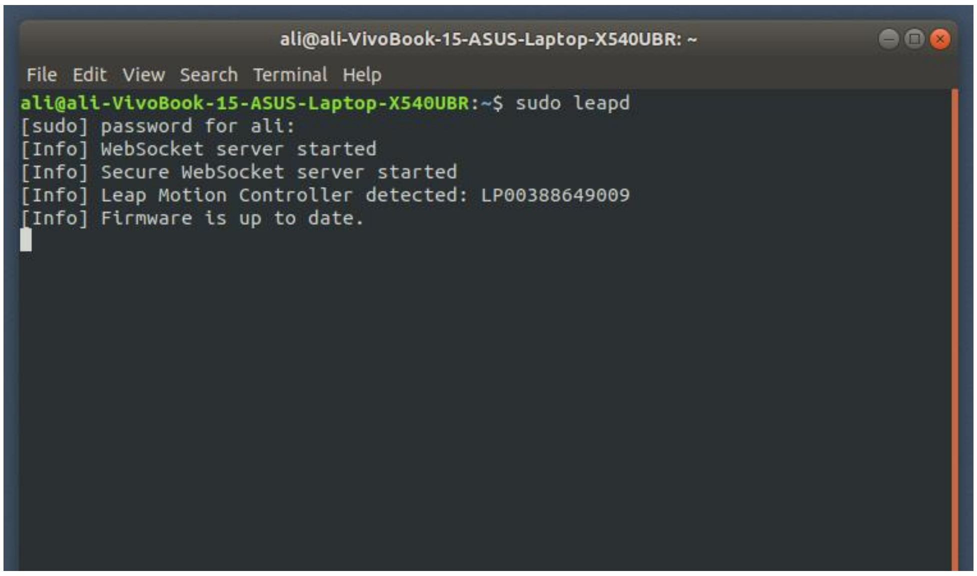
Render Time: 16.7 ms (60 fps)

Data Framerate: 115.2314

Figure 30 One hand recognition

Next we have figures on how the counting fingers is performed.

First we have to run the Leap Motion service on Terminal using “Sudo Leapd” line command, this command require an admin authority this is why we used the “Sudo” command.


ali@ali-VivoBook-15-ASUS-Laptop-X540UBR: ~
File Edit View Search Terminal Help

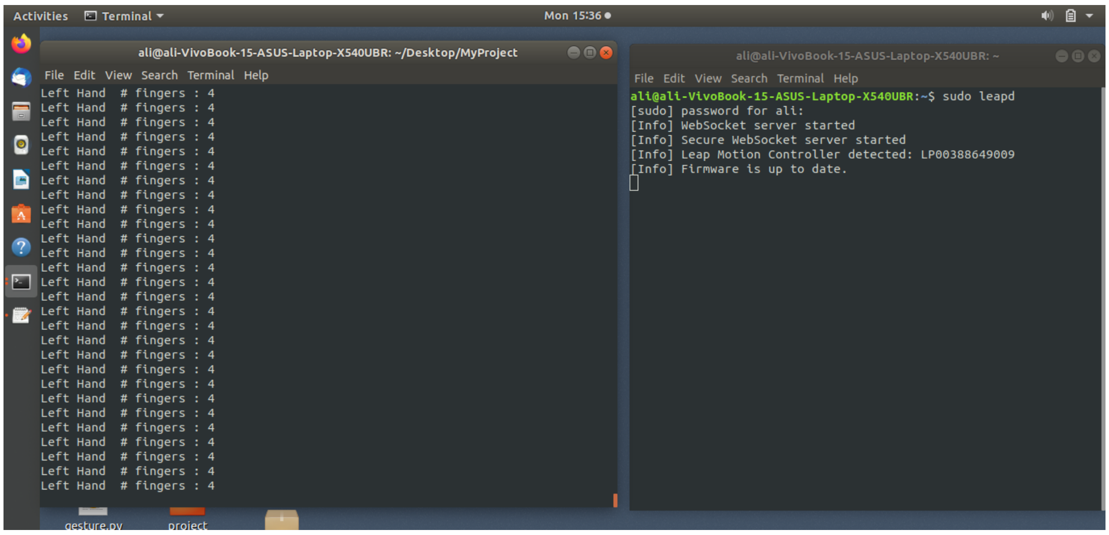
```
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR:~$ sudo leapd
[sudo] password for ali:
[Info] WebSocket server started
[Info] Secure WebSocket server started
[Info] Leap Motion Controller detected: LP00388649009
[Info] Firmware is up to date.
```

Figure 31 Leapd Service running

Leapd is a service that comes along the SDK, it’s completely free to download.

