MICROSOFT’S KINECT CONTROLLED ROBOTICARM

A PROJECT REPORT

Submitted by

NITHIN SUNDARARAJ (13BCS065)

PRAVEEN.M (13BCS072)

UDHAYA SANKAR (13BCS108)

in partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

KUMARAGURU COLLEGE OF TECHNOLOGY,

COIMBATORE

(An Autonomous Institution Affiliated to Anna University, Chennai)

ANNA UNIVERSITY::CHENNAI 600 025

APRIL 2017

KUMARAGURU COLLEGE OF TECHNOLOGY

COIMBATORE 641 049

(An Autonomous Institution Affiliated to Anna University, Chennai)

BONAFIDE CERTIFICATE

Certified that this project report titled **"MICROSOFT'S KINECT CONTROLLED ROBOTIC ARM"** is the bonafide work of **NITHIN SUNDARARAJ (13BCS065), PRAVEEN M (13BCS072) and UDHAYA SANKAR (13BCS108)** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. P. Devaki	**Mrs. Syed Ali Fathima S J**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor and Head	Assistant Professor
Department of Computer Science and Engineering	Department of Computer Science and Engineering
Kumaraguru College of Technology	Kumaraguru College of Technology
Coimbatore-641049	Coimbatore- 641049

The candidates with **University Register Numbers 13BCS065, 13BCS072** and **13BCS108** were examined by us in Project Viva-Voice examination held on

__________.

INTERNAL EXAMINER	**EXTERNAL EXAMINER**
NAME:	**NAME:**
DESIGNATION:	**DESIGNATION:**

ACKNOWLEDGEMENT

We take this opportunity to express our profound gratitude to our founder Padmabushan Arutselvar **Dr.N.Mahalingam, B.Sc., F.I.E,** Co-Chairman **Dr.B.K.Krishnaraj Vanavarayar, B.Com., B.L.,** Correspondent **Mr.M.Balasubramanian, M.Com., M.B.A.,** and Joint Correspondent **Mr.Shankar Vanavarayar, M.B.A., PGDIEM.,** for giving a great opportunity to pursue this course.

We extend our gratefulness to **Dr.R.S.Kumar BE (Hons) M Tech Ph.D.,** Principal for providing the necessary facilities to complete our project.

We would like to make a special acknowledgement and thanks to **Prof.Dr.P.Devaki Ph.D,** Head of Department of Computer Science and Engineering, for her support and encouragement throughout the project.

We thank our project coordinator, **Dr.Mrs.Chandrakala Ph.D.,** professor, Department of Computer Science and Engineering for her support during the course of our project.

We express our heartfelt thanks to our project guide **Mrs.S.J.Syed Ali Fathima M.E.,** Assistant Professor, Department of Computer Science and Engineering, for her incredible support during the project.

We would like to convey our honest thanks to all **faculty members and non-teaching** staff of the department for their support.

ABSTRACT

The field of sensor based human computer interaction and robotics is rapidly developing. As there is a continuous advancement in the techniques followed, the way of using every day technological products is also improving proportionally. From man-powered to machine controlled environment the importance is given to the way technology is used. Nowadays the physical control of devices is just an old fashioned style whereas there are unique ways of controlling the machines such as using user’s voice and gesture.

The idea is about using a Microsoft Kinect device to implement a suitable interface between a machine and human where users can control machines (for e.g. computer, robot etc.). The actual need of such a technology is to reduce human efforts over controlling machines and this will be the first step towards the concept of artificial intelligence.

The actual problem arises when a human is not able to execute his commands over a machine. This is due to the difficulty in understanding the in-depth mechanism or cause of human errors (such as programming errors or manufacturing errors). Such difficulties can be avoided by using advanced machines which can understand or get the commands through the human gestures (using Kinect) and execute the required action, thus there is no necessity of programming knowledge or any physical controller.

The skeletal tracking ability of Kinect provides an efficient way of controlling a device through gestures. The Kinect sensor uses a depth map and colour image as well. The Kinect SDK for programming can be used to create user required applications over the usage of Kinect. Thus the idea revolves around how Kinect as an input device can be implemented for a robotic arm which works based on controlling the servo motors with the help of Arduino board for processing the gesture input. If the same methodology is implemented for an entire robot, intelligent robots that can perceive and act like humans can be developed.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

S.NO	ACRONYM	ABBREVIATION
1	KINECT	Kinetic-connect
2	IDE	Integrated development environment
3	IR	Infra-red
4	PCB	Printed circuit board
5	PWM	Pulse Width Modulation
6	TX	Transmitter
7	RX	Receiver
8	I2C	Inter Integrated Circuit
9	NI	Natural Interaction
10	OS	Operating System
11	API	Application Programming Interface
12	JDK	Java Development Kit
13	SDK	Software Development Kit
14	ADC	Analog to Digital Converter
15	DAC	Digital to Analog Converter

CHAPTER 1

INTRODUCTION

The range of Human Computer Interaction started by using devices such as keyboard through which users enter their commands and based on which the system responds. Although this approach was inconvenient or unnatural, these devices were dominant. Thus the evolution of text based UI to a 2D graphical based interface was made possible.

Finally on the path of inventing a whole new level of technology the gesture based innovations began to rise. Thus a more natural way of communicating with machines using human body movements can be developed. This idea introduces various sensors that could scan or track human head, hands, arms, face or body and based on its movements the corresponding instructions are sent to the system providing semantic information.

Microsoft developed the Kinect device as a medium of gaming, but later its applications improved to various fields of research and development. The skeletal tracking ability of Kinect provides an efficient way of controlling a device through gestures. The Kinect sensor uses a depth map and colour image as well. The Kinect SDK can be used for programming applications over the usage of Kinect. Thus the idea revolves around how Kinect as an input device can be used for a robotic arm which works based on controlling the servo motors with the help of Arduino board for processing the gesture input. If the same methodology is implemented for an entire robot, intelligent robots that can perceive and act like humans can be introduced.

The proposed methodology introduces a basic understanding on the gesture based technologies and how a robotic arm using a device known as Kinect can be developed. The skeletal tracking ability of Kinect plays an important role in the functioning of the robotic arm.

CHAPTER 2
LITERATURE REVIEW

Kinect device has been used in various research projects which have been carried out on a large industrial scale as well. Some of the important research works are:

2.1 DEVELOPING A GESTURE BASED REMOTE HUMAN-ROBOT INTERACTION SYSTEM USING KINECT – [4]

2.1.1 PROBLEM DEFINITION

Gesture based natural human-robot interface is an important function of robot teleoperation system. Such interface not only enables users to manipulate a remote robot by demonstration, but also ensures user-friendly interaction and software reusability in developing a networked robot system. In this paper, an application of gesture-based remote human-robot interaction is proposed using a Kinect sensor. The gesture recognition method combines depth information with traditional Camshift tracking algorithm by using Kinect and employs HMM in dynamic gesture classification. A Client/Server structured robot teleoperation application system is developed, which provides a favourable function of remotely controlling a dual-arm robot by gestural commands. Experiment results validate the practicability and effectiveness of the application system.

2.1.2 DOMAIN ANALYSIS

Human-computer interaction arose as a field from intertwined roots in computer graphics, operating systems, human factors, ergonomics, industrial engineering, cognitive psychology, and the systems part of computer science. Computer graphics was born from the use of cathode-ray tube (CRT) displays and pen devices very early in the history of computers. The pursuit to create man-machine symbiosis led to the development of many human-computer interaction techniques, starting from primitive text and graphical user interfaces (GUI) to speech and gesture recognition interfaces.

An advantage of gesture is the ability to provide multidimensional input. While most pointing device provide position and orientation information for a single point in space, a gesture interface can input many positions simultaneously since the system tracks multiple features. The aim of the proposed work in this document is to demonstrate this ability, by developing an environment where the gestures are segmented, tracked and recognized. In addition the environment will be able to recognize the position of the palm and each fingertip, generate data to represent the trajectory, acceleration and velocity, as well as the feature of the hand and translate this data on a display, allowing control of a GUI.

2.1.3 METHODOLOGY

In this paper, four basic patterns of gestures which are designed as robot control commands. These gestures are WaveLeft, WaveRight, RiseUp, and PutDown. An interactive hand tracking strategy combing color data and depth data is designed. Firstly, a person waves his/her hand in front of a Kinect sensor. Then, a motion detector in color camera between successive frames is applied to locate the hand, which is taken as the initial tracking target. After the initialization, Camshift tracking algorithm is employed for real-time hand tracking. Since it is well known that Camshift-based tracking algorithm using only color information may produce tracking error when there is similar color in background. Depth information is introduced to facilitate hand tracking according to the fact that hand is usually separated from the background object in depth, and has fixed moving range. Hence threshold segmentation in depth map can accurately distinguish the player from the background.

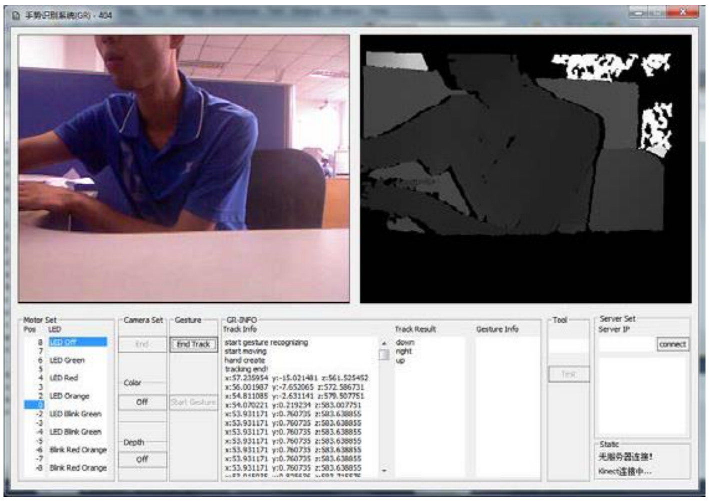
2.1.4 ALGORITHM EXPLANATION

Hidden Markov models are used to model the underlying processes of gestures and determine the underlying processes behind the individual components of a gesture.

Firstly, the obtained gesture trajectories are converted into 16 direction codes representing direction vectors. Each angle is converted into one of the sixteen direction codes. The angle ranges of the direction codes have different widths. The feature of 2D motion trajectory of hand gesture is comprised of a series of discrete movement points, and it is represented by a series of discrete movement direction value. For the 2D motion plane, directions are divided into eight discrete values. Therefore, the trajectory of dynamic gesture ban be described by the sequence of discrete direction value. This transforms the original state sequence into an encoded observation sequence.

2.1.5 RESULT AND METRICS

In the experiment, a gesturer stood 0.3m~0.5m in front of a Kinect sensor connected to a client computer. He made four of the command gestures such as WaveLeft, WaveRight, RiseUp, and PutDown to direct robot. Client and server computers are both with Intel Core 2 Duo E8200 2.66GHz CPUs, and the software on both client and server computers was developed using Microsoft Visual Studio 2010. To test the accuracy of gesture recognition, a set of 50 image sequences containing four of the above and other undefined gestures were collected. These sample sequences of images were performed by different gesturers and backgrounds under normal lighting conditions. Figure 5 shows an example of the confidence estimation of a WaveRight gesture. In addition, Figure 6 shows the result that corresponds to the situation that the user was making a sequence of WaveLeft, WaveRight and RiseUp gestures. In general, recognition accuracy of nearly 85% was achieved.



```
start gesture recognizing
start moving
hand create
tracking end!
x:57.235954  y:-15.021481  z:561.525452
x:56.001987  y:-7.652065  z:572.586731
x:54.811085  y:-2.631141  z:579.507751
x:54.070221  y:0.219234  z:583.007751
x:53.931171  y:0.760735  z:583.638855
x:53.931171  y:0.760735  z:583.638855
x:53.931171  y:0.760735  z:583.638855
x:53.931171  y:0.760735  z:583.638855
x:53.931171  y:0.760735  z:583.638855
x:53.931171  y:0.760735  z:583.638855
x:53.931171  y:0.760735  z:583.638855
```

```
down
right
up
```

Figure 2.1: Software GUI

2.1.7 CONCLUSION AND FUTURE WORK

In this paper a natural interaction framework is proposed that integrates an intuitive tool for tele-operating a mobile robot through gestures. It provides a comfortable way of interacting with remote robots and offers an alternative input for encouraging non-experts to interact with robots. Application system is developed combing low-cost Kinect device and network technology. For the future work, gesture recognition will be improved to support mobile robot navigation scenarios. The better gesture recognition ratio will allow the usage of more ergonomic poses and gestures for the user, thus reducing the fatigue associated with the current gesture recognition.

2.2 MOTION DETECTION USING KINECT DEVICE FOR CONTROLLING ROBOTIC ARM – [5]

2.2.1 PROBLEM DEFINITION

With the advance in video technology, motion-based computing system has an effect on both hardware and software. Kinect is a webcam style add-on peripheral for Xbox 360 game console manufactured by Microsoft, which is featured by RGB camera and multi-array microphone. Kinect is a motion-based software technology that can provide 3-D motion detection, skeleton motion tracking and voice and facial recognition. In this paper, a mechanism is built to use human body to control robotic arm through Kinect Device using the Kinect SDK for Windows library. To develop a program to track the human motion by getting the stereo skeleton joints from the stream of the Kinect device, build an Action Script program with Adobe Flash and control the robotic arm.

2.2.2 DOMAIN ANALYSIS

Human-computer interaction arose as a field from intertwined roots in computer graphics, operating systems, human factors, ergonomics, industrial engineering, cognitive psychology, and the systems part of computer science. Computer graphics was born from the use of cathode-ray tube (CRT) displays and pen devices very early in the history of computers. The pursuit to create man-machine symbiosis led to the development of many human-computer interaction techniques, starting from primitive text and graphical user interfaces (GUI) to speech and gesture recognition interfaces.

An advantage of gesture is the ability to provide multidimensional input. While most pointing device provide position and orientation information for a single point in space, a gesture interface can input many positions simultaneously since the system tracks multiple features. The aim of the proposed work in this document is to demonstrate this ability, by developing an environment where the gestures are segmented, tracked and recognized. In addition the environment will be able to recognize the position of the palm and each fingertip, generate data to represent the trajectory, acceleration and velocity, as well as the feature of the hand and translate this data on a display, allowing control of a GUI

2.2.3 METHODOLOGY

Kinect is a kind of device that creates three dimensional data. This is because it has infrared ray emitter and receiver. Furthermore, it also has a Colour Camera that supports traditional two dimensional colour-image data. The Infrared Projector will emit infrared ray and be reflected by the first meeting object. The other side of the Kinect Device assembled an Infrared Camera which will receive the infrared ray returned. Base on the data, the computer will format it and pass it as a depth image stream which can be display as a depth image.

Analysing the characteristics of the Kinect depth data has been examined and it shows the depth data is formatted and display on a bit map. Because the max value has been set as 256, so the image looks like a grey style image, the litter the colour is, the further the corresponding object is. Because the Microsoft SDKs does not support the access to infrared data for any developer other than OpenNI, a third party library, which allow programmer to edit the infrared data directly, so some deep details in Microsoft SDKs are transparent to developers.

Microsoft embedded interfaces containing image processing algorithms for developers. However, those algorithms are transparent to users. But the stable image processing technique can help us to detect human body. Base on the human body, the Kinect library can calculate 20 joints, and get their three dimensional locations. The three dimensional points of Kinect world is showing on the figure below.

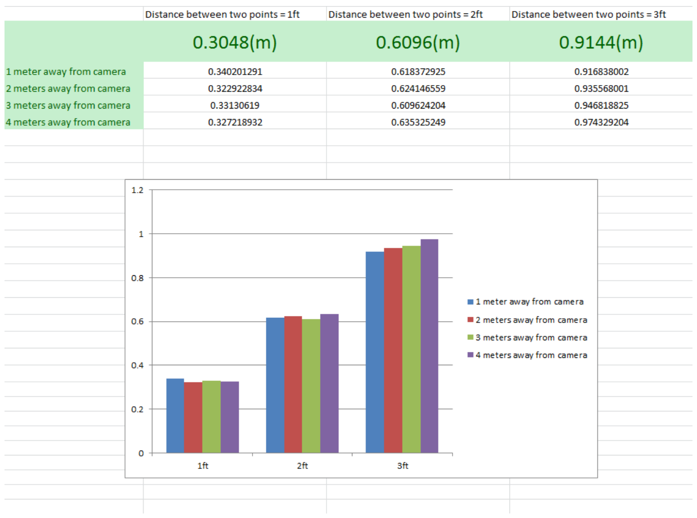
2.2.4 ALGORITHM EXPLANATION

Since there are three dimensional locations for points in the Kinect space, the distance between them can be measured.