Second we run the python code on a different terminal using “python prg.py” line command.

The Leap Motion system have to be running in the background or else the python code won’t respond.


Activities
Terminal
Mon 15:36
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR: ~/Desktop/MyProject
File Edit View Search Terminal Help
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR: ~
File Edit View Search Terminal Help
gesture.py
project

```
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
Left Hand  # fingers : 4
```

```
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR:~$ sudo leapd
[sudo] password for ali:
[Info] WebSocket server started
[Info] Secure WebSocket server started
[Info] Leap Motion Controller detected: LP00388649009
[Info] Firmware is up to date.
```

Figure 32 running script of counting fingers

The python script running on Terminal allows us to see which hand is used and how many fingers are extended, this is running in real-time so every time a different parameter is changed it will be immediately shown on the terminal.

The python script running on Terminal allows us to see which hand is used and how many fingers are extended, this is running in real-time so every time a different parameter is changed it will be immediately shown on the terminal.

Our Terminal will show exactly which hand is in use and the number of fingers, but also it show the dynamic gestures. Along with finger counting we made sure to detect other static gestures like”OK” gesture.

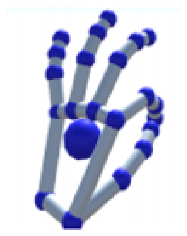
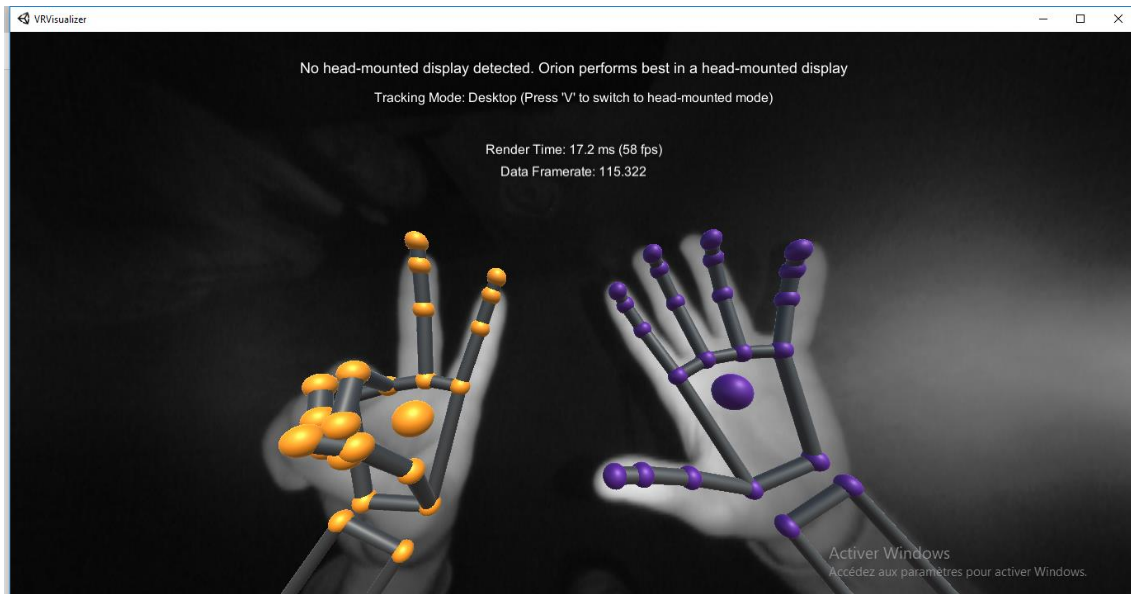


Figure 33 Ok gesture

Other dynamic gestures are detected too like swipe, finger circle, hand circle and tap. These gestures can be applied to new robot movements too or it can be used to mimic hand gestures. Also to extend the number of movements in the robot we can use the second hand too which doubles the number of functions.



No head-mounted display detected. Orion performs best in a head-mounted display

Tracking Mode: Desktop (Press 'V' to switch to head-mounted mode)

Render Time: 17.2 ms (58 fps)

Data Framerate: 115.322

Figure 34 two hands recognition

One of the main advantages of the LeapMotion camera is it covers a concentrate space as it consist of two cameras and three infrared LED’s aimed at above. And since the sensors need to cover a much smaller field, the hardware is significantly more accurate, even the smallest finger movements are recognized and translated nearly without latency.

The field of view is not very large though and spans about 140°-150° which gives it a high accuracy but its accuracy decreases when the hand is around the edges of the field.

One disadvantage of this sensor is that when one finger covers the sensor won’t detect it also when the fingers are really tight next to each other the cameras won’t be able to recognize each finger individually, as we mentioned prior using a second sensor to cover the blind spot here is an optimal solution.

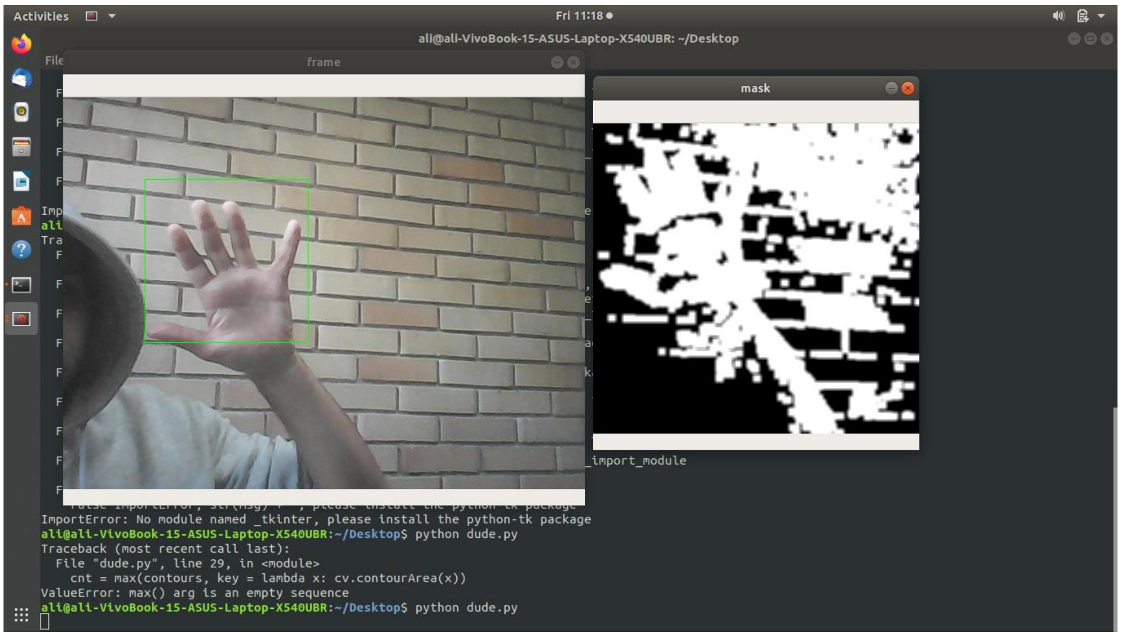
Also one of the big advantages of this device is the availability of open-source software which allows users to contribute to the development of this device. Also it provides flexibility of development as it provides development kits in multiple programming languages as C++, C# and Python.

After all the tests, and considering the ISO9283 the accuracy of human hands is around 0.4 mm thereby the LeapMotion sensor falls under the margin of acceptance as its accuracy is around 0.2 mm in 3D.

Furthermore, we tested the sensor in a no light conditions, the detection will not be successful, so the presence of light is mandatory.