Such as there are two points: A ($x1$,1,$z1$), B ($x2$,$y2$,$z2$)

Then the distance between A and B is: D= ($x1-x2$)2+($y1-y2$)2+($z1-z2$)22

Compared with the real distance, the accuracy of this algorithm can be seen in the following chart. As Figure shows, when a person stand away from the camera for 1 meter, 2 meters, 3 meters, 4 meters, the measured values for 1feet, 2 feet, and 3 feet are listed in the chart.



	Distance between two points = 1ft	Distance between two points = 2ft	Distance between two points = 3ft
	0.3048(m)	0.6096(m)	0.9144(m)
1 meter away from camera	0.340201291	0.618372925	0.916838002
2 meters away from camera	0.322922834	0.624146559	0.935568001
3 meters away from camera	0.33130619	0.609624204	0.946818825
4 meters away from camera	0.327218932	0.635325249	0.974329204

1.2
1
0.8
0.6
0.4
0.2
0
1ft
2ft
3ft
1 meter away from camera
2 meters away from camera
3 meters away from camera
4 meters away from camera

Figure 2.2: Value comparison chart

2.2.5 RESULT AND METRIC

The result involves building the Server side project which is the Kinect project. It will keep listening in the host processor. Second, the Client side project is built which will be the Flash project. It is going to send application to the host. After a Handshake process, these two processors can communicate to each other. The command generated will be packaged and sent from host processor to Client processor. Finally the robotic arm will be controlled.

The eight simple commands that need to be identified are UP, DOWN, LEFT, RIGHT, FORWARD, BACKWARD, OPEN, and CLOSE. To implement the commands either real robotic arm or virtual robotic arm is used. However, in this implementation a flash project is used to simulate a robotic arm. The details of each command are as following:

1) UP: To move the robotic arm up.
2) DOWN: To move the robotic arm down.

3) LEFT: To move the robotic arm left.
4) RIGHT: To move the robotic arm right.
5) FORWARD: To move the robotic arm forward.
6) BACKWARD: To move the robotic arm backward.
7) OPEN: Open the robotic arm.
8) CLOSE: Close the robotic arm.

2.2.6 CONCLUSION AND FUTURE WORK

In conclusion, robotic is able to perform tasks where the human cannot access, and for controlling the robot commands generated by Kinect device which detects human motion can be used efficiently. Through socket communication, commands can be passed to the robotic device which will implement them. This methodology introduced motion detection using Kinect device and Robotic control. Through socket communication, either virtual or real robotic arm can be manipulated by the commands. Such innovative controlling methods can followed for other mechanical based machines as well.

2.3 GLOVE BASED AND ACCELEROMETER BASED GESTURE CONTROL – [7]

2.3.1 PROBLEM DEFINITION

This paper reviews the technology for using hand, body and facial gestures as a means for interacting with computers and other physical devices. It discusses the rationale for gesture based control technology, methods for acquiring and processing such signals from human operators, applications of these control technologies, and anticipated future developments. Today, there is a growing interest in research and development of new human-machine interaction systems that are more natural and ergonomic for the users. The gesture recognition plays an important part of human-machine interaction systems. The focus is done in systems that are based on accelerometers, and on glove based equipments. Based on several papers, the process for different types of approach of gesture control is described. Some applications of these technologies are also presented.

2.3.2 DOMAIN ANALYSIS

Gestures are defined as human motion sequences with trajectories performed in a short interval of time. Human gestures are a natural means of interaction and communication among people. Gestures employ hand, limb and body motion to express ideas or exchange information non- verbally.

Gestures can be divided into two groups: static and dynamic gestures. According to input device, the gesture recognition technique can be divided into three categories: glove-data based, vision- based and accelerometer-based.

Glove-data based gesture recognition systems require users to wear gloves and cumbersome devices to record the movement status. The performance of vision-based systems is relied heavily on the operation environment, e.g., the background and the lighting condition. In addition, these systems face the occlusion problem, let alone the low sampling rate issue.

With the rapid development of micro-electro-mechanical system technology (MEMS), the accelerometer-based gesture recognition becomes increasingly popular and already shows potential in practical applications.

2.3.3 METHODOLOGY

The recognition engine is divided into two components: the data acquisition and the gesture manager.

Data acquisition: The data acquisition component is responsible for processing the received data and then transmits them to the gesture manager. First, a set of filter is used to optimize the data.

For example, the position/orientation information is very noisy due to dependence of lighting conditions. Thus, orientation data that exceed a given limit are discarded as improbable and replaced with their previous values. This type of filters is applied: dead band filter, dynamically adjusting average filter. Note that to be recognizing as a posture, the user has to hold a position between 300 and 600 milliseconds in order to allow the system to detect a posture.

Gesture manager: The gesture manager is the principal part of the recognition system. This library maintains a list of known postures. The system tries to match incoming data with existing posture. This is done by first looking for the best matching fingers constellation. Five dimensional vectors represent the bend values of the fingers and for each posture definition the distance to the current data is calculated. Then, the position/orientation data is compared in a likewise manner.

Finally, in this gesture recognition system, a gesture is just a sequence of successive postures. For example, let’s consider the detection of a ”click” gesture. This gesture is defined as a pointing posture with outstretched index finger and thumb and the other fingers flexed, then a tapping posture with half-bent index finger.

2.3.4 ALGORITHM EXPLANATION

There is a system allowing the training and recognition of arbitrary gestures with the use of 3-axis accelerometers. With this type of device, spatial and temporal data has to be built. A mathematical process is required to exploit these signals. Gestures are represented with data vectors representing the current acceleration of the controller in 3-axis. Theses vectors are analysed to train and to recognize patterns for distinct gestures.

The process of gesture recognition with accelerometer based device. First the “Quantizer” is responsible to cluster the data using a k-mean algorithm. Then, the “model” is a discrete Hidden Markov Model (HMM) that is used to train/recognize characteristic patterns for distinct gestures. Finally, a “Bayes classifier” is used to select the appropriate gesture.

Filtering: Two filters are applied to the vector data to get a minimum representation of a gesture. The first filter is a simple threshold eliminating all vectors which do not have a significant contribution to the characteristic of a gesture.

Quantizer: Acceleration sensor produces too much data. Before using HMM, data has to be clustered and abstracted. A k-means clustering method is used to partition the n observations into k clusters. Each observation belongs to the cluster with the nearest mean. The k is the number of clusters, their collection forms the codebook.

Model and Classifier: The gesture recognition system works in two phases: training and recognition. The training consists of several repetitions of each gesture that must be recognized later.

2.3.5 CONCLUSION AND FUTURE WORK

Technologies developed based on gesture are now really affordable and converged with familiar and popular technologies like TV, large screen. It’s ubiquitous and non- intrusive as a camera or remote with the TV. From this paper the trends of gesture controlled communication systems can be seen. Easing of the technology use, affordability and familiarity indicate that gesture based user interface can open new opportunity for elderly and disable people.

The research's gesture controlled communication aid for elderly and disabled people' can be a significant task for future. The two important aims of the research are to identify the different gestures of elderly and disabled people for communication and to design a rich augmented-reality interface for communication via ubiquitous device such as a television set.

2.4 A KINECT BASED NATURAL INTERFACE FOR QUADROTOR CONTROL – [1]

2.4.1 PROBLEM DEFINITION

This paper presents a new and challenging approach to the control of mobile platforms. Natural user interfaces (NUIs) and visual computing techniques are used to control the navigation of a quadrotor in GPS-denied indoor environments. A visual odometry algorithm allows the platform to autonomously navigate the environment, whereas the user can control complex manoeuvres by gestures and body postures.

This approach makes the human–computer interaction (HCI) more intuitive, usable, and receptive to the user's needs. The NUI presented in this paper is based on the Microsoft Kinect and users can customize the association among gestures/postures and platform commands, thus choosing the more intuitive and effective interface.

2.4.2 DOMAIN ANALYSIS

Human–robot interaction (HRI) is a subset of HCI and can be considered as one of the most important domains of the computer vision. Although a lot of works based on gesture recognition in the domain of HRI are known in the literature. Recent technological advances have opened new and challenging research horizons. In particular, controllers and sensors used for home entertainment can be exploited also as affordable devices supporting the design and implementation of new kinds of HRI.

The variety of hand and body gestures, compared with traditional interaction paradigms, can offer unique opportunities also for new and attractive forms of HCI. Thus, new gesture-based solutions have been progressively introduced in various interaction scenarios (encompassing, for instance, navigation of virtual worlds, browsing of multimedia contents, management of immersive applications, etc. The design of gesture-based systems will play an important role in the future trends of the HCI.

2.4.3 METHODOLOGY

The aim of this work is to create a human–robot interaction framework to allow a quadrotor both to perform autonomous navigation tasks (by completing path-following missions constituted by a sequence of pre-specified way-points) and to be controlled by user's body postures (for instance, when complex actions/ movements have to be performed). The main requisites needed to implement a system capable of controlling the aerial vehicle by means of user's posture are: (1) extracting spatial information from specific parts of the body (2) recognizing postures from this information (3) associating recognized postures to specific commands to be sent the quadrotor. In this work, the Microsoft Kinect is used as gesture tracking device; recognized postures are then used to control an Ar.Drone quadrotor platform (in the following of the paper the terms: Ar.Drone, quadrotor, and platform will be used interchangeably). The user is the ‘‘controller’’, and a new form of HRI can therefore be experienced. Interaction with quadrotors via Microsoft Kinect is not new. In particular, the ETH Zurich group proposes a way to dynamically interact with quadrotors based on the position of arms. The local 3D coordinates of the user's arms are mapped to the local 3D coordinates of the Quadrotor. In this way, a direct mapping between arms coordinates and quadrotor's position can be established.

2.4.4 ALGORITHM EXPLANATION

In the proposed framework, drifts are bounded by placing a certain number of tags on the floor. Each tag is placed at a well-defined position with respect to the WCS. When a tag enters in the camera field of view, the pose estimation system switches from the feature-based to the tag-based working mode, thus resetting the cumulated drifts. The ArToolKit library has been used to implement the tag-based pose estimation system.

The visual odometry algorithm, which is the base brick to estimate the position of the quadrotor, is shown. A thread is in charge to gather the frame coming from the vertical camera and to cope with the synchronization of the two pose estimation systems (feature-based and tag-based). Then, two threads run concurrently. The first thread searches for a visual marker in the frame, whereas the second thread calculates the correlations (matches) between features extracted from the current frame and features computed over a reference image. When a tag is identified, the absolute position of the camera is computed and values are forwarded to the feature-based pose estimation system to reset the drifts.

2.4.5 RESULT AND METRIC

Several tests have been performed to characterize the two pose estimation systems in order to evaluate the error during the localization process. Tests were aimed at measuring the error of the two separate systems and, afterwards, the error of the two systems working together as shown. Moreover, characterization tests were aimed at identifying the best setup of the environment; in particular, the size of the tags, the optimal flight altitude and the minimum number of floor features has been determined. All these parameters are strongly dependent on the quality of the vertical camera that, unfortunately, in the current setup provides very low resolution frames, thus limiting the maximum flight altitude.

2.4.6 CONCLUSION AND FUTURE WORK

This paper presents a framework for controlling the navigation of a quadrotor in GPS-denied environments. A NUI based on body gestures/postures allows the user to control the platform, whereas a hybrid visual odometry algorithm (based on both tag detection and features extraction) supports autonomous navigation.

Results presented in this paper showed how affordable devices such as the Microsoft Kinect are opening new scenarios allowing creating innovative forms of HRI unthinkable until a few months ago. The evolution of devices designed to implement novel user-centric forms of entertainment will provide researchers with alternative tools to re-design more intuitive, robust and fun HRI paradigms.

2.5 DEVELOPMENT OF HUMAN FOLLOWING MOBILE ROBOT SYSTEM USING LASER RANGE SENSOR – [6]

2.5.1 PROBLEM DEFINITION

This paper discussed a human following mobile robot system using laser range scanner. A human detecting algorithm is developed to detect the shins of target person. The mobile system can detect and follow the target person based on the positions of both shins. The mobile robot acts as an assistant that follows humans helping with carrying objects and luggage and the effectiveness of the system is demonstrated by several performance tests.

2.5.2 DOMAIN ANALYSIS

The Human machine interaction domain involves the active interface between the machine and the human. In this paper, the human detection method uses only a single range scanner to detect the waist of the targeted person. The human tracking system for an autonomous mobile robot using neural based motion detectors is proposed. This method uses the Kalman filtering scheme to estimate the location of moving robot. The mobile robot system that has the functions of human following and returning to the starting location autonomously while avoiding the obstacles using LRF and camera is developed.

2.5.3 METHODOLOGY

The methodology is based on the functioning of the web camera, sensor, control PC and a movement device.

First, it proposes a human detection method that uses only a single laser range scanner to detect the waist of the target person.

Second, owing to the limited speed of a robot and the potential risk of obstructions, a new human-following algorithm is proposed.

The speed and acceleration of a robot are adaptive to the human-walking speed and the distance between the human and robot.

Finally, the performance of the proposed control system is successfully verified through a set of experimental results obtained using a two-wheel mobile robot working in a real environment under different scenarios.

2.5.4 ALGORITHM EXPLANATION

The paper introduces a human following algorithm to be implemented in the robot’s sensor. Based on this, the cluster points are overlapped in the search circle. The position of the target person is estimated by the centre positions of two shin’s clusters which are obtained in the previous phase. These conditions are determined in consideration of the depth and width of cluster and step number of scan data.

Similarly, the position of the target person is estimated using both shin positions. A situation called as occlusion occurs where only one shin is detected. The system estimates the position of the hidden shin position based on the collection value.

The target speed and the direction of the mobile robot are obtained based on the centre positions of the person. The offset distance is set to 0.6m in consideration of the width of the mobile robot.

2.5.5 RESULT AND METRIC

Two performance tests were undertaken to clarify the effectiveness of the system. The first performance test resulted as the mobile robot following the target person who walks the rectangular course of 1.8m in width and 2.5m in depth. Four subjects participated in the experiment. The success rate of the performance test was 83%.

In the second test, three subjects participated and the mobile robot was made to follow the target through a narrow space in the room. The success rate was 66% and the reason for failure was that the robot misrecognized the leg of the desk to the person's shin.

2.5.6 CONCLUSION AND FUTURE WORK

In this paper, a human following mobile robot system using laser range scanner is developed. In order to detect a person a laser range scanner is very useful for getting the environment information around the robot. The developed human detecting algorithm enables the laser scanner to detect the shins of the targeted person. The scanning data of person's shins shows two parabola shapes of the convex below whether the person changes the standing position. The mobile robot can also follow the targeted person to move through a narrow space. The reason of the failure was that the mobile robot misrecognized the leg of the desk to the person's shin. In future work, it is intended to explore further the practicality of the system by various experimental scenarios. The same idea is expected to apply to the intelligent cargo transportation robot.

2.6 INDUSTRIAL IMPLEMENTATION OF KINECT

2.6.1 Robot control and online programming by human gestures using a Kinect motion sensor [25]

The working presents a concept and implementation of online programming, controlling and monitoring an industrial robot using a Kinect sensor as a HMI part. Presented innovative solution is based on human gestures. A communication interface for gestures analysis performed by the robot operator was created using LabVIEW applications. Communication between the kinetic motion sensor and the robot was carried out through PLC Siemens S7-300 inside a flexible manufacturing system. The connection between the logic controller and the robot controller R30iA was realized in the ProfibusDP network.

2.6.2 iRobot's AVA Tech Demonstrator [25]

AVA is short for ‘Avatar,’ although iRobot was careful not to call it a tele-presence robot so as not to restrict perceptions of what it's capable of. AVA is capable of fully autonomous navigation, relying on a Kinect-style depth sensing camera, laser rangefinders, inertial movement sensors, ultrasonic sensors, and (as a last resort) bump sensors. The entire sensor data crunching is taken care of by a heavyweight on-board computer, but the brains of the operation is really whatever AVA happens to be wearing for a head, in this case, a tablet PC.

Figure 2.3: iRobot AVA

2.6.3 Gesture recognition system for surgical robot's manipulation [10]

The working presents the application of a capture system gestures to manipulate two virtual surgical robots. The capture system gestures performed by a Kinect device which detects the movement of the user's hands in order to move the surgical robots, and in his right knee, used to change surgical instruments each robot. This system captures natural interface is tested in a surgical simulator for laparoscopic surgeries, which consists of a carrier endoscope robot is operated with a joystick, and the two surgical robots whose bodies terminals are led from the signals received by the Kinect device.