5.2. Intel RealSense

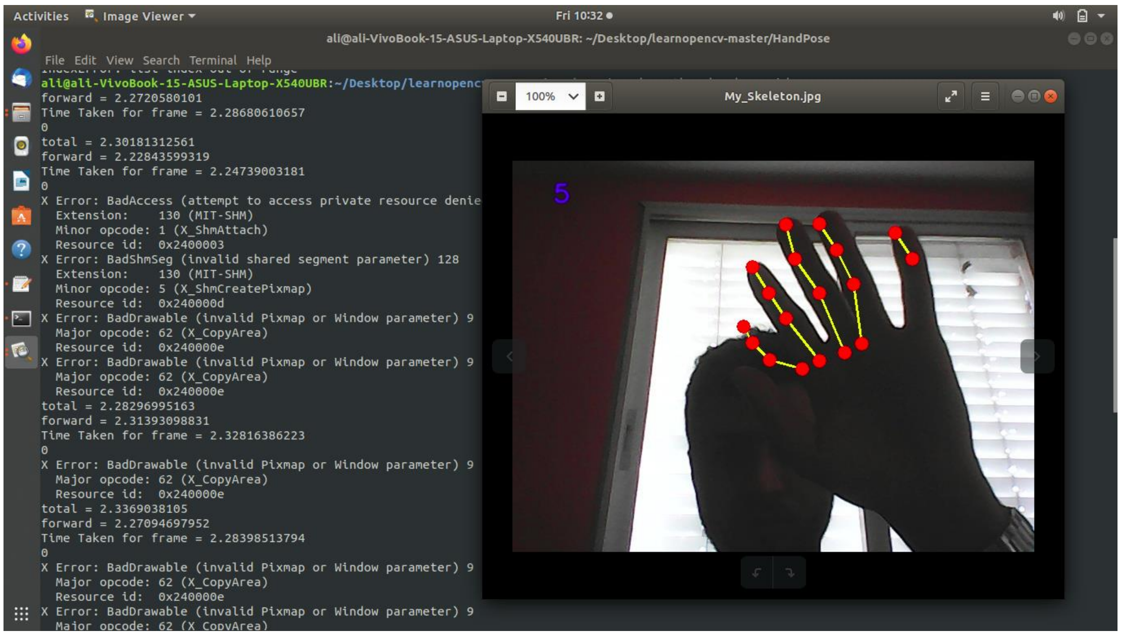
Intel RealSense D415 is used here it is capable taking high definition videos up to 1280 x 720 pixels. We had to try different algorithms to get the best result of this camera, as first we took one picture and tried to fade the background to adjust to the skin color margin that we used, used contours to draw a rectangle around the hand, calculate the angles between each finger to count them, this solution was simple but it lacked the accuracy.



```
_import_module
ImportError: No module named _tkinter, please install the python-tk package
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR:~/Desktop$ python dude.py
Traceback (most recent call last):
  File "dude.py", line 29, in <module>
    cnt = max(contours, key = lambda x: cv.contourArea(x))
ValueError: max() arg is an empty sequence
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR:~/Desktop$ python dude.py
```

Figure 35 hand frame tracking

Introducing a model training using OpenPose libraries allowed us to generate keypoint maps, these maps will provide us with the exact positions of the hand keypoints which represent every bone on the hand. This will be crucial in extracting useful information later such as finger counting. We tried to take a video process each frame on its own, this solution is accurate but it takes significant amount of time to process all the frames as it depends on the processor's capacity, but due the necessity of our system to be able to run in real-time this solution is lacking in term of functionality.


Activities
Image Viewer
Fri 10:32
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR: ~/Desktop/learnopencv-master/HandPose
File Edit View Search Terminal Help
100%
My_Skeleton.jpg
5

```
ali@ali-VivoBook-15-ASUS-Laptop-X540UBR:~/Desktop/learnopenc
forward = 2.2720580101
Time Taken for frame = 2.28680610657
0
total = 2.30181312561
forward = 2.22843599319
Time Taken for frame = 2.24739003181
0
X Error: BadAccess (attempt to access private resource denie
  Extension:    130 (MIT-SHM)
  Minor opcode: 1 (X_ShmAttach)
  Resource id:  0x2400003
X Error: BadShmSeg (invalid shared segment parameter) 128
  Extension:    130 (MIT-SHM)
  Minor opcode: 5 (X_ShmCreatePixmap)
  Resource id:  0x240000d
X Error: BadDrawable (invalid Pixmap or Window parameter) 9
  Major opcode: 62 (X_CopyArea)
  Resource id:  0x240000e
X Error: BadDrawable (invalid Pixmap or Window parameter) 9
  Major opcode: 62 (X_CopyArea)
  Resource id:  0x240000e
total = 2.28296995163
forward = 2.31393098831
Time Taken for frame = 2.32816386223
0
X Error: BadDrawable (invalid Pixmap or Window parameter) 9
  Major opcode: 62 (X_CopyArea)
  Resource id:  0x240000e
total = 2.3369038105
forward = 2.27094697952
Time Taken for frame = 2.28398513794
0
X Error: BadDrawable (invalid Pixmap or Window parameter) 9
  Major opcode: 62 (X_CopyArea)
  Resource id:  0x240000e
X Error: BadDrawable (invalid Pixmap or Window parameter) 9
  Major opcode: 62 (X_CopyArea)
```

Figure 36 Running script of hand gesture recognition

The script running on terminal takes one frame process it and generate two maps one with keypoints located on the detected hand and one with a skeleton which connects the keypoints to form a skeleton map.

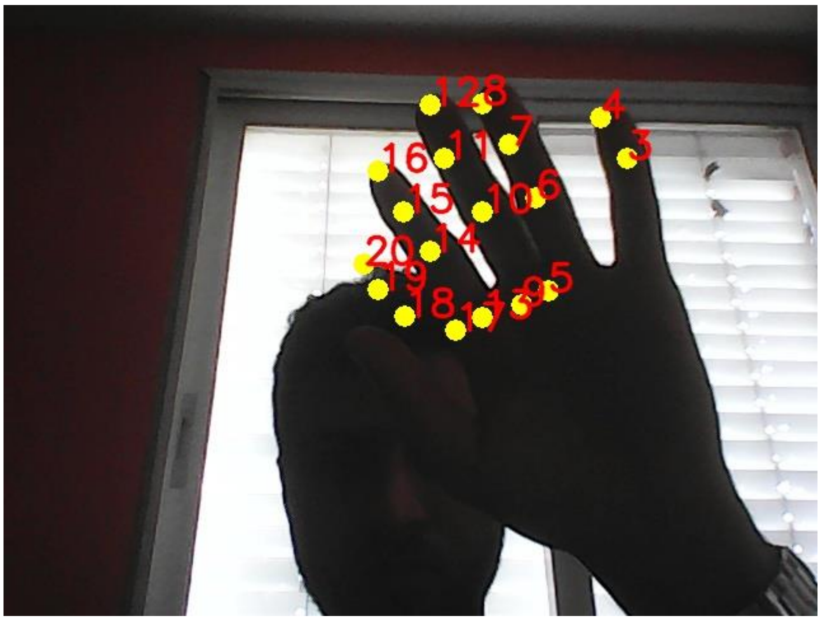


Figure 37 Keypoint map

Each two keypoint represent a bone in our hand, so pairing them was a good idea to further calculate distances and angles.

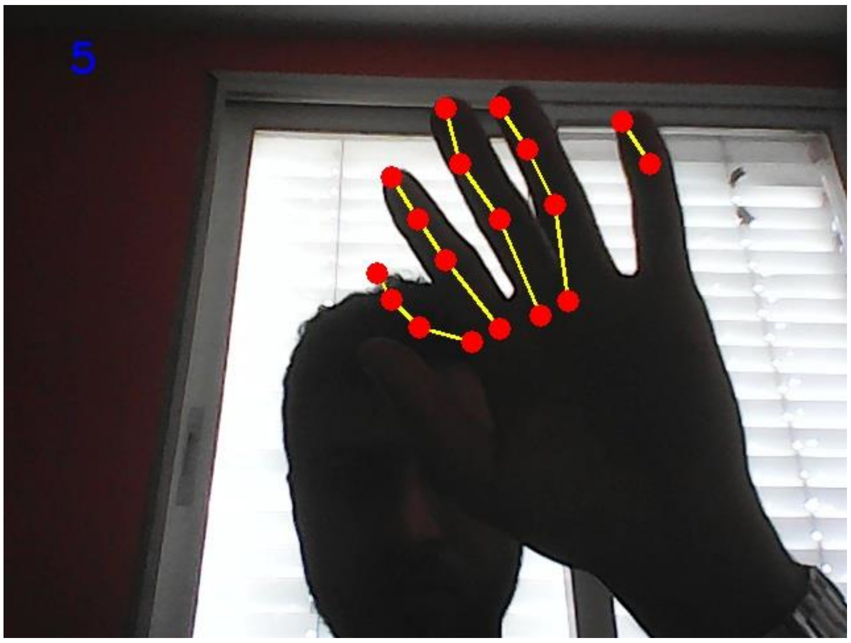


Figure 38 Skeleton map

The skeleton map will show us the pairing of points, also in the picture above it shows the number of fingers after calculating them in the script.

Processing each frame will allow us to recognize all static gestures that we did.

One of the pros of using the Intel RealSense camera is its robustness with a good build quality, it functions well in small distances under 3 meters, it has a good support Intel driver updates and it works on all major operating systems.

But we recognized some disadvantages regarding our system. It has a lot of noise we had to filter the noise off the background every time we use it.

Comparing to the Leapmotion sensor Intel RealSense camera is more versatile and flexible in general usage. If we are focusing on hand gestures recognizing, we found that the Leapmotion camera performs better but in everything else the Intel RealSense is much better. One thing the RealSense camera is better in our system is its safety as it provides the ability to track different objects at the same time which increases the safety of our system, in our case tracking the Robot and other humans around the area of control to maintain the safety of the manipulator, surrounding humans, the robot and the goods that we are dealing with and to ensure the well-functioning of the system.

In term of price both sensors are relatively cheap if we are taking in consideration that we are working in industry but the Intel Real sense's price is higher as it costs around 182 €, as for the LeapMotion sensor it costs 70€ which is much lower price than the Intel RealSense D415.

The presence of light is essential for this sensor to function proporly.