Figure 2.4: Surgical robot manipulation using Kinect

2.6.4 Usage of Kinect motion sensing technology in Game-Oriented technology [25]

The learning environment based on the KINECT Motion Sensing technology is able to fully mobilize the learners' multi-sensory organs, closely combine study with sports and enhance human-computer interactions, which can be conducive to the learners' health, greatly increase the relishes of learning and promote effective learning in the game, and finally compensate for the shortage of human computer interactions in the traditional mouse and keyboard mode. The article elaborates on the KINECT Motion Sensing Technology and its educational applications status by analysing its effective supports for game-oriented studying environment, based on which the article establishes a game-oriented learning environment. Eventually the article reveals an applicable case of game-oriented teaching and learning as a reference for related researches.

Figure 2.5: Kinect gaming

2.6.5 NAO robot using Kinect [27]

The methodology introduces three methods to imitate human upper body motion are implemented on a NAO humanoid robot: (i) direct angle mapping method (ii) inverse kinematics using fuzzy logic and (iii) inverse kinematics using iterative Jacobian. In the first method, the Kinect sensor is used to obtain coordinates of the shoulder, elbow and wrist joints of the operator. The four angles that are required to completely describe the position of the robot's wrist-shoulder roll, shoulder pitch, elbow roll and elbow yaw- are then calculated using vector algebra.

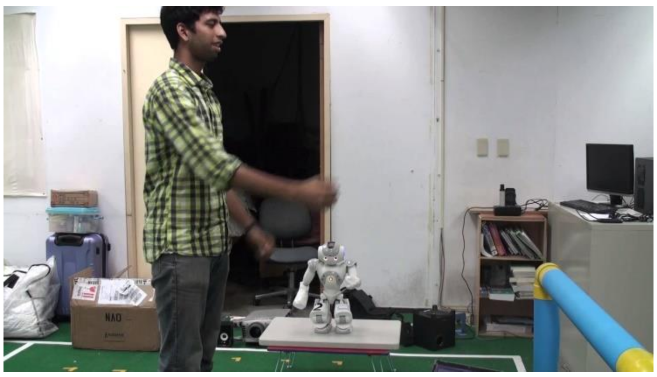


Figure 2.6: NAO robot control using Kinect

2.6.6 Master slave system using Kinect and Industrial robot for Tele-operations [25]

This involves the integration of a Kinect sensor device with a robotic arm Kuka KR6 as a master-slave communication for applications of Tele-operation and bioengineering. The Kinect device captures the position of the user's right hand, filters its resulting data and detects its movement using Haar wavelets. This data is then sent to the Kuka KR6 controller and it moves using forward kinematics. A series of tests were implemented to evaluate the accuracy of the slave system position compared to the user's right hand position using the Pearson correlation coefficient method. It resulted in a very strong correlation between them.

2.6.7 Application of gaming sensor in Forensic science [25]

The methodology includes the comparison versus well-established scanners (Faro and Trimble). The parameters used for the comparison are the quality in the fitting of primitives, a qualitative evaluation of facial data, the data quality for different ranges, and the accuracy in the measurement of different lengths. The results show that the Kinect noise level increases with range, from 5 mm at 1.5 m range to 15 mm at 3 m range. It is considered that for detail measurements the sensor must be placed close to the target. A general measurement of a sample crime scene was analysed. Errors in length measurements are between 2% and 10% for 3 m range. The measurement range must be limited to *c*. 3 m.

3.6.8 Kinect based real time obstacle detection for legged robots in complex environments [25]

Legged robots have different terrain traverse capabilities with ground vehicles. They can across steps and grooves, which are obstacles for ground vehicles. Some new challenges are posed for autonomous legged robot navigation in complex environments. In this paper, a Kinect based real time obstacle detection algorithm for legged robots is introduced. A graph based method is used to label all traversable areas connecting to a starting point. A spiral searching strategy is proposed to find the most ground-like point for graph based traversal. Sometime efficient rules are presented to increase the calculation efficiency. Experiments in both indoor and outdoor environment have been done and the results show that our algorithm can be effectively used in autonomous legged robots navigation.

2.6.9 KUKA robot control based Kinect image analysis [29]

This work has investigated the processes required for visual extracting and the remote control of KUKA KR-125 industrial robot manipulator. For this purpose, the robot controller communicates with the external system via an Ethernet cable IEEE 802.3. The exchanged data are transmitted thanks to TCP/IP Protocol. To do this, a client/server application with all relevant motions control is performed. Second, a Kinect in the robot proximity is set for the detection of objects (recognition of form, determination of position etc.) and finally applied it to a practical example: the robot is programmed to be able to stack object thanks to the reliability of the visual processing.

CHAPTER 3

ISSUES IN EXISTING SYSTEM

All the existing systems are based on various technologies and methodologies. Each concept makes use of different set of hardware and software which acts as a deciding factor on their functionality. The various issues present in these existing systems can be documented as follows:

3.1 LACK OF INNOVATIVE TECHNOLOGY

Human machine interaction has evolved through different generations from complex coding and easy handheld devices to hands-free gesture based devices. All the existing systems still use out-dated technologies that are based on complex controls and mechanism. Hence, depending on old methodologies will result in a drastic lag in performance rate as well as they will be considered inefficient. The proposed project tackles this issue with the usage of advanced gesture recognition technology of Microsoft’s Kinect device and implements the same in terms of a real time robotic project [1].

3.2 COMPLEX ALGORITHM

Most of the existing systems (like Human following robot based on laser range sensor and robot based on tether steering) involve complex algorithms and coding. Hence it results in poor understanding of the code and the algorithm used. These complex algorithms make it difficult for users to compile the code and detect errors accurately. Due to complicated tools and mechanisms involved, it requires an expert programmer to handle the product in case of any errors or changes to be implemented. Whereas the proposed project is based on a simple hand gesture recognition algorithm which is simple to use and modify as required. The product functionality is easy to understand by any casual user and doesn’t require any expert [3].

3.3 EFFECTIVE IDENTIFICATION OF USER

The existing systems like Human tracking robot using face detection (by Suzuki) and similar projects makes use of technologies that as a poor user recognition feature. There are cases where user is among the crowd or in a dark environment, in such cases the existing systems has problem in identifying the user. This issue is due to the usage of old camera technologies which doesn’t have features like low light imaging and high pixel cameras. This issue has to be tackled by using a much advanced camera that can capture much larger range of pixels and identify users efficiently. The Microsoft’s Kinect device in the proposed project has the ability to identify the users presence by detecting the entire body efficiently and respond to the instructions. It consists of a RGB camera as well to get the depth images [2].

3.4 SIZE OF ROBOT/MACHINE

Most of the existing systems based on Human Machine Interaction are based on large sized materials and equipment. Due to the large size users find it difficult to use it effectively in the environment. Thus, all the projects are advised to be compact in size as it will be more user-friendly and tension free. By introducing compact sized products a wider audience can be expected to accept the product for real time uses. In simple words, greater the size greater will be the complexity [5].

3.5 LIMITED OPERATIONAL TIME

The existing products that are based on mobile (movement) mechanism have a limited operational time as there are chances of operational power shortage. Even though the systems can be equipped with large power sources, the operational time is limited to the battery lifetime. Similarly, in remote environments the functioning of the product is limited to the range up to which the signals can be transferred [2].

3.6 OPERATIONAL DELAY AND RESPONSE TIME

Delay can be a major issue in most of the existing systems. All the systems show a significant delay during implementation of functions based on human gesture input in real-time. The existing systems functionality involves complex calculations and evaluation of the input data before actually implementing the output action. Thus a much intelligent systems is required that can detect the human gesture and implement the required robotic/machine action with a negligible delay [3].

3.7 ACCURACY RATE

All the existing systems have a poor accuracy in detecting the user structure and body gestures effectively. Even though the devices can detect the user gesture inputs, the real time implementation on the machine or robot will not be accurate. These devices don't have the tendency to detect the skeletal structure of the user and convert those data into respective gesture inputs. Such methodology can help in controlling robots with better accuracy rate. The proposed project makes use of Kinect's advanced gesture technology that detects the skeletal structure automatically and generates the gesture input [6].

3.8 NO HANDS-FREE CONTROL

Most of the existing systems are based on complex controllers (like glove based sensors, complex joy pads, handheld gesture stick etc.). all these controllers requires users to handle them as required to give the necessary instructions to the system for performing an operation. This issue can be tackled by making use of advanced gesture technology that allows users to have a hands-free control over the system. Thus, users don't have to depend on controller's usage or device direction movement guidelines or chances of damage to the wired connections in the controller. Users can provide the input based on body gestures and these gestures can be easily identified for their respective instructions based on the code uploaded. [4]

3.9 REQUIREMENT OF COMPLEX PROGRAMMING SKILLS

The existing systems make use of complex programming software and tools that requires high level programming skills. Hence general users will find it more difficult to understand the code or make any changes as per the requirement. Only expert users can perform such operations. Thus issue arises when errors or invalid output is generated where users will be unaware of the reasons. The proposed system will tackle the problem as it implements a simple logic and is more user friendly [5].

3.10 COST

Most of the existing systems use expensive devices for developing the interface between the human and machine. These devices tend to perform the required functionalities at a higher cost which forces general users to look for alternative solutions. In today’s technology, there are multiple devices that can perform the same functionalities at a comparatively cheaper rate like the new Kinect device. Such devices will make the overall project more economical and user friendly [1][2].

CHAPTER 4

KINECT BASICS

4.1 INTRODUCTION

The Kinect was launched on November 4, 2010 and sold an impressive 8 million units in the first 60 days, entering the Guinness World Records as the “fastest selling consumer electronics device in history”. The Kinect was the first commercial sensor device to allow the user to interact with a console through a natural user interface (using gestures and spoken commands instead of a game controller).

Figure 4.1: Microsoft Kinect for Xbox 360

Figure 4.2: Kinect AC adapter with standard USB connection

4.2 OFFICIAL FRAMEWORKS

In 2010, PrimeSense (the company behind Kinect's 3D imaging) released its own drivers and programming framework for the Kinect, called OpenNI. Soon after that, it announced a partnership with ASUS in producing a new Kinect-like device, the Xtion. In 2011, Microsoft released the non-commercial Kinect SDK (software development kit). In February 2012, it released a commercial version, accompanied by the Kinect for Windows device. Laptops with integrated Kinect-like cameras are most probably on their way. This is only the beginning of a whole range of hardware and applications using the technology behind Kinect to make computers better understand the world around them.

4.3 THE KINECT SENSOR

The Kinect sensor features an RGB camera, a depth sensor consisting of an infrared laser projector and an infrared CMOS sensor, and a multi-array microphone enabling acoustic source localization and ambient noise suppression. It also contains an LED light, a three-axis accelerometer, and a small servo controlling the tilt of the device.

The infrared CMOS (complementary metal–oxide semiconductor) sensor is an integrated circuit that contains an array of photo detectors that act as an infrared image sensor. This device is also referred to as IR camera, IR sensor, depth image CMOS, or CMOS sensor, depending on the source.

The project will focus on the 3D scanning capabilities of the Kinect device accessed through OpenNI/NITE. The RGB camera is an 8-bit VGA resolution (640 x 480 pixels) camera. The two depth sensor elements, the IR projector and IR camera, work together with the internal chip from PrimeSense to reconstitute a 3D motion capture of the scene in front of the Kinect.

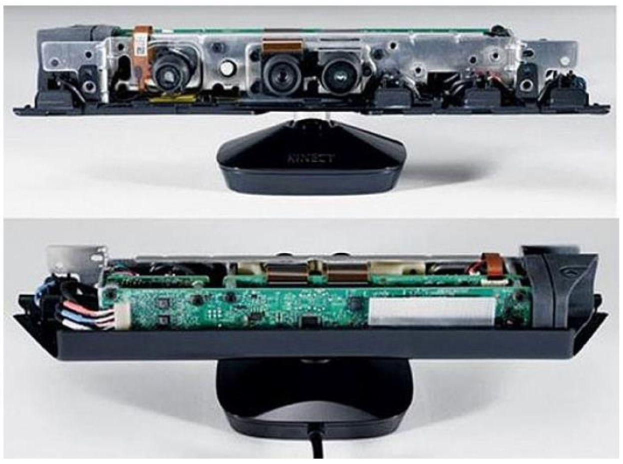


Figure 4.3: Left to Right: Kinect IR camera, RGB camera, LED and IR projector

4.4 POSITIONING THE KINECT

The Kinect’s practical range goes from 1.2m to 3.5m. If the objects stand too close to the sensor, they will not be scanned and will just appear as black spots; if they are too far, the scanning precision will be too low, making them appears as flat objects. If the Kinect for Windows device is used, the range of the camera is shorter, from 40cm to 3m.

4.5 KINECT ARCHITECTURE AND HARDWARE INTERFACE

The Kinect works as a 3D camera by capturing a stream of collared pixels with data about the depth of each pixel. Skeleton tracking is generally handled by the SDK with gesture recognition left to the developer, though multiple libraries exist to aid in recognition of gestures. In addition, speech recognition is done by external SDKs such as the Microsoft Speech Platform.

The Kinect sensor has the following components:

- An **RGB Camera** that stores three channel data in a 1280x960 resolution at 30Hz.
- An **infrared (IR) emitter** and an **IR depth sensor** used for capturing depth image.

- An array of **four microphones** to capture positioned sounds.
- A **tilt motor** which allows the camera angle to be changed without physical interaction and a **three-axis accelerometer** which can be used to determine the current orientation of the Kinect.

4.6 KINECT CAPABILITIES

Once the Kinect is properly installed and communicating with the computer, the system will be able to access a series of raw data streams and other capabilities provided by specific middleware. The project will make use of the OpenNI drivers and NITE middleware.

4.7 DEPTH MAP

The depth map is the result of the operations performed by PrimeSense's PS1080 chip on the IR image captured by Kinect's IR CMOS. This VGA image has a precision of 11 bits, or 2048 different values, represented graphically by levels of grey from white (2048) to black (0).

The sensor's distance measurement doesn't follow a linear scale but a *logarithmic scale*. Without getting into what a logarithmic scale is the precision of the depth sensing is lower on objects further from the Kinect sensor.

4.8 HAND AND SKELETON TRACKING

After the depth map has been generated, it can be used for applications or run it through a specific middleware to extract more complex information from the raw depth map.

NITE middleware is used to add hand/skeleton tracking and gesture recognition to the hand.

4.9 KINECT DRIVERS AND FRAMEWORK

In order to access the Kinect data streams, the necessary drivers have to be installed on the computer. The drivers can be downloaded for free from the

Microsoft's webpage for Kinect downloads although there are many versions available with respect to the different OS and specifications.

4.10 PRIMESENSE: OpenNI AND NITE

The project will use OpenNI and NITE to access the Kinect data streams and the skeleton/hand tracking capabilities The **c**ompany **PrimeSense** developed the technology behind Kinect's 3D imaging and worked with Microsoft in the development of the Kinect device. In December 2010, PrimeSense created an industry-led, not-for-profit organization called OpenNI, which stands for **open natural interaction** (http://www.openni.org). This organization was formed to "certify and promote the compatibility and interoperability of natural interaction (NI) devices, applications, and middleware." The founding members of the OpenNI organization are PrimeSense, Willow Garage, Side-Kick, ASUS, and AppSide.

4.10.1 OpenNI

In order to fulfill its goal, OpenNI released an open source framework called OpenNI Framework. It provides an API and high-level middleware called NITE for implementing hand/skeleton tracking and gesture recognition.

4.10.2 NITE

For natural interaction to be implemented, the developer needs more than the 3D point cloud from Kinect. The most useful features come from the skeleton and hand tracking capabilities. Not all developers have the knowledge, time, or resources to develop these capabilities from scratch, as they involve advanced algorithms. PrimeSense decided to implement these capabilities and distribute them for commercial purposes but keep the code closed, and so NITE was developed.

The major differences between OpenNI and NITE are OpenNI is PrimeSense's framework; it allows users to acquire the depth and RGB images from the Kinect. OpenNI is open source and for commercial use. NITE is the middleware that allows users to perform hand/skeleton tracking and gesture recognition. NITE is not open source, but it is also distributed for commercial use. Develop middleware that processes the OpenNI point cloud data and extracts the joint and gesture information. Without a doubt, there will be other middleware developed by third parties in the future that will compete with OpenNI and NITE for natural interaction applications.

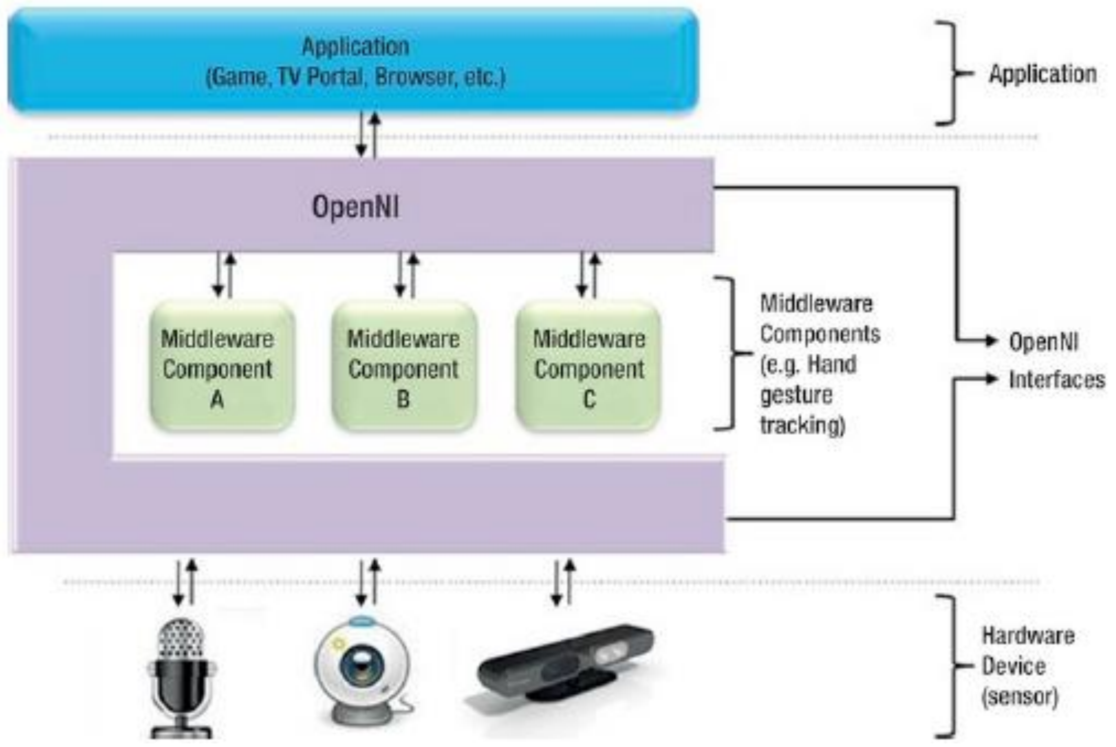


Figure 4.4: OpenNI abstract layered view

4.10.3 MICROSOFT KINECT FOR WINDOWS

Microsoft announced the release of the official Microsoft Kinect SDK for non-commercial use. This SDK offered the programmer access to all the Kinect sensor capabilities plus hand/skeleton tracking. At the time of writing, Kinect for Windows SDK includes the following:

• Drivers for using Kinect sensor devices on a computer running Windows 7 or Windows 8 developer preview (desktop apps only)
• APIs and device interfaces, along with technical documentation
• Source code samples

Unfortunately, the non-commercial license limited the applications to testing or personal use. Also, the SDK only installs on Windows 7, leaving out the Linux and Mac OSX programmer communities. Moreover, the development of applications is limited to C++, C#, or Visual Basic using Microsoft Visual Studio 2010. From February 2012, the Kinect for Windows includes a new sensor device specifically designed for use with a Windows-based PC and a new version of the SDK for commercial use. The official Microsoft SDK will continue to support the Kinect for Xbox 360 as a development device.

4.11 Structured-Light 3D Scanning

Most structured-light scanners are based on the projection of a narrow strip of light onto 3D object, using the deformation of the stripe when seen from a point of view different from the source to measure the distance from each point to the camera and thus reconstitute the 3D volume. This method can be extended to the projection of many stripes of light at the same time, which provides a high number of samples simultaneously.

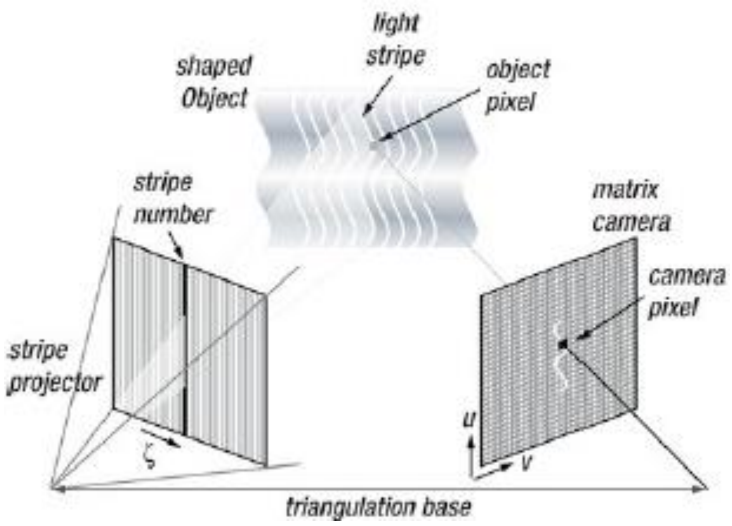


Figure 4.5: Triangulation principle for structured-light 3D scanning

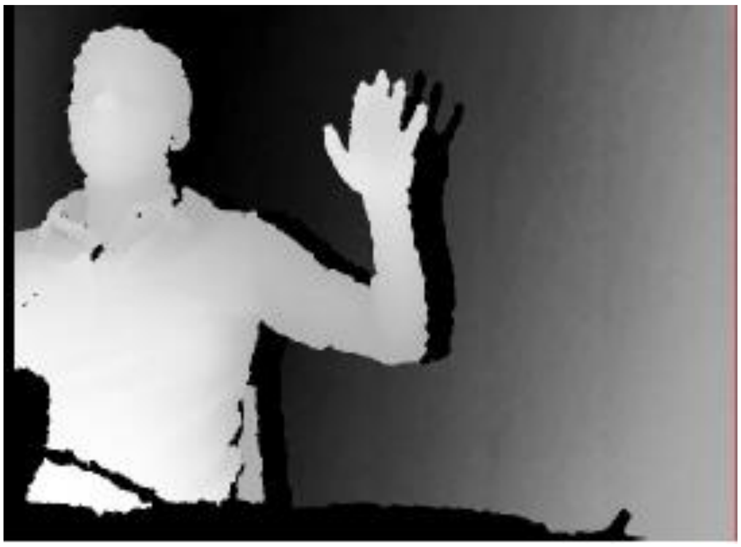


Figure 4.6: Depth map

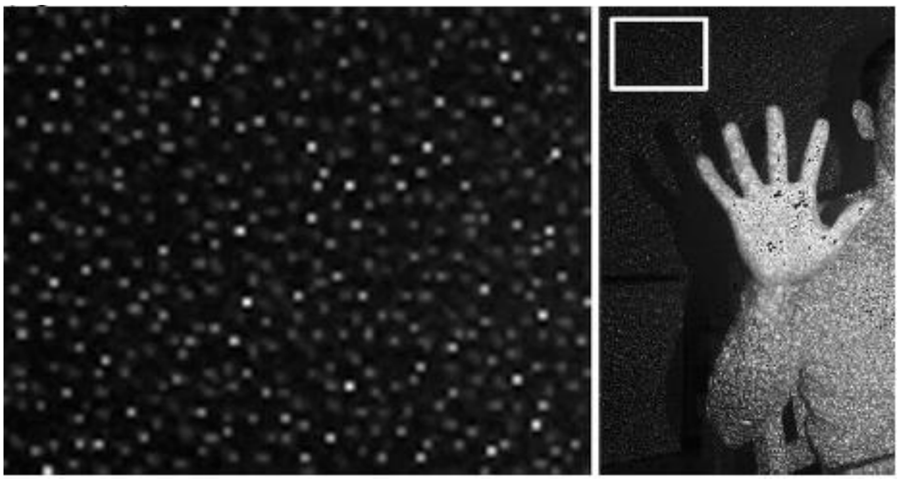


Figure 4.7: Kinect IR coding image

4.12 Converting the light coding image to a depth map

Once the light coding infrared pattern is received, PrimeSense's PS1080 chip compares that image to a reference image stored in the chip's memory as the result of a calibration routine performed on each device during the production process. The comparison of the "flat" reference image and the incoming infrared pattern is translated by the chip into a VGA sized depth image of the scene that users can access through the OpenNI API.

CHAPTER 5
PROCESSING IDE

5.1 INTRODUCTION

Processing is an open source computer programming language and IDE suitable for working on images, animations and gesture based interactions.

Processing has different programming modes to make it possible to deploy sketches on different platforms and program in different ways.

- Java mode is the default
- Other programming modes may be downloaded by selecting "Add Mode..." from the menu in the upper-right corner of the PDE.

Processing Development Environment (PDE) consists of:

- Simple text editor for writing code
- Message area
- Text console
- Tabs for managing files
- Toolbar with buttons for common actions and
- Series of menus

SKETCHES: All Processing projects are called sketches. Each sketch has its own folder. The main file for each sketch has the same name as the folder and is found inside.

COORDINATES: Processing uses a Cartesian coordinate system with the origin in the upper-left corner. If the sketch is 320 pixels wide and 240 pixels high, coordinate (0, 0) is the upper-left pixel and coordinate (320, 240) is in the lower-right.

5.2 INSTALLING PROCESSING

Processing can be installed from the link http://www.processing.org and the appropriate version has to be downloaded according to the OS. The Windows version of the software will include the Java Development Kit along with it. Upon finishing the installation, Processing IDE can be used for programming. To include new libraries, the downloaded library files can be copied to the Processing folder in documents or else the libraries can be directly downloaded and installed from the IDE settings menu.

5.3 PROCESSING VARIABLES

Variables are symbolic names used to store information in a program. There are eight primitive data types in Java, and they are all supported in Processing: byte, short, int, long, float, double, Boolean, and char. Integers (int) will be used to store positive and negative whole numbers (this means numbers without a decimal point, like 42, 0, and -56. An integer variable can store values ranging from - 2,147,483,648 to 2,147,483,647 (inclusive). If there is a need to store larger numbers, variable has to be defined as long.

The data type double works the same way as float, but it is more precise. In general, Processing encourages the use of floats over doubles because of the savings in memory and computational time. The data type Boolean stores two values: true and false. It is common to use Booleans in control statements to determine the flow of the program. The character data type, or char, stores typographic symbols (a, U, $).

5.4 STRUCTURE OF A PROCESSING SKETCH

A timer variable has to be declared in Processing IDE outside of any function. These variables defined like this are treated as global variables in Processing, so they can be called from any method and subclass.

int timer;

5.4.1 SETUP() FUNCTION

This function is called in the program (unless it is called explicitly from another function). For this project, the size of sketches has to be specified within this function.

```
void setup(){
size(800,600);}
```

5.4.2 DRAW() FUNCTION

Next, a draw() function has to be included that will run as a loop until the program is terminated by the user. A circle has to be drawn on screen that will move to the right of the screen as time passes. A background() function has to be declared to clear the frame with a color at every iteration.

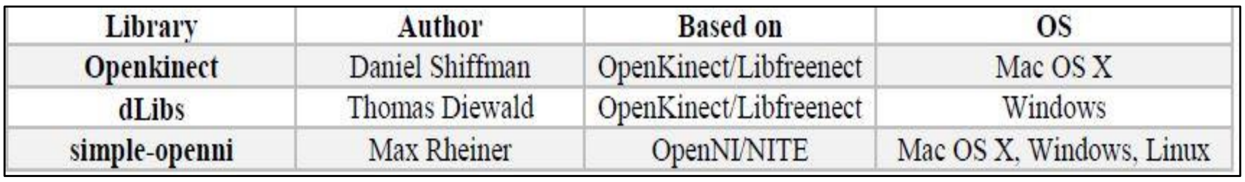
```
void draw() {
background(255);
ellipse(timer, height/2, 30, 30);
timer = timer + 1;
}
```

5.5 PROCESSING LIBRARIES

Processing libraries are lines of Java code that have been packaged into a .jar file and placed in the libraries folder in the system. There are two sorts of libraries: core libraries, like OpenGL and Serial, which are included with Processing when it is downloaded, and contributed libraries, which are created and maintained by members of the Processing community. Contributed libraries need to be downloaded and stored in the libraries folder in the system.

For Mac OS X users and the Processing application is saved in the Applications folder:

/Applications/Processing/Contents/Resources/Java/modes/java/libraries to open the Processing application on Mac OS X, right-click the icon and select "Show Package Contents" from the pop-up menu.



Library	Author	Based on	OS
Openkinect	Daniel Shiffman	OpenKinect/Libfreenect	Mac OS X
dLibs	Thomas Diewald	OpenKinect/Libfreenect	Windows
simple-openni	Max Rheiner	OpenNI/NITE	Mac OS X, Windows, Linux

TABLE 5.1: KINECT LIBRARIES

5.6 SIMPLE-OpenNI

Simple-OpenNI is a NITE and OpenNI wrapper for Processing implemented by Max Rheiner, a media artist and software developer currently lecturing at Zurich University. NITE and OpenNI are implemented in C++, and Rheiner has wrapped all the functions and methods in such a way that they are accessible from Java. Then a library was developed to make the Java code available for Processing.

5.6.1 INSTALLING SIMPLE-OpenNI

There are some specific prerequisites for the library to work on the computer. After installing the prerequisites described, users can go to the download web site at http://code.google.com/p/simple-openni/ downloads/list to download the library and install it in the Processing libraries folder. Installing Simple-OpenNI requires the previous installation of OpenNI and NITE on user's machine. A detailed description on performing this operation is available at the Simple-OpenNI web site: http://code.google.com/p/simple-openni/.

5.6.2 INSTALLATION ON WINDOWS

On Windows machine, the following links can be used for downloading OpenNI and NITE:

Download and install OpenNI.

http://www.openni.org/downloadfiles/opennimodules/openni-binaries/20-latest-unstable

Download and install NITE.

http://www.openni.org/downloadfiles/opennimodules/openni-compliant-middleware-binaries/

Download and install Primesensor.

http://www.openni.org/downloadfiles/opennimodules/openni-compliant-hardware-binaries/

Install the Kinect driver.

Download SensorKinect, unzip it, and open the file at

/avin2/SensorKit/SensorKinect-Win-OpenSource32-?.?.?.msi

https://github.com/avin2/SensorKinect

5.7 ACCESSING THE DEPTH MAP AND RGB IMAGE

After installing the Simple-OpenNI Processing library, a simple example that will access the data streams from Kinect from Processing can be developed. First, import the Simple OpenNI library, and declare the variable kinect that will contain the Simple-OpenNI object, like so:

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
```

Within setup(), initialize the kinect object, passing this as an argument. The Kinect has a standard RGB camera and an infrared camera on board, used in combination with an infrared projector to generate the depth image for 3D scanning. Enable the depth map and RGB images in the Kinect object and the mirroring capability that will mirror the Kinect data so it’s easier to relate to user’s image on screen. Then, set the sketch size to the total size of the RGB and depth images placed side by side, so that both can fit in the frame.

```
void setup() {
kinect = new SimpleOpenNI(this);
// enable depthMap and RGB image
kinect.enableDepth();
kinect.enableRGB();
// enable mirror
kinect.setMirror(true);
size(kinect.depthWidth()+kinect.rgbWidth(), kinect.depthHeight());
}
```

In the draw loop the Kinect object has to be updated so that the latest data is received from the Kinect device. Next, display the depth image and the RGB images on screen.

```
void draw() {
kinect.update();
// draw depthImageMap and RGB images
image(kinect.depthImage(), 0, 0);
image(kinect.rgbImage(),kinect.depthWidth(),0);
}
```

5.8 PROCESSING IN 3D

There are two 3D primitives already implemented in Processing, box() and sphere(). Users can draw a three-dimensional box on screen. For this, import the OpenGL Processing core library into the sketch so user's can set the renderer to OPENGL in the size() function within setup(). Then, in the draw() loop, set the background to white, move to the centre of the screen, and draw a cube of 200 units each side.

```
import processing.opengl.*;
void setup()
{
size(800, 600, OPENGL);
}
void draw()
{
background(255);
noFill();
translate(width/2,height/2);
box(200);
}
```

5.9 HAND TRACKING

Hand tracking is a very useful feature for natural interaction. The hands can be precisely positioned in space, and a vast amount of gestures can be performed with them. These abilities can be used to drive pakinectrameters or trigger different behaviours in the natural interaction applications. OpenNI/NITE includes a framework for hand tracking and hand gesture recognition that can be used off the shelf. This is a great feature that doesn't exist in the official Microsoft SDK, at least in the beta version.

To take advantage of this, a series of functions or methods has to be included in the main Processing sketch that will be called from the Simple-OpennNI library when certain events are triggered.

Some of the methods and their applications are:

The **onCreateHands()** method is called when a hand is detected and a hand object is initialized. An integer containing the hand ID is obtained, a PVector defining its position on recognition, and a float with the timestamp value.