5.3. Performance of the system

To determine the accuracy of our detection methods, we evaluated our approach using confusion matrix. Confusion matrix allows visualization of the performance of each method.

We classified the hand gestures into 6 gesture positions, these positions represent the finger counting from 0 to 5 as shown in the figure below.

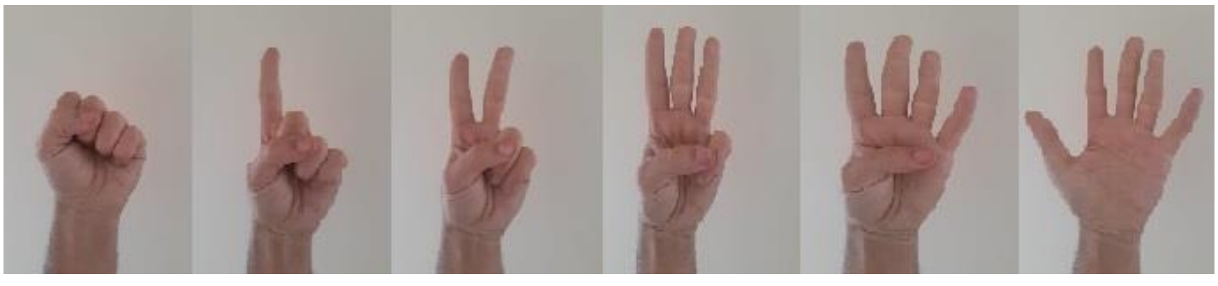


Figure 39 Hand gestures positions

These gestures are used to show how accurate the classification using both cameras, we posed 6 gestures with 20 samples each, totally there were 240 samples of gestures that we used for this test.

First we used the LeapMotion sensor to detect the hand gestures and to provide a confusion matrix.

Table 3 Confusion matrix Leapmotion sensor detection

	Zero_fingers	One_finger	Two_fingers	Three_fingers	Four_fingers	Five_fingers
Zero_fingers	20	0	0	0	0	0
One_finger	0	19	1	0	0	0
Two_fingers	0	1	18	1	0	0
Three_finger	0	0	0	16	3	1
Four_fingers	0	0	0	0	18	2
Five_fingers	0	0	0	0	1	19

This Confusion matrix allowed us to measure the performance of this method.

Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{110}{120}$ = 91.66%

Miss classification = $\frac{FP+FN}{TP+TN+FP+FN} = \frac{10}{120}$ = 8.33%

Table 4 Confusion matrix Intel RealSense D415 camera detection

	Zero_fingers	One_finger	Two_fingers	Three_fingers	Four_fingers	Five_fingers
Zero_fingers	18	1	1	0	0	0
One_finger	0	18	2	0	0	0
Two_fingers	0	2	17	1	0	0
Three_finger	0	0	1	17	2	0
Four_fingers	0	0	0	1	16	3
Five_fingers	0	0	0	0	2	18

Accuracy = $\frac{TP+TN}{TP+TN+FP+FN} = \frac{104}{120}$ = 86.66%

Miss classification = $\frac{FP+FN}{TP+TN+FP+FN} = \frac{16}{120}$ = 13.33%

For further performance analysis we calculated the Precision and the sensitivity of each gesture.

Precision = $\frac{TP}{TP+FP}$

Sensitivity (Recall) = $\frac{TP}{TP+FN}$

Table 5 Precision comparison

	LeapMotion	Intel RealSense D415
Zero_fingers	100%	90%
One_finger	95%	90%
Two_fingers	90%	85%
Three_finger	80%	85%
Four_fingers	90%	80%
Five_fingers	95%	90%

Table 6 Sensitivity comparison

	LeapMotion	Intel RealSense D415
Zero_fingers	100%	100%
One_finger	95%	85.714%
Two_fingers	94.737%	80.952%
Three_finger	94.118%	89.474%
Four_fingers	81.818%	80%
Five_fingers	86.364%	85.714%

5.4. Tasks done by the Robot

Our robot has 3 main actions, these actions are already programmed so our job is to send the command by ModbusTCP.

The ModbusTCP is protocol with a TCP interface allows us to connect our system to the robot using a port.

The first action is to build a tower from blocks in going from the big block in the bottom until the smallest one in the top.