```
void onCreateHands(int handId, PVector pos, float time){
}
```

The function **onUpdateHands()** is called every time Kinect updates the hand data. A new position vector and time for the hand is obtained.

```
void onUpdateHands(int handId, PVector pos, float time){
}
```

When a hand is destroyed (when it disappears from screen or is not recognizable any more), the
function **onDestroyHands()** will be invoked.

```
void onDestroyHands(int handId, float time){
}
```

NITE also calls two methods related to gesture recognition. The **onRecognizeGesture()** function is called when NITE recognizes a gesture. The string strGesture contains the name of the gesture
recognized (Wave, Click, or RaiseHand).

```
void onRecognizeGesture(String strGesture, PVector idPosition, PVector endPosition){
}
```

Finally, the function **onProgressGesture()** is triggered while the gesture is in progress but before it has been recognized. It gives the percentage of completion of the gesture, along with the gesture type and the current position of the hand. This is normally a very short period of time, but it can be useful for some applications.

```
void onProgressGesture(String strGesture, PVector position,float progress){
}
```

5.10 SKELETON TRACKING

Skeleton tracking is probably the most impressive of NITE's, and therefore of Kinect's capabilities. This framework allows the computer to understand a person's body position in 3D and to have a quite accurate idea of where the person's joints stand in space at every point in time. This is quite an overwhelming feature that eliminates the need of advanced cameras, suits, wearable etc. The only downside of skeleton tracking (for certain projects) is the need for the user to stand in a certain "start pose" to allow the middleware to detect the user's limbs and start the tracking. This inconvenience seems to have been fixed in the current version of the official Microsoft SDK.

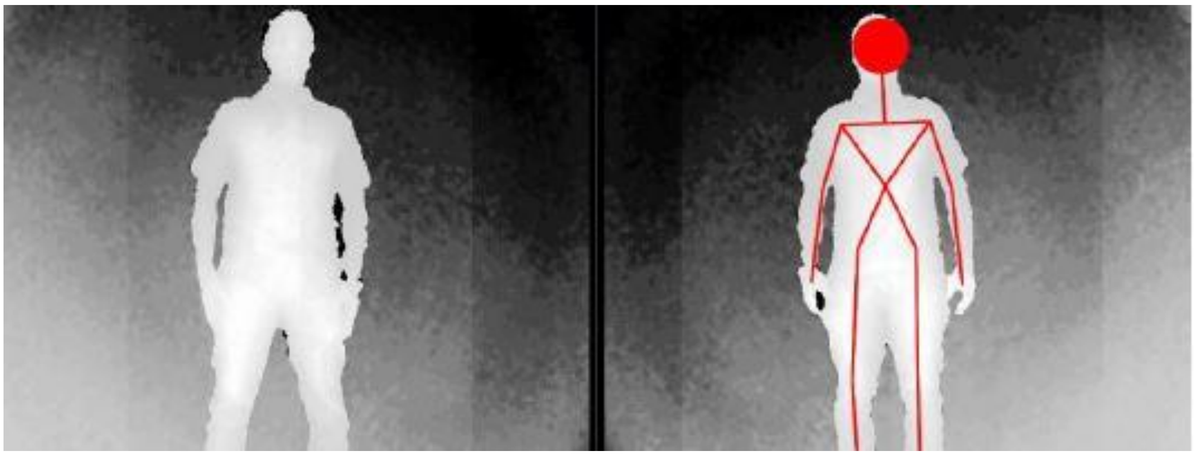


Figure 5.1: Skeletal tracking of user

When a new user is recognized the function **onNewUser( )** is called. The user hasn't been "skeletonized" yet, just detected, so the pose detection routine has to be started if the user's limbs has to be tracked.

```
void onNewUser(SimpleOpenNI curContext, int userId){
println("New User Detected - userId: " + userId);
// start tracking of user id
curContext.startTrackingSkeleton(userId);
}
```

The function **onLostUser( )** is invoked when Simple-OpenNI isn't able to find the current user. This usually happens if the user goes off-screen for a certain amount of time.

```
void onLostUser(SimpleOpenNI curContext, int userId){
// print user lost and user id
println("User Lost - userId: " + userId);
}
```

The function **onVisibleUser ( )** marks the beginning of the calibration process, which can take some seconds to perform (Called when a user is tracked).

```
void onVisibleUser(SimpleOpenNI curContex
{
// print user lost and user id
println("visible - userId: " + userId); }
```

CHAPTER 6

PROPOSED SYSTEM

6.1 OBJECTIVE OF PROPOSED SYSTEM

- To build a ROBOTIC ARM that can be controlled using MICROSOFT KINECT's gesture technology.
- To develop and test the robotic arm's application to handle real-time objects.

6.2 ARCHITECTURE

The proposed architecture consists of Kinect device, robotic arm, Arduino board and PC. Kinect device is connected to the computer system via USB and the computer system is connected to the Arduino board.

6.2.1 SYSTEM BLOCK DIAGRAM

The proposed method involves Kinect device connected to the computer system (via USB) and the computer system is connected to the Arduino board. The Microsoft Kinect device is programmed using Processing IDE to accept the gesture inputs from the user and these skeleton data are converted to Arduino accepted data (angles) which is sent to the Arduino board via serial port.

The general architecture consists of Microsoft's Kinect device acting as the input device that will receive the use gesture inputs. The PC processes the skeletal data and based on the coding in the respective IDE the skeletal data is converted to arm servo rotation data. Using this data the angle among the joints are found. The calculated angles are passed on through the serial communication using which servos move. The Arduino microcontroller used is Arduino MEGA 2560. This microcontroller consists of 54 digital I/O pins, 16 analogue inputs and supports complex projects easily.

6-axix control palletizing robot arm model: A robotic arm is required to perform the desired gesture action that is given as input using Kinect. The 6-axis model provides more precise bending and movement like a human hand.

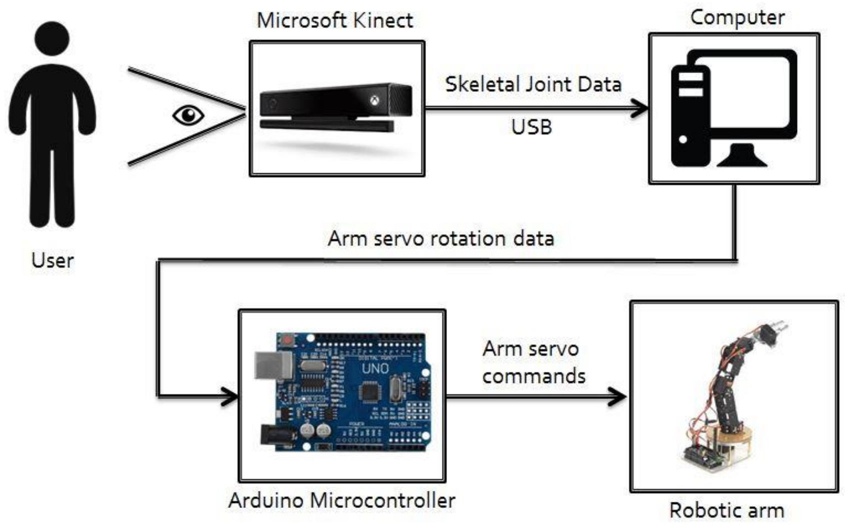


Figure 6.1: General architecture

6.2.2 SERVOS

The movements of Arm will be based on the rotation of six servos. Servos are DC motors that include a feedback mechanism that allows keeping track of the position of the motor at every moment. The way users control a servo is by sending it pulses with a specific duration. Generally, the range of a servo goes from 1 millisecond for 0 degrees to 2 milliseconds for 180 degrees. There are servos that can rotate 360 degrees.

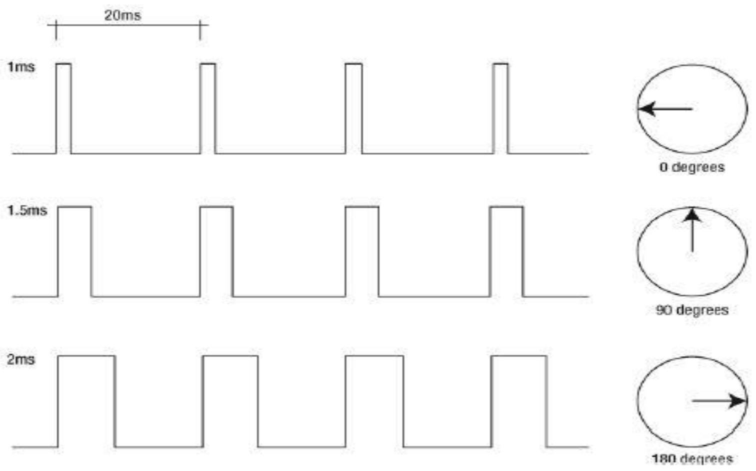


Figure 6.2: Movement graph of servos

The following is the code needed for using the Servo library. First, import the Servo library and declare the servo object.

```
#include <Servo.h>
Servo servo1;
```

In setup(), use the Servo.attach() function to set the servo to pin 5.

```
void setup() {
servo1.attach(5);
}
```

And in loop(), read the potentiometer value and map its values to a range of 0-180. Finally, write the angle value to the servo and add a delay to allow the servo to reach the new position.

```
void loop()
{
int angle = analogRead(0);
angle = map(angle, 0, 1023, 0, 180);
servo1.write(angle);
delay(15);
}
```

6.2.3 ROBOTIC ARM ARCHITECTURE

The robotic arm is a rotating multiple-dimensional model. The elongation distance of robotic arm is range from 50mm to 450mm. The torque force of a single servo is up to 30KG, and the load capacity of bottom joint is up to 500g. The rotating platform brings more stabilization and precision to the arm, the drive board makes it more convenient to control the arm.

6-axix control palletizing robot arm model: A robotic arm is required to perform the desired gesture action that is given as input using Kinect. The 6-axis model provides more precise bending and movement like a human hand.

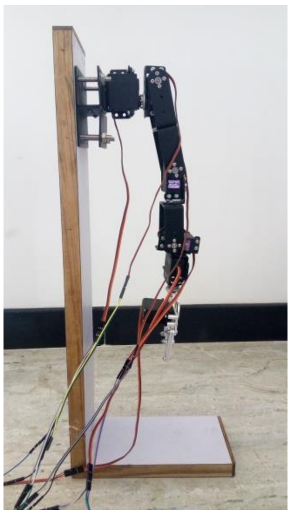


Figure 6.3: Robotic arm

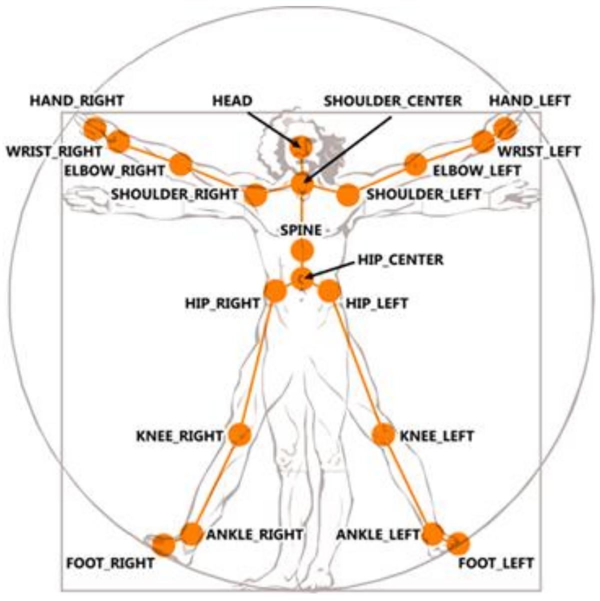
MEGA 2560 R3: MEGA 2560 is an update to Arduino Mega, with greater number of I/O pins to implement more connections and contains everything needed to support the microcontroller.

6.3 MODULES DESCRIPTION

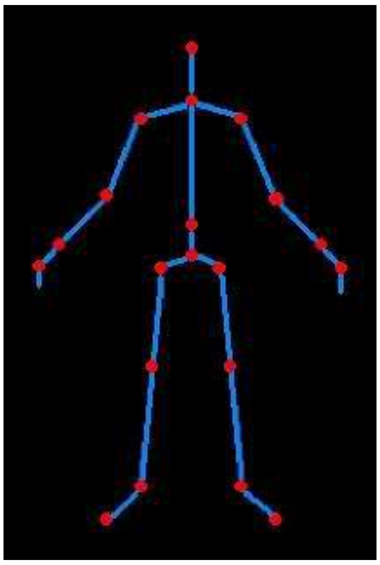
6.3.1 INPUT MODULE: MICROSOFT'S KINECT

The Kinect device acts as the input module of the project. It works by capturing the user gesture inputs and performs the conversion of the gesture data to skeletal data. The sensor interface with the PC via a standard USB 2.0 port; however an additional power supply is needed because the USB port cannot directly support the sensor's power consumption.

The Kinect device is static and the movement of the body to be detected along the camera axis is constrained to a range. First foreground segmentation takes place followed by extended distance transform computation and arm fitting. Also, the RGB camera captures image that is used for face & upper body detection along with skin segmentation.



(a)



(b)

Figure 6.4: (a) Skeletal Joint Positions (b) Skeletal joint data recognized by Kinect device

6.3.2 PROCESSING AND ARDUINO

Processing is an open source computer programming language and IDE suitable for working on images, animations and gesture based interactions.

Arduino software (IDE) is an open source software designed to work with Arduino boards and Arduino based projects.

6.3.2.1 CONNECTING KINECT TO PROCESSING

Requirements:

- Processing environment and complier
- OpenKinect libraries
- Open NI libraries
- Kinect sensor

Steps:

- Installing Processing and OpenKinect Libraries
- Building Processing application

- Building depth tracking function
- Building hand tracking application

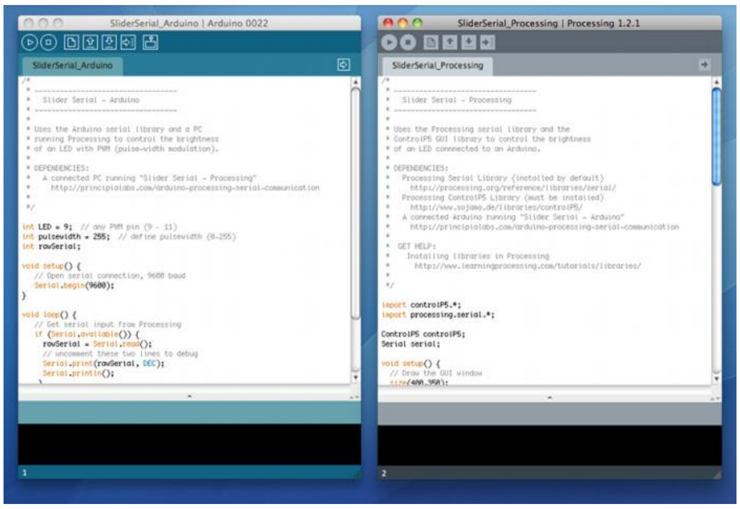
6.3.2.2 COMMUNICATION BETWEEN PROCESSING AND ARDUINO

Processing:

- Import Serial library and declare global Serial object.
- In setup() method, initialise serial communication on the port.
- In draw() loop, whatever data needed can be sent over the serial port using write() method.

Arduino:

- In start(), start the serial communication on the same port.
- In loop() method, receive the incoming serial data i.e arm servo data and activate the servo motors as required.


SliderSerial_Arduino | Arduino 0022
SliderSerial_Arduino
SliderSerial_Processing | Processing 1.2.1
SliderSerial_Processing

```
/*
 * ----------------------------------
 *   Slider Serial - Arduino
 * ----------------------------------
 *
 * Uses the Arduino serial library and a PC
 * running Processing to control the brightness
 * of an LED with PWM (pulse-width modulation).
 *
 * DEPENDENCIES:
 *   A connected PC running "Slider Serial - Processing"
 *     http://principialabs.com/arduino-processing-serial-communication
 *
 */

int LED = 9;  // any PWM pin (9 - 11)
int pulsewidth = 255;  // define pulsewidth (0-255)
int rawSerial;

void setup() {
  // Open serial connection, 9600 baud
  Serial.begin(9600);
}

void loop() {
  // Get serial input from Processing
  if (Serial.available()) {
    rawSerial = Serial.read();
    // uncomment these two lines to debug
    Serial.print(rawSerial, DEC);
    Serial.println();
```