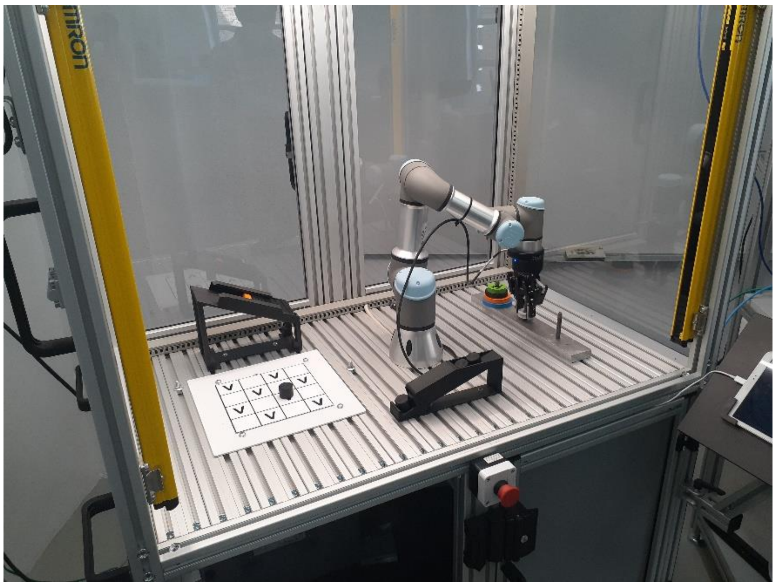


Figure 40 Tower building

The second action is a game which the robot take the black blocks from the beak and put them in the ticked area, it uses a built in camera to recognize where it should put the block. It will continue putting all the black blocks until all the ticked areas are full.

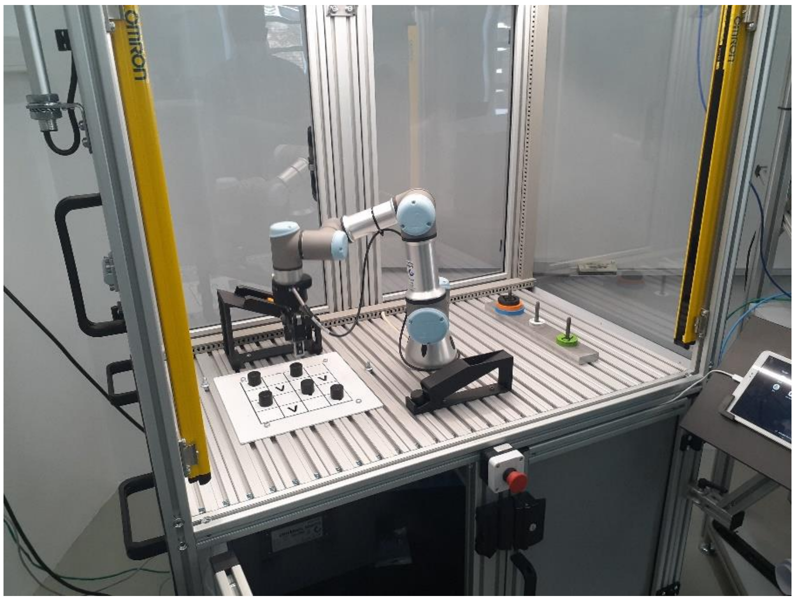


Figure 41 putting and removing blocks in and from the ticked area

The third action is to remove the black blocks from the ticked areas and put them back in the beak, it will take the blocks one by one until all the ticked areas are empty.

Our system will count the number of fingers we are extending and then translate it to a command, per example if I showed two finger as hand gesture the robot will do the task number 1 which is the Tower building.

Chapter 6: Conclusion and Future work

6.1. Conclusion

This thesis shows the using of different methods in the field of hand gesture these methods consist of detecting different gestures in short distance using different cameras. The aim of applying these methods is to compare different algorithms and hardware to classify hand gesture recognition methods, we evaluated these methods in term of many parameters such as accuracy, simplicity, execution time, safety and sensitivity .the first method is using a Leap motion sensor, this method provides a lot of functions already set by manufactures and relatively easy to use which made it simpler and relatively easier to implement in comparison to other methods as hand manipulation libraries are rich of pre developed functions, also it's cheaper than most all the other alternatives out there, performs well in term of accuracy and sensitivity, great in term of execution time as it focuses solely on hand gesture recognition and ignore all the potential of tracking other objects in frame or background which decreases the safety in comparison to other hardware, Intel Real sense camera provides the possibility to track hand gestures and other objects in the field of view or background which made it safer and more reliable in the industrial field also it offers a great amount of accuracy and sensitivity, but it increases the complexity of the system as it offers the use of different algorithms. In term of time execution it falls short to the first method as processing all the data in a video frame by frame takes a lot of time if we want to control the robot in a real time. We applied different algorithms to reduce the execution time without stripping this method of its strengths. One algorithm uses 3D video extraction and processing each frame one by one. This algorithm amasses a great amount of usable data in term of hand tracking and other objects in the background also it offers a great amount of accuracy but it lacks significantly in term of time execution. Extracting 2D video reduced the execution time but it didn't help much in Real-time domain. Offering a second algorithm to extract a set amount of frames and process each of them reduced the execution time but made the system less accurate, the third algorithm uses CNN which is multi layers neural network it's one of the deep learning techniques. Applying CNN in this algorithm reduced two phases which are image extraction and classification to one phase only, so processing data becomes faster.