```
/*
 * ----------------------------------
 *   Slider Serial - Processing
 * ----------------------------------
 *
 * Uses the Processing serial library and the
 * ControlP5 GUI library to control the brightness
 * of an LED connnected to an Arduino.
 *
 * DEPENDENCIES:
 *   Processing Serial Library (installed by default)
 *     http://processing.org/reference/libraries/serial/
 *   Processing ControlP5 Library (must be installed)
 *     http://www.sojamo.de/libraries/controlP5/
 *   A connected Arduino running "Slider Serial - Arduino"
 *     http://principialabs.com/arduino-processing-serial-communication
 *
 *  GET HELP:
 *    Installing libraries in Processing
 *      http://www.learningprocessing.com/tutorials/libraries/
 *
 */

import controlP5.*;
import processing.serial.*;

ControlP5 controlP5;
Serial serial;

void setup() {
  // Draw the GUI window
  size(400,350);
```

Figure 6.5: Screenshots of Processing IDE and Arduino IDE

6.3.3 IMPLEMENTATION MODULE

This is the final module where the outcome is generated. The final outcome is determined by how the robotic arm is responding to the user gesture inputs and whether the actions of the robotic arm are accurate. A negligible delay will be noted however the robotic arm will have a greater response rate.

The final steps involve:

- Power the drive board and upload the code into MEGA
- Build the robotic arm and initialise position of servos
- Limit the range of every joints in case of overload

6.3.3.1 BUILDING THE CIRCUIT

The circuit that is going to be prepared should to be plugged into the Arduino board and is designed to control six servo motors. The six servo motors are connected to the respective pins in the Arduino board say 2, 5, 6, 7, 8 and 9. To the breadboard positive and negative wires from the power supply are connected in the following way.

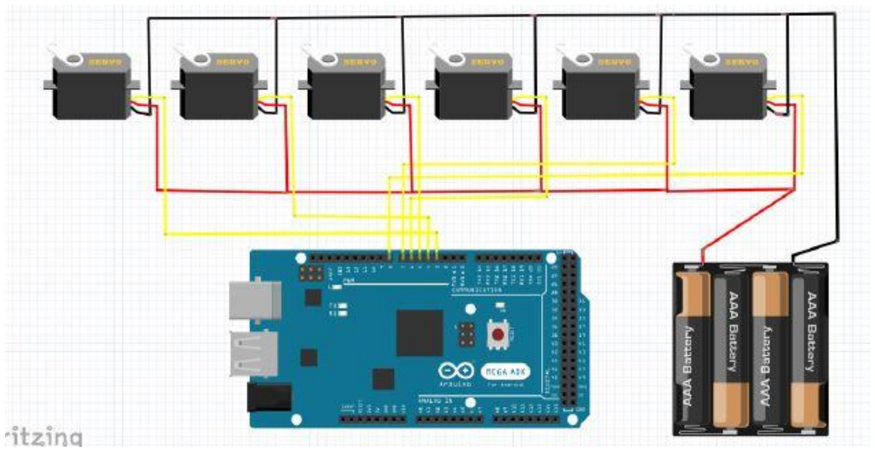


Figure 6.6: Servos and Arduino connection circuit

6.3.3.2 SETTING THE SERVOS TO THE STARTING POSITION

Reconnect Arduino, batteries and check that all servos move smoothly. After this, all servos should be assigned an starting position. All shoulder and leg servos are assigned to 500 (0 degrees) or 2500 (180 degrees) depending if they are on the left side or right side, and the elbows to 1500 (90 degrees), allowing the movement in two directions. The last thing is to position the servo that will rotate the Arm to 1500 (90 degrees).

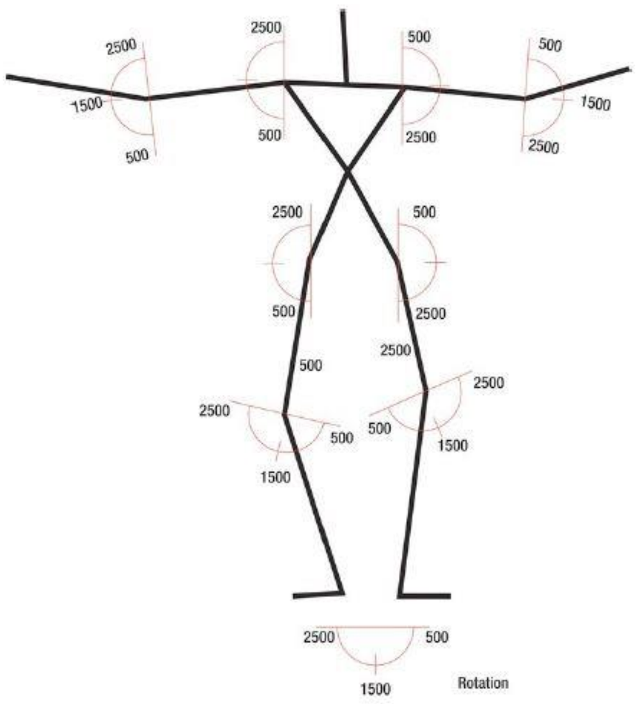


Figure 6.7: Starting position of the joints in the body

6.3.3.3 SKELETON TRACKING ON-SCREEN

Simple-OpenNI provides a nice interface for skeleton tracking and a couple of callback functions to which can be added to the code. This can be started by importing and initializing Simple-OpenNI.

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
// import the processing serial library
import processing.serial.*;
// and declare an object for our serial port
Serial port;
```

The **drawSkeleton( )** function from one of the examples in Simple-OpenNI is used. It takes an integer as a parameter, which is the user ID This function runs through all the necessary steps to link the skeleton joints with lines, defining the skeleton that will be seen on screen. The function makes use of the public method **drawLimb( )** from the Simple-OpenNI class.

```
void drawSkeleton(int userId){
```

```
// get 3D position of head
kinect.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_HEAD,headPosition);
// convert real world point to projective space
kinect.convertRealWorldToProjective(headPosition,headPosition);
// create a distance scalar related to the depth in z dimension
distanceScalar = (525/headPosition.z);
// draw the circle at the position of the head with the head size scaled by the
distance scalar
ellipse(headPosition.x,headPosition.y,
distanceScalar*headSize,distanceScalar*headSize);
//draw limb from head to neck
kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD,
SimpleOpenNI.SKEL_NECK);
//draw limb from neck to left shoulder
kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_LEFT_SHOULDER);
//draw limb from left shoulde to left elbow
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_LEFT_ELBOW);
//draw limb from left elbow to left hand
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,
SimpleOpenNI.SKEL_LEFT_HAND);
//draw limb from neck to right shoulder
kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_RIGHT_SHOULDER);
//draw limb from right shoulder to right elbow
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_RIGHT_ELBOW);
//draw limb from right elbow to right hand
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
SimpleOpenNI.SKEL_RIGHT_HAND);
//draw limb from left shoulder to torso
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
//draw limb from right shoulder to torso
```

```
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_TORSO); //draw limb from torso to left hip
kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_LEFT_HIP);
//draw limb from left hip to left knee
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,
SimpleOpenNI.SKEL_LEFT_KNEE);
//draw limb from left knee to left foot
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,
SimpleOpenNI.SKEL_LEFT_FOOT);
//draw limb from torse to right hip
kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_RIGHT_HIP);
//draw limb from right hip to right knee
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
SimpleOpenNI.SKEL_RIGHT_KNEE);
//draw limb from right kneee to right foot
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,
SimpleOpenNI.SKEL_RIGHT_FOOT);
} // void drawSkeleton(int userId)}
```

6.3.3.4 ANGLE CALCULATION

The Six servos are placed on the shoulders, elbows, wrist and hand of the robotic arm. If the current bending angles of the joints can be calculated, angles to the servos can be mapped and thus make the robotic arm move like the user's arm. This can be implemented with the cunning use of geometry. First, add some PVectors at the beginning of the sketch (outside of any function) to store the position of all user's joints.

```
// reduce our joint vectors to two dimensions
PVector rightHand2D = new PVector(rightHand.x, rightHand.y);
PVector rightHand2Dz = new PVector(rightHand.z, rightHand.y);
PVector rightElbow2D = new PVector(rightElbow.x, rightElbow.y);
PVector rightElbow2Dz = new PVector(rightElbow.z, rightElbow.y);
PVector rightShoulder2Dz = new PVector(rightShoulder.z, rightShoulder.y);
PVector rightShoulder2D = new PVector(rightShoulder.x, rightShoulder.y);
```

```
// calculate the axes against which we want to measure our angles
PVector torsoOrientationr = PVector.sub(rightShoulder2D, rightHip2D);
PVector upperArmOrientationr = PVector.sub(rightElbow2D,
rightShoulder2D);
PVector newarmr = PVector.sub(rightShoulder2Dz, rightHand2D);
PVector upperLegOrientationr = PVector.sub(rightKnee2D, rightHip2D);
PVector torsoOrientationrz = PVector.sub(rightShoulder2Dz, rightHip2Dz);
PVector torsoOrientationl = PVector.sub(leftShoulder2D, leftHip2D);
PVector upperArmOrientationl = PVector.sub(leftElbow2D, leftShoulder2D);
PVector newarml = PVector.sub(leftShoulder2Dz, leftHand2D);
PVector upperLegOrientationl = PVector.sub(leftKnee2D, leftHip2D);
PVector torsoOrientationlz = PVector.sub(leftShoulder2Dz, leftHip2Dz);
```

A function is required that calculates the angle described by the lines joining three points this can be called as the angle() function.

```
float angle(PVector one, PVector two, PVector axis) {
PVector limb = PVector.sub(two, one);
return degrees(PVector.angleBetween(limb, axis));
}
```

The function **updateAngles( )** performs the calculation of all these angles and stores them into the
previously defined angles[] array. The first step is storing each joint position in one PVector. The
method getJointPosition( ) helps with this process. It takes three parameters: the first one is the ID of the skeleton, the second one is the joint from which the coordinates from has to be extracted, and the third one is the PVector that is set to those coordinates.

```
void updateAngles() {
// Left Arm
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_HAND, lHand);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_ELBOW,
lElbow);
```

```
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_SHOULDER,
lShoulder);
// Left Leg
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_FOOT, lFoot);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_KNEE, lKnee);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_LEFT_HIP, lHip);
// Right Arm
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_HAND,
rHand);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_ELBOW,
rElbow);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
rShoulder);
// Right Leg
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_FOOT,
rFoot);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_KNEE,
rKnee);
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_HIP, rHip);

// calculate the angles of each of our arms
float shoulderAngler =angleOf(rightElbow2D, rightShoulder2D,
torsoOrientationr);
float elbowAngler =angleOf(rightHand2D, rightElbow2D,
upperArmOrientationr);
float shoulderAnglezr =angleOf(rightHand2Dz, rightShoulder2Dz,
torsoOrientationrz);
//float shoulderAnglezr =angleOf(rightElbow2Dz, rightShoulder2Dz,
torsoOrientationrz);
float legAngler =angleOf(rightFoot2D, rightKnee2D, upperLegOrientationr);
float shoulderAnglel =angleOf(leftElbow2D, leftShoulder2D,
torsoOrientationl);
float elbowAnglel =angleOf(leftHand2D, leftElbow2D, upperArmOrientationl);
float shoulderAnglezl =angleOf(leftHand2Dz, leftShoulder2Dz,
torsoOrientationlz);
```

CHAPTER 7
CONCLUSION AND FUTURE ENHANCEMENT

7.1 CONCLUSION

Human movements (human motion analysis) in recent years has become a natural interface for HCI and has been the focus of recent researchers in modeling, analyzing and recognition of gestures. Hand gesture recognition still remains a prominent research area in the field of HCI as it provides a more natural way of interaction between humans and machine. This helps in building a bridge between machines and humans than other interfaces like text and GUI (graphical user interfaces). The Robotic arm which is built based on Kinect's gesture technology will act as a base for various advanced robotics automation and control. There will be a possibility in creating humanoid robots that are self-learning based on their perception of Human gestures. The applied methodology will have vast applications for industrial backgrounds as well as other research areas as this is a new approach towards Human Machine Interaction.

7.2 FUTURE ENHANCEMENT

The proposed project has covered all the requirements. Further requirements and improvements can easily be done since the coding is mainly structured and modular in nature. Changing the existing modules or adding new modules can append improvements. Further enhancements can be made to the application such that the robotic arm's real world applications improve. Some of the major changes that can be introduced are:

- Providing mobility to the robotic arm
- Improving the built of the robot
- Wireless controlling of robotic arm

CHAPTER 8
APPENDICES

8.1 HARDWARE DESIGN OF KINECT

The key components of Kinect are indicated in the figure 1, they are:

1. Multi-array microphone: This is an array of four microphones that can isolate the voices of the user from the noise in the room. By comparing the delay in each microphone, the voice source can be located.

2. IR laser emitter: Actively emitting near infrared spectrum, which can be distorted by uneven surface and then randomly formed as reflected speckles. The speckles can be received by infrared camera (No. 3 in the figure).

3. IR camera: Capturing infrared signal which can be converted into depth map.

4. Motorized tilt: The motor can be programmed in order to achieve the best view angle.

5. USB cable: Transmitting video stream, depth stream and audio stream. The extra power source must be connected to get all functions of Kinect. (The power of Kinect is 12W while the power of normal USB cable is 2.5W)

6. RGB camera: Capturing colour video stream.

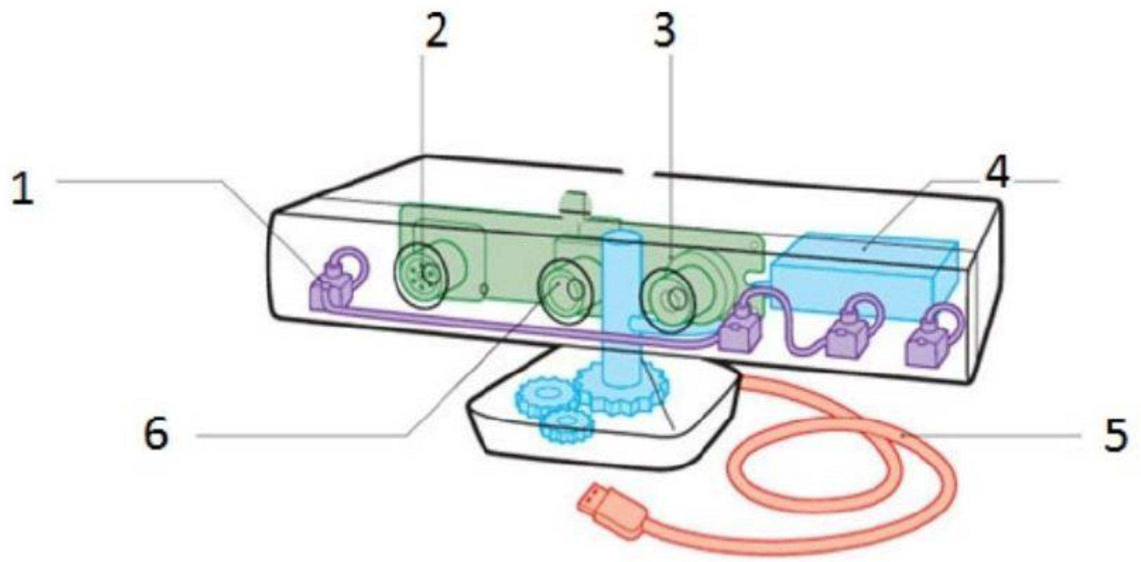
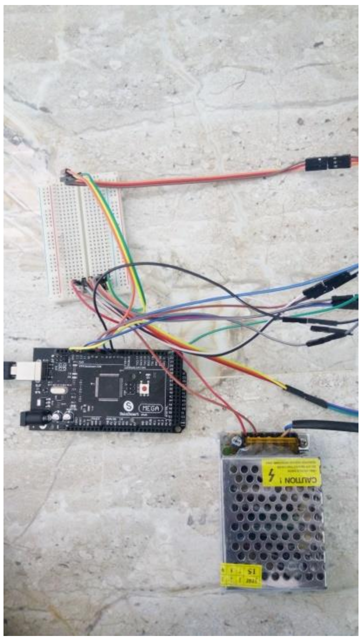


Figure 8.1: Structure of Kinect and its key components


(a)

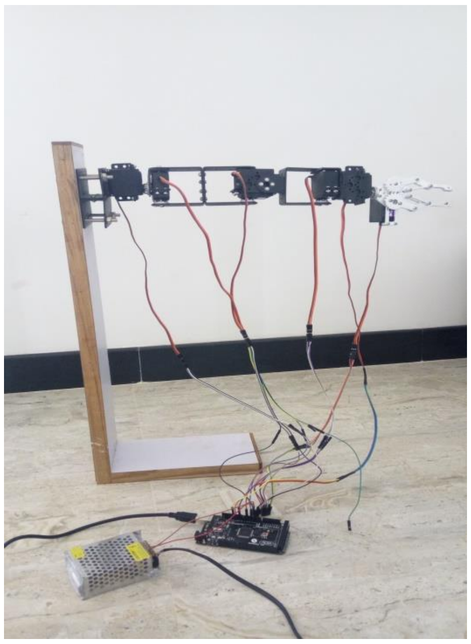

(b)

Figure 8.3: (a) Circuit connection (b) Working of robotic arm

The "heart" of Kinect is the PS1080 system on chip (SoC) produced by PrimeSense. It is a multi-sensor system which can provide depth image, colour image and audio signal at the same time. As shown in figure 2, PS1080 encode the IR light and project it to the scene while IR camera capture the IR light and send the signal back to PS1080. The PS1080 process the signal and retrieve the depth image and combine it with the corresponding colour image. Since the audio part is not important in this project so far, no detail of it will be introduced.