At the end the best solution we come to is to uses a hybrid system of multi cameras/sensors which take the strengths of the Leap motion sensor and the strengths of the Real sense camera, in that way we get the best of both worlds.

6.2. Future Work

After performing all the methods and reviewing the results, the following are suggestions for future works:

Designing a mobile application to translate common gestures to meaningful words, the output could be words shown on a screen or be dictated by sound to aid people with hearing impairments.

Record gestures with high resolution size such as 6k to examine the CNN algorithm.

Implement different algorithms in deep learning like RNN and compare the efficiently of CNN and RNN in terms of accuracy.

Bibliography

Agrawal, S. S. (2012). *Vision based hand gesture recognition for human computer interaction: a survey.* Dordrecht: Springer Science+Business Media.

Baxter, J. (2000). A model of inductive bias learning. *J Artif Intell*, 149-198.

Bourke A, O. J. (n.d.). *Evaluation of a threshold-based tri-axial accelerometer fall*. Retrieved from https://www.sciencedirect.com/science/article/B6T6Y-4MBCJHV-1/2/f87e4f1c82f3f93a3a5692357e3fe00c

Cote M, P. P. (2006). Comparative study of adaptive segmentation techniques for gesture analysis in unconstrained environments. *IEEE*, 28-33.

Dina Satybaldina, G. K. (2016). *Application Development for Hand Gestures.* Moscow,Russian Federation: National Research Nuclear University "MEPhI".

FrankWeichert*, D. (2013, June 13). AnalysisoftheAccuracyandRobustnessoftheLeap MotionController. *ISSN*, p. 14.

geeksforgeeks. (2015, 5). *geeksforgeeks.org.* Retrieved from www.geeksforgeeks.org: https://www.geeksforgeeks.org/openpose-human-pose-estimation-method/#:~:text=OpenPose%20is%20the%20first%20real,C%2B%2B%20implementation%20and%20Unity%20Plugin.

Haiyang Jin, Q. C. (2016, june 2). *Sciencedirect.* Retrieved from Sciencedirect.com: https://www.elsevier.com/catalog?producttype=journal

Hongyi Liu, L. W. (2017, January 13). Gesture recognition for human robots collaborative: a review. *International journal of industrial ergonomics*, p. 13.

Hornegger, J. (2008). Gesture recognition with a Time-Of-Flight camera. *International Journal of Intelligent Systems Technologies and Applications*, 5.

Jože Guna *, G. J. (2014, January 14). *An Analysis of the Precision and Reliability of the Leap Motion Sensor and Its Suitability for Static and Dynamic Tracking.* Retrieved from mdpi.com: www.mdpi.com/journal/sensors

Kruger, J. L. (2012, Octobre 7). Spatial Programming for Industrial Robots based on Gestures and Augmented Reality. *IEEE*, p. 7.

LeCun, Y. (1998). *analyticssteps*. Retrieved from analyticssteps.com: https://www.analyticssteps.com/blogs/convolutional-neural-network-cnn-graphical-visualization-code-explanation

Mitra S, A. T. (2007). Gesture recognition: a survey. *IEEE* , 311-324.

Monica Carfagni, R. F. (2018, Decembre 21). Metrological and Critical Characterization of the Intel. *MDPI*, p. 20.

Pantic M, N. A. (2008).) Human-centred intelligent human–computer Interaction. *Int J Auton Adapt Commun*, 168-187.

S. ROBLA-GÓMEZ 1, V. M.-S.-F.-O. (2017, Septembre 26). Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments. *IEEE*, p. 20.

Wachs, J. K. (2011). Vision-based hand-gesture applications. *ACM*, 60-71.

Wechsler, F. L. (2005). Open Set Face Recognition Using Transduction. *IEEE*, 1686-1697.

Yonjae Kim, P. C. (2014). Experimental Evaluation of Contact-Less Hand Tracking Systems for. *IEEE*, 3502-3509.

Zhang, Y. H. (2015). wearable, low-cost electrical impedance tomography. *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 215-243.