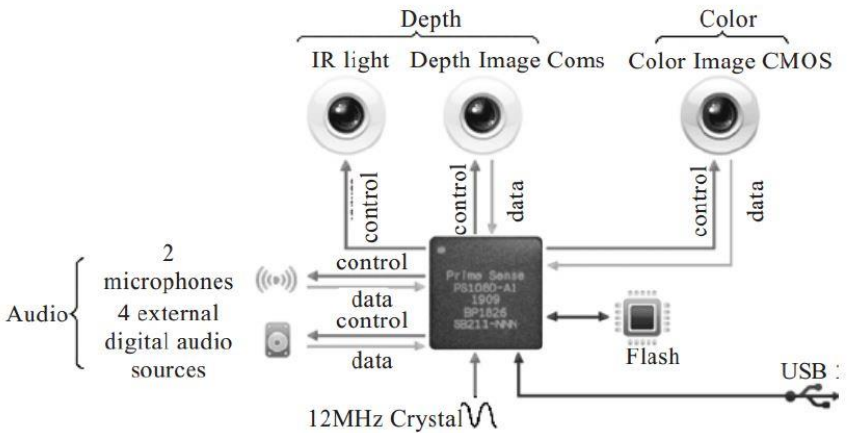


Figure 8.4: Recommended design of PrimeSense chip

The field of view in the system is 57 degrees horizontal, 43 degrees vertical, and the operational range is between 0.8 meters and 4 meters (normal mode). For the near mode, Kinect can detect object as close as 0.4 meters and as far as 3 meters.

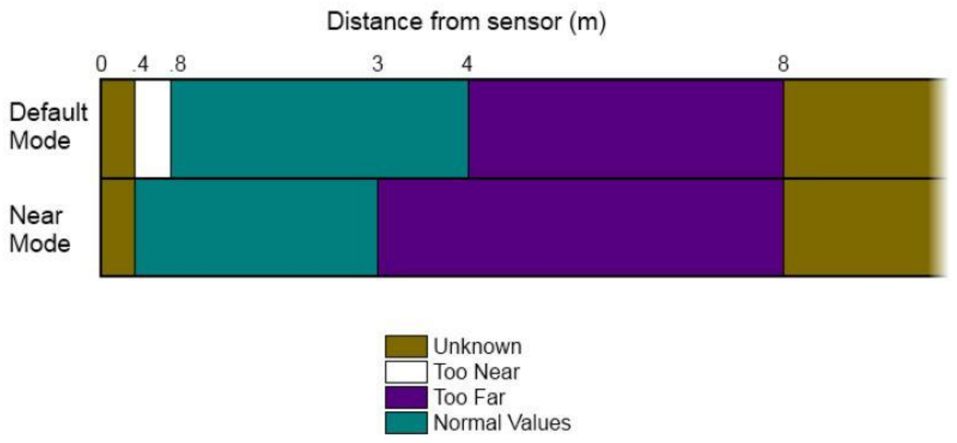
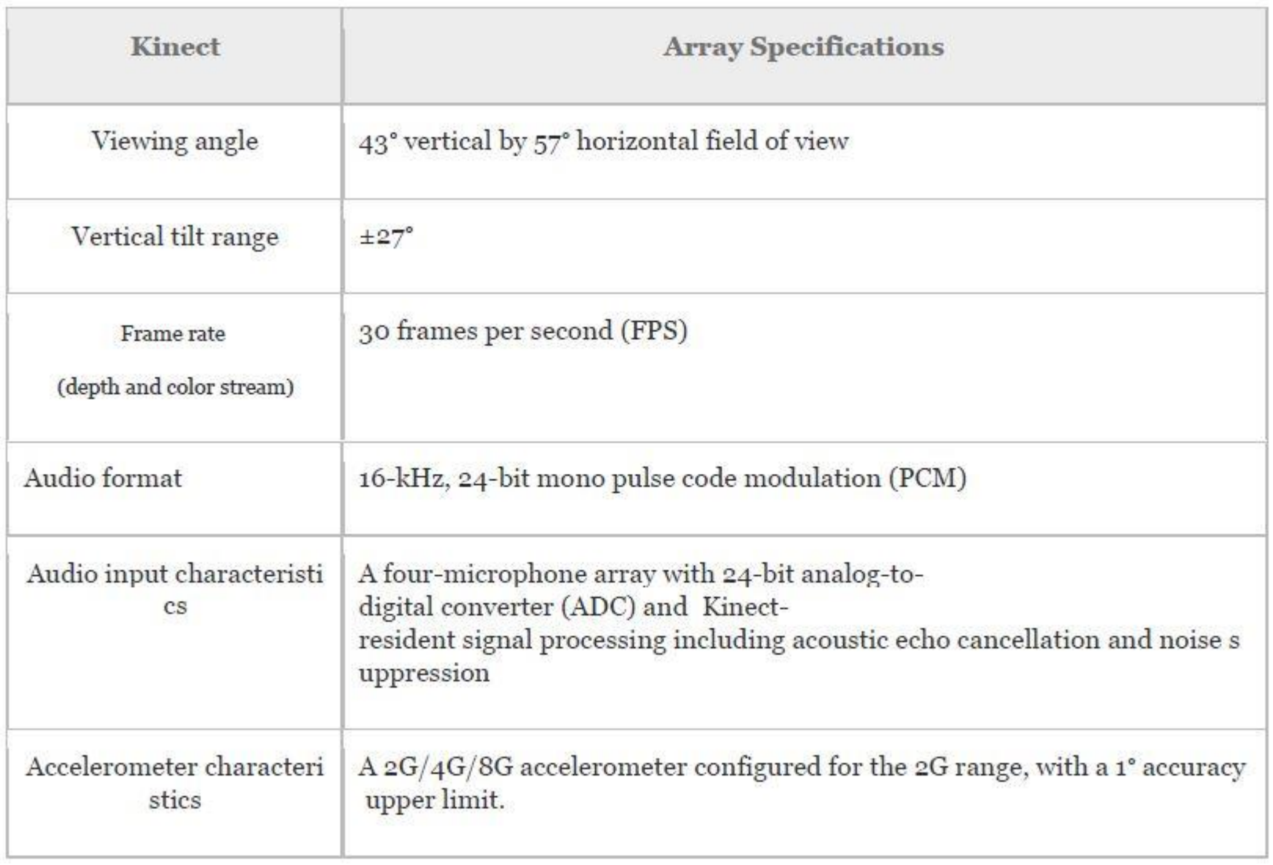


Figure 8.5: Types of values returned by the runtime

8.2 SPECIFICATION OF KINECT



Kinect	Array Specifications
Viewing angle	43° vertical by 57° horizontal field of view
Vertical tilt range	±27°
Frame rate (depth and color stream)	30 frames per second (FPS)
Audio format	16-kHz, 24-bit mono pulse code modulation (PCM)
Audio input characteristi cs	A four-microphone array with 24-bit analog-to-digital converter (ADC) and Kinect-resident signal processing including acoustic echo cancellation and noise s uppression
Accelerometer characteri stics	A 2G/4G/8G accelerometer configured for the 2G range, with a 1° accuracy upper limit.

TABLE 8.1: KINECT SPECIFICATION

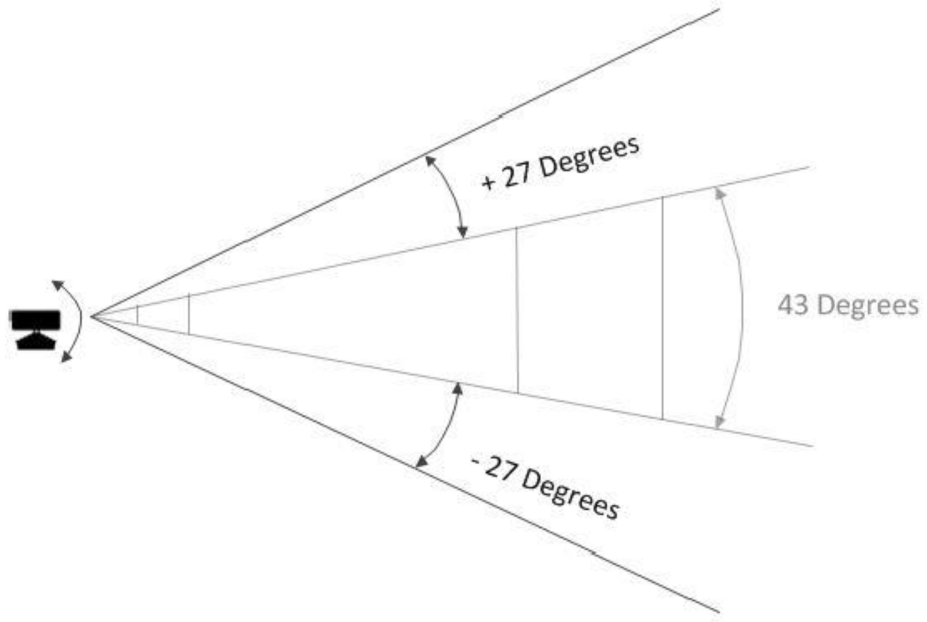
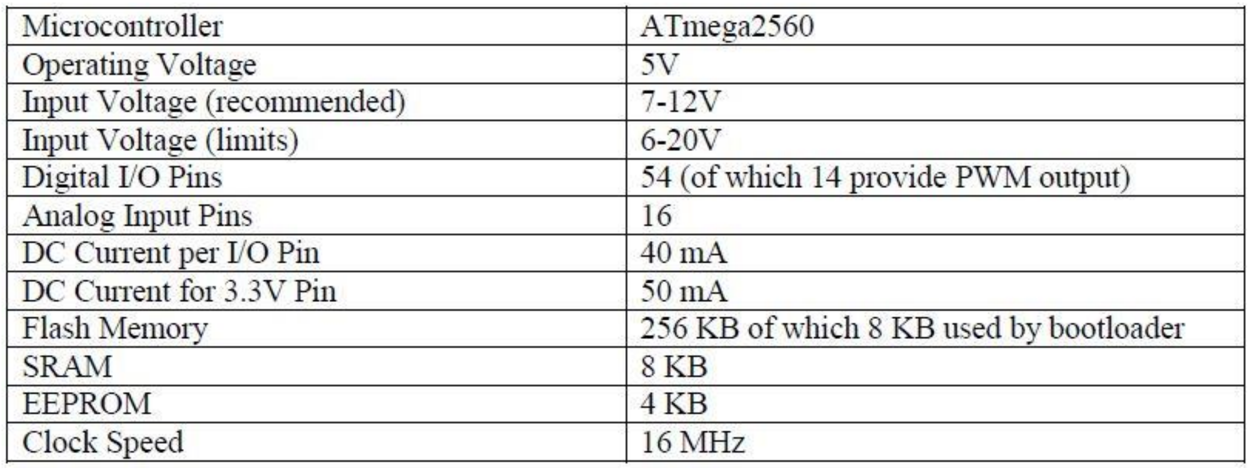


Figure 8.6: Kinect Degree of tilting

8.3 ARDUINO MEGA ADK



Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	54 (of which 14 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz

TABLE 8.2: ARDUINO MEGA ADK SPECIFICATION

8.4 POWER

The Arduino Mega can be powered via the USB connection or with an external power supply.

The power source is selected automatically.

External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-

positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector.
The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.
The Mega2560 differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter.

8.5 MEMORY

The ATmega2560 has 256 KB of flash memory for storing code (of which 8 KB is used for the bootloader), 8 KB of SRAM and 4 KB of EEPROM (which can be read and written with the EEPROM library).

8.6 INPUT AND OUTPUT

Each of the 54 digital pins on the Mega can be used as an input or output, using pinMode() , digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA and has an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins have specialized functions:

● Serial: 0 (RX) and 1 (TX); Serial 1: 19 (RX) and 18 (TX); Serial 2: 17 (RX) and 16 (TX); Serial 3: 15 (RX) and 14 (TX): Used to receive (RX) and transmit (TX) TTL serial data. Pins 0 and 1 are also connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip.

● External Interrupts: 2 (interrupt 0), 3 (interrupt 1), 18 (interrupt 5), 19 (interrupt 4), 20 (interrupt 3), and 21 (interrupt 2). These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

● **PWM: 0 to 13.** Provide 8-bit PWM output with the analogWrite() function.

● **SPI: 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS).** These pins support SPI communication using the SPI library. The SPI pins are also broken out on the ICSP header, which is physically compatible with the Uno, Duemilanove and Diecimila.

● **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED is on, when the pin is LOW, it's off.

● **I2C: 20 (SDA) and 21 (SCL).** Support I2C (TWI) communication using the Wire library (documentation on the Wiring website). Note that these pins are not in the same location as the I2C pins on the Duemilanove or Diecimila. The Mega2560 has 16 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). By default they measure from ground to 5 volts, though is it possible to change the upper end of their range using the AREF pin and analogReference() function.

There are a couple of other pins on the board:

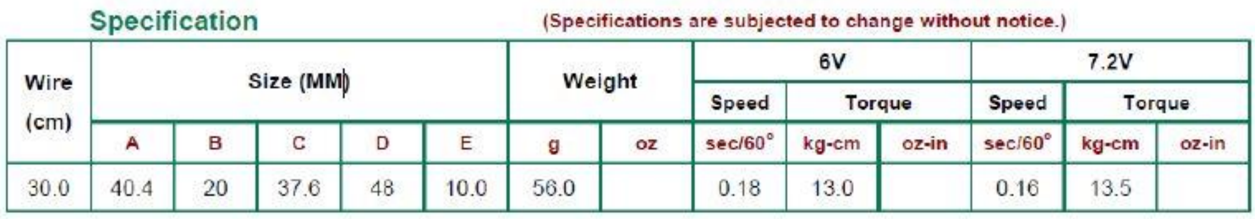
AREF: Reference voltage for the analog inputs. Used with analogReference().

Reset: Bring this line LOW to reset the microcontroller. Typically used to add a reset button to shields which block the one on the board.

8.7 PROGRAMMING

The Arduino Mega can be programmed with the Arduino software (download). For details, see the reference and tutorials. The ATmega2560 on the Arduino Mega comes preburned with a bootloader that allows to upload new code to it without the use of an external hardware programmer. It communicates using the original STK500 protocol (reference, C header files). It is also possible to bypass the bootloader and program the microcontroller through the ICSP (In-Circuit Serial Programming) header; see these instructions for details.

8.8 SERVO MOTOR DATA SHEET



Specification (Specifications are subjected to change without notice.)

Wire (cm)	Size (MM)					Weight		6V			7.2V		
								Speed	Torque		Speed	Torque	
	A	B	C	D	E	g	oz	sec/60°	kg-cm	oz-in	sec/60°	kg-cm	oz-in
30.0	40.4	20	37.6	48	10.0	56.0		0.18	13.0		0.16	13.5	

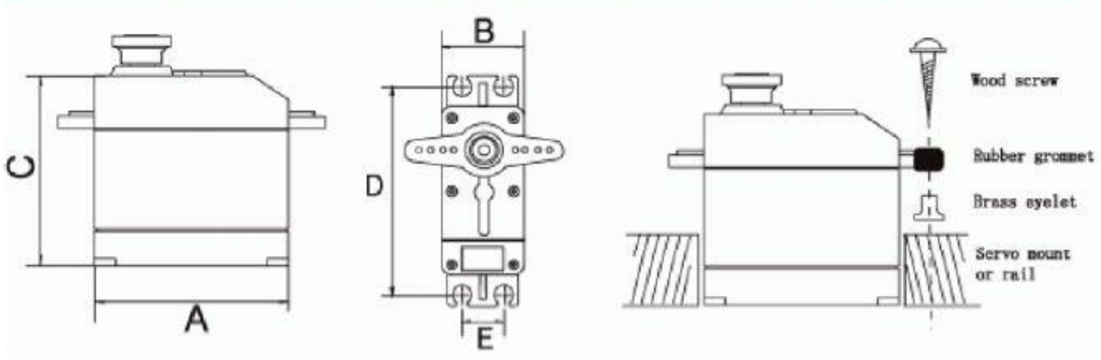


Figure 8.7: Servo motor data sheet

8.9 FINAL PROCESSING CODE:

```
import SimpleOpenNI.*;
SimpleOpenNI kinect;
// import the processing serial library
import processing.serial.*;
// and declare an object for our serial port
Serial port;
int[] userID;
// user colors
color[] userColor = new color[]{ color(255,0,0),
color(0,255,0),
color(0,0,255),
color(255,255,0),
color(255,0,255),
color(0,255,255)};
// postion of head to draw circle
PVector headPosition = new PVector();
// turn headPosition into scalar form
float distanceScalar;
```

```
// diameter of head drawn in pixels
float headSize = 200;
// threshold of level of confidence
float confidenceLevel = 0.5;
// the current confidence level that the kinect is tracking
float confidence;
// vector of tracked head for confidence checking
PVector confidenceVector = new PVector();
void setup() {
size(640*2, 480);
kinect = new SimpleOpenNI(this);
kinect.enableDepth();
kinect.enableRGB();
kinect.enableUser();
kinect.setMirror(false);
// Get the name of the first serial port
// where we assume the Arduino is connected.
// If it doesn't work, examine the output of
// the println, and replace 0 with the correct
// serial port index.
println(Serial.list());
//String portName = Serial.list()[1];
// initialize our serial object with this port
// and the baud rate of 9600
port = new Serial(this, "COM4", 9600);
port.bufferUntil('\n');
}
void draw() {
kinect.update();
PImage depthImage = kinect.depthImage();
PImage rgbImage = kinect.rgbImage();
image(depthImage, 0, 0);
image(rgbImage, 640, 0);
IntVector userList = new IntVector();
// get all user IDs of tracked users
userID = kinect.getUsers();
// loop through each user to see if tracking
```

```
for(int i=0;i<userID.length;i++)
{
if(kinect.isTrackingSkeleton(userID[i]))
{
// get the positions of the joints
drawSkeleton(userID[i]);
// get confidence level that Kinect is tracking head
confidence =
kinect.getJointPositionSkeleton(userID[i],SimpleOpenNI.SKEL_HEAD,confide
nceVector);
// if confidence of tracking is beyond threshold, then track user
if(confidence > confidenceLevel)
{
// change draw color based on hand id#
stroke(userColor[(i)]);
// fill the ellipse with the same color
fill(userColor[(i)]);
// draw the rest of the body
drawSkeleton(userID[i]);
//right arm
PVector rightHand = new PVector();
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_HAND,
rightHand);
PVector rightElbow = new PVector();
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_ELBOW,
rightElbow);
PVector rightShoulder = new PVector();
kinect.getJointPositionSkeleton(1,
SimpleOpenNI.SKEL_RIGHT_SHOULDER, rightShoulder);
//right leg
PVector rightKnee = new PVector();
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_KNEE,
rightKnee);
PVector rightHip = new PVector();
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_HIP,
rightHip);
PVector rightFoot = new PVector();
```

```
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_RIGHT_FOOT,
rightFoot);
//neck for depth
PVector torso = new PVector();
kinect.getJointPositionSkeleton(1, SimpleOpenNI.SKEL_TORSO, torso);
// convert our arm joints into screen space coordinates
//right arm
PVector convertedRightHand = new PVector();
kinect.convertRealWorldToProjective(rightHand, convertedRightHand);
PVector convertedRightElbow = new PVector();
kinect.convertRealWorldToProjective(rightElbow, convertedRightElbow);
PVector convertedRightShoulder = new PVector();
kinect.convertRealWorldToProjective(rightShoulder, convertedRightShoulder);
//right leg
PVector convertedRightHip = new PVector();
kinect.convertRealWorldToProjective( rightHip, convertedRightHip);
PVector convertedRightFoot = new PVector();
kinect.convertRealWorldToProjective( rightFoot, convertedRightFoot);
PVector convertedRightKnee = new PVector();
kinect.convertRealWorldToProjective( rightKnee, convertedRightKnee);
// reduce our joint vectors to two dimensions
PVector rightHand2D = new PVector(rightHand.x, rightHand.y);
PVector rightHand2Dz = new PVector(rightHand.z, rightHand.y);
PVector rightElbow2D = new PVector(rightElbow.x, rightElbow.y);
PVector rightElbow2Dz = new PVector(rightElbow.z, rightElbow.y);
PVector rightShoulder2Dz = new PVector(rightShoulder.z, rightShoulder.y);
PVector rightShoulder2D = new PVector(rightShoulder.x, rightShoulder.y);
PVector rightHip2D = new PVector(rightHip.x, rightHip.y);
PVector rightHip2Dz = new PVector(rightHip.z, rightHip.y);
PVector rightFoot2D = new PVector(rightFoot.z, rightFoot.y);
PVector rightKnee2D = new PVector(rightKnee.z, rightKnee.y);
// calculate the axes against which we want to measure our angles
PVector torsoOrientationr = PVector.sub(rightShoulder2D, rightHip2D);
PVector upperArmOrientationr = PVector.sub(rightElbow2D,
rightShoulder2D);
PVector newarmr = PVector.sub(rightShoulder2Dz, rightHand2D);
PVector upperLegOrientationr = PVector.sub(rightKnee2D, rightHip2D);
PVector torsoOrientationrz = PVector.sub(rightShoulder2Dz, rightHip2Dz);
// calculate the angles of each of our arms
float shoulderAngler =angleOf(rightElbow2D, rightShoulder2D,
torsoOrientationr);
float elbowAngler =angleOf(rightHand2D, rightElbow2D,
upperArmOrientationr);
float shoulderAnglezr =angleOf(rightHand2Dz, rightShoulder2Dz,
torsoOrientationrz);
//float shoulderAnglezr =angleOf(rightElbow2Dz, rightShoulder2Dz,
torsoOrientationrz);
float legAngler =angleOf(rightFoot2D, rightKnee2D, upperLegOrientationr);
//float shoulderAnglezl =angleOf(leftElbow2Dz, leftShoulder2Dz,
torsoOrientationlz);
float[] torsof = torso.array();
// gribber control
float rightf= abs(rightFoot.z/500);
int rf =int(rightf);
// right gribber
int gribr=90 ;
if(rf<=3 ) { gribr=180; }
else {gribr=45;}
// show the angles on the screen for debugging
fill(0,0,255);
scale(2);
text("shoulder_r: " + int(shoulderAngler) + "\n" +
" elbow_r: " + int(elbowAngler) + "\n" +
" Arm_r: " + int(shoulderAnglezr) + "\n" +
"torsof_x" + byte(torsof[0]/100) + "\n" +
"torsof_z" + byte(torsof[2]/500) + "\n" +
"foort_z" + int(abs(rightFoot.z/500)) + "\n" +
"gribr" + int(gribr), 20, 20);
byte shoulderAngler_b = byte(shoulderAngler);
byte elbowAngler_b = byte(elbowAngler);
byte shoulderAnglezr_b = byte(shoulderAnglezr);
byte gribr_b = byte(gribr);
port.write(shoulderAngler_b);
port.write(elbowAngler_b);
port.write(shoulderAnglezr_b);
port.write(gribr_b);
} }
} }
float angleOf(PVector one, PVector two, PVector axis) { PVector limb =
PVector.sub(two, one);
return degrees(PVector.angleBetween(limb, axis));
}
/*------------------------------------------------------------
Draw the skeleton of a tracked user. Input is userID
-------------------------------------------------------------*/
void drawSkeleton(int userId){
// get 3D position of head
kinect.getJointPositionSkeleton(userId,
SimpleOpenNI.SKEL_HEAD,headPosition);
// convert real world point to projective space
kinect.convertRealWorldToProjective(headPosition,headPosition);
// create a distance scalar related to the depth in z dimension
distanceScalar = (525/headPosition.z);
// draw the circle at the position of the head with the head size scaled by the
distance scalar
ellipse(headPosition.x,headPosition.y,
distanceScalar*headSize,distanceScalar*headSize);
//draw limb from head to neck
kinect.drawLimb(userId, SimpleOpenNI.SKEL_HEAD,
SimpleOpenNI.SKEL_NECK);
//draw limb from neck to left shoulder
kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_LEFT_SHOULDER);
//draw limb from left shoulde to left elbow
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_LEFT_ELBOW);
//draw limb from left elbow to left hand
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_ELBOW,
SimpleOpenNI.SKEL_LEFT_HAND);
//draw limb from neck to right shoulder
```

```
kinect.drawLimb(userId, SimpleOpenNI.SKEL_NECK,
SimpleOpenNI.SKEL_RIGHT_SHOULDER);
//draw limb from right shoulder to right elbow
//draw limb from right elbow to right hand
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_ELBOW,
SimpleOpenNI.SKEL_RIGHT_HAND);
//draw limb from left shoulder to torso
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
//draw limb from right shoulder to torso
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_SHOULDER,
SimpleOpenNI.SKEL_TORSO);
//draw limb from torso to left hip
kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_LEFT_HIP);
//draw limb from left hip to left knee
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_HIP,
SimpleOpenNI.SKEL_LEFT_KNEE);
//draw limb from left knee to left foot
kinect.drawLimb(userId, SimpleOpenNI.SKEL_LEFT_KNEE,
SimpleOpenNI.SKEL_LEFT_FOOT);
//draw limb from torse to right hip
kinect.drawLimb(userId, SimpleOpenNI.SKEL_TORSO,
SimpleOpenNI.SKEL_RIGHT_HIP);
//draw limb from right hip to right knee
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_HIP,
SimpleOpenNI.SKEL_RIGHT_KNEE);
//draw limb from right kneee to right foot
kinect.drawLimb(userId, SimpleOpenNI.SKEL_RIGHT_KNEE,
SimpleOpenNI.SKEL_RIGHT_FOOT);
} // void drawSkeleton(int userId)
/*-----------------------------------------------------------
When a new user is found, print new user detected along with
userID and start pose detection. Input is userID
-----------------------------------------------------------*/
void onNewUser(SimpleOpenNI curContext, int userId){
println("New User Detected - userId: " + userId);
```

```
// start tracking of user id
curContext.startTrackingSkeleton(userId);
} //void onNewUser(SimpleOpenNI curContext, int userId)
/*-------------------------------------------------------------
Print when user is lost. Input is int userId of user lost
-------------------------------------------------------------*/
void onLostUser(SimpleOpenNI curContext, int userId){
// print user lost and user id
println("User Lost - userId: " + userId);
} //void onLostUser(SimpleOpenNI curContext, int userId)
/*-------------------------------------------------------------
Called when a user is tracked.
-------------------------------------------------------------*/
void onVisibleUser(SimpleOpenNI curContext, int userId)
{
// print user lost and user id
println("visible - userId: " + userId);
} //void onVisibleUser(SimpleOpenNI curContext, int userId)
```

8.10 FINAL ARDUINO CODING

```
#include <Servo.h>
Servo shoulderR;
Servo shoulderFR;
Servo elbowR;
Servo gribR;
Servo stoppos;
void setup()
{
// attach servos to their pins
shoulderFR.attach(6);
shoulderR.attach(7);
elbowR.attach(8);
gribR.attach(9);
stoppos.attach(2);
stoppos.write(90);
shoulderR.write(90);
shoulderFR.write(180);
elbowR.write(90);
gribR.write(80);
Serial.begin(9600);
}
void loop()
{
while(!(Serial.available()>=4));
byte shoulderAngler_br = Serial.read();
byte elbowAngler_br = Serial.read() ;
byte shoulderAnglezr_br = Serial.read() ;
byte gribr_br =Serial.read();
shoulderR.write(abs(180-shoulderAngler_br));
shoulderFR.write(abs(180-shoulderAnglezr_br));
elbowR.write(abs(elbowAngler_br));
gribR.write(abs(gribr_br));
}
```

CHAPTER 9
REFERENCES

[1] Andrea Sanna, “A Kinect based natural interface for quadrotor control”, International Conference on Intelligent Technologies for Interactive Entertainment, 2011, Pages 48-56

[2] Badar, “Human detecting and following mobile robot using laser range sensor”, Mechatronics and Automation (ICMA), IEEE International Conference, ISBN-9781467355605, 2013

[3] Daniel James Ryan, “Finger and gesture recognition with Microsoft Kinect”, University of Stavanger, IEEE Conference on Computer Vision and Pattern Recognition, Pages 274-280, 1999

[4] Kun Qian, “Developing a gesture based remote Human-robot interaction system using Kinect”, International Journal of Smart Home Vol. 7, No. 4, July, 2013

[5] Mohammed Eltaher, “Motion detection using Kinect device for controlling robotic arm”, 2013 ASEE Northeast Section Conference, Norwich University, March 14-16, 2013

[6] Noriyuki Kawarazaki, “Development of human following mobile robot system using laser range sensor”, 2015, Procedia Computer Science-ScienceDirect, Vol 76, 2015, Pages 455-460

[7] Priya Matnani, “Glove based and accelerometer based gesture control”, International Journal of Technical Research and Applications e-ISSN: 2320-8163, www.ijtra.com Volume 3, Issue 6 (November-December, 2015), Pages. 216-221

[8] Suzuki, “Human tracking mobile robot with face detection”, Toshiba Review, vol. 60, no. 7, Pages 112-115, 2005.

[9] Tim Braun, “Detecting and following Humans with a mobile robot”, IEEE Conference on Computer Vision and Pattern Recognition, 1, Pages 274-280, 1999

[10] http://sunnybrook.ca/media/item.asp?i=616

[11] http://www.informationweek.com/big-data/hardware-architectures/microsoft-kinect-inexpensive-big-data-tool/d/d-id/1252896

[12] About Kinect, www.hongkiat.com/blog/kinect

[13] Gesture based devices, www.gesturetek.com

[14] HCI, www.interaction-design.org/human_computer_interaction_hci.html

[15] About Kinect retrieved from www.stackoverflow.com/tags/kinect/info

[16] https://brage.bibsys.no/xmlui/handle/11250/181783

[17] https://kinectlibrary.codeplex.com/

[18] dev.windows.com/en-us/kinect/

[19] www.microsoft.com/en-in/download/details.aspx?id=40278

[20] www.stackoverflow.com/tags/kinect/info

[21] About Kinect for Xbox 360 https://www.engadget.com/2010/11/04/kinect-for-xbox-360-review/

[22] Usage of Kinect for various DIY projects: http://www.theverge.com/2011/12/6/2616242/kinect-hacks

[23] About Kinect for Xbox ONE http://www.xbox.com/en-US/xbox-one/accessories/kinect

[24] Skeletal tracking: http://www.contentmaster.com/kinect/kinect-sdk-skeleton-tracking/

[25] About Industrial applications http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S1692-17982015000100003

[26] 3D sensor in robotics www.intorobotics.com/working-with-kinect-3d-sensor-in-robotics-setup-tutorials-applications/

[27] About NAO robot
www.researchgate.net/publication/283470552_Inverse_kinematics_of_a_NAO_humanoid_robot_using_kinect_to_track_and_imitate_human_motion
[28] https://forum.processing.org/one/topic/hand-tracking-with-simpleopenni-and-kinect.html
[29] https://www.intorobotics.com/7-tutorials-start-working-kinect-arduino/
[30] Kinect drivers, http://forum.ubi-interactive.com/hc/en-us/articles/202247526-Download-Kinect-Drivers
[31] https://forum.processing.org/one/topic/servo-control-with-processing-kinect-and-arduino.html
[33] https://channel9.msdn.com/Forums/TechOff/Controlling-servo-with-kinect-and-arduino
[34] http://nuigroup.com/forums/viewthread/11154/
[35] http://kelvinelectronicprojects.blogspot.in/2013/08/kinect-driven-arduino-powered-hand.html
[36] Simple Open-NI downloads, http://openni.ru/files/simpleopenni/index.html
[37] Kinect SDK download, https://www.microsoft.com/en-us/download/confirmation.aspx?id=40276