MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

MOTION CONTROL OF ROBOT BY USING 3D CAMERA

YUSUF SARIKAYA

MASTER THESIS

Department of Mechatronics Engineering

Thesis Supervisor

Assoc. Prof. Dr. Erkan KAPLANOGLU

ISTANBUL, 2017

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

MOTION CONTROL OF ROBOT BY USING 3D CAMERA

YUSUF SARIKAYA

(523515002)

MASTER THESIS

Department of Mechatronics Engineering

Thesis Supervisor

Assoc. Prof. Dr. Erkan KAPLANOGLU

ISTANBUL, 2017

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Yusuf SARIKAYA, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended his thesis entitled "**Motion control of robot by using 3D camera**", on October 27, 2017 and has been found to be satisfactory by the jury members.

Jury Members

Assoc. Prof. Dr. Erkan Kaplanoğlu (Advisor)
Marmara University .... (SIGN) [illegible]

Assist. Prof. Dr. Hüseyin Yüce (Jury Member)
Marmara University .... (SIGN) [illegible]

Assist.Prof. Gökhan Erdemir (Jury Member)
Sebahattin Zaim University .... (SIGN) [illegible]

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Yusuf SARIKAYA be granted the degree of Master of Science in Department of Mechatronics Engineering, Mechatronics Engineering Program on 06/11/2017.... (Resolution no:2017/24).02

Director of the Institute

Prof. Dr. Adı SOYADI

[illegible]

Prof. Dr. Bülent EKİCİ
Enstitü Müdürü

ACKNOWLEDGMENT

I would like to thank Prof. Sansoni for the opportunities presented to me at optoelectronics laboratory in the Universita degli Studi di Brescia to develop this project.

I also want to thank to my tutor Assoc. Prof. Erkan Kaplanoğlu for his help throughout the whole project.

Especially I give my thanks to my family and my friends because they support me in every moment of my project and my life.

October/2017 **Yusuf Sarıkaya**

TABLE OF CONTENTS

ÖZET

“3D CAMERA İLE ROBOT HAREKETLERİNİN UZAKTAN KONTROLÜ”

Endüstride birçok zaman zor koşullarda çalışması için insan gücü yerine robot gücü kullanılmaktadır. Özellikle kaynak yapmak, yük taşımak ya da montaj yapmak için robot sistemleri kullanılmaktadır. Robotu kullanmak içinse uzaktan kontrol sistemleri geliştirilmiştir. Kinect Kamera gibi 3 boyutlu kameralar uzaktan kontrol sistemlerinde iyi bir alternatif sunmaktadırlar. Çünkü kamera ile yapılan robot kontrolünde kullanıcının robotu kontrol etmesi için kumanda gibi cihazlar kullanmasına gerek yoktur.

Bu çalışmada da Kinect kamera kullanılarak robot kol için bir uzaktan kontrol sistemi tasarlanmıştır. İlk önce Kinect kamera kullanıcının iskelet izlemesini yaparak, tüm eklemler için koordinatları tespit etmektedir. Bu iskelet izlemesi Microsoft’un resmi yazılım kütüphanesi kullanılarak elde edilmiştir. Bu tespit edilen noktalar eklemler arasındaki açıların hesaplanması için kullanılmıştır. Daha sonrada bu hesaplanan açı değerleri 4 eksenli robotik kola gönderilmiştir. Böylece uzaktan kontrol sistemi sağlanmıştır. Bu uzaktan kontrol sistemi temelde üç aşamadan oluşmaktadır. İlk aşamada kullanıcının eklem bilgileri geliştirilen bilgisayar programıyla alınır. İkinci aşamada bu alınan veriler nokta çarpımı ve ters kinematik metotları ile ayrı ayrı işlenir. Üçüncü aşamada ise metotlar sonucu hesaplanan açı değerleri robot kola gönderilir.

Bu aşamalar sırasında bir adet Kinect kamera, bir adet bilgisayar, bir adet Arduino Uno mikro kontroller ve son olarak da 3 boyutlu yazıcı yardımıyla yapılan bir adet robot kol kullanılmıştır. Kinect kamera daha öncede söylediğimiz gibi kullanıcının eklem koordinatlarını bulmak için kullanılmıştır. Bilgisayar ise ara yüz programını oluşturmak için kullanılmıştır. Bu program C# programlama dilinde Visual Studio 2015’de geliştirilmiştir. Ayrıca açı hesaplamaları da bu programın arka planında gerçekleşmektedir. Arduino Uno hesaplanan açıları robot kola göndermek için kullanılmaktadır. Bu yüzden bilgisayar ve robot kol arasında bir köprü görevi görmektedir. Robot kol (EEZYbotARM MK2, EEZYRobotsTM), tüm parçaları açık kaynak kodlu olan ve 3 boyutlu yazıcıda bastırılıp birleştirilmiştir.

Geliştirilen projenin amacı Kinect kamera yardımıyla robot kol için uzaktan kontrol sistemi geliştirmektir. Geliştirilen sistem uygun fiyata mal olmuştur. Ayrıca endüstri veya eğitim için kolayca değiştirilerek kullanılabilmektedir.

Anahtar Kelimeler: Hareket Algılama, İnsan-robot etkileşimi, Kinect, Robot Kol

Ekim/2017 **Yusuf Sarıkaya**

ABSTRACT

"MOTION CONTROL OF ROBOT BY USING 3D CAMERA"

In industrial applications, the operatives have to use robots which are in a risky environment. Tele-control represents a good solution to control the robot from distant places (for example during handling, welding or assembling operations) maintaining the right level of safety. An innovative solution can be the use of Time-Of-Flight (TOF) cameras (for example Kinect camera, Microsoft ^{TM}) to remote control the robotic system without using other types of wearable or holding controllers.

In the current work, the movement of the human body is captured with Kinect Camera by measuring 3D point clouds. The human body skeleton is then tracked by using skeleton tracking developed thanks to the Microsoft Kinect Software Development Kit (SDK). Furthermore, the measured joints are used in order to calculate the angles between the human limbs. All these data are finally used for the remote control of a four axis robotic arm. All parts of the robotic arm were made and reconstructed using a 3D printer.

A remote control system is created for a 3D printed robotic arm by using Kinect Sensor. The system is composed of three parts. The first part aims to find the upper limb's joints by using the point clouds acquired by the Kinect Sensor; the second part calculates the dot product method and the kinematics equations of the robot arm; the last part sends the processed data to the robot to control the movement.

The system is composed of the Kinect Camera, a computer for creating a program, a microcontroller and a Robotic arm. The Kinect Camera, which is a kind of TOF camera is able to measure the 3D coordinate of the scene. It can be used to track the human body skeleton and to recognize the human gesture.

The computer is used as a bridge to receive data from Kinect sensor and to send the processed data to the microcontroller. The visual interface program is developed in Visual Studio 2015 with C# language to calculate the angles between limbs.

Arduino Uno, which is a kind of microcontroller, is used in the present project. The purpose of the microcontroller is to drive the servo motors of the robotic arm.

A four axis open source 3D robotic arm is used in this work (EEZYbotARM MK2, EEZYRobots^{TM}). All parts of the robotic arm are printed by a 3D printer and reconstructed completely.

In the following, the logic of the system will be explained more in detail. The data processing starts to capture the user joint coordinates by using Kinect camera. The Data is sent to the computer in order to calculate the upper limb joints angles by using the developed software. In the background of the program two methods, which are the dot product method and the inverse kinematics method, are used. Afterwards, the calculated angles are sent to the Arduino Uno microcontroller via an USB-cable. The motors are controlled by using the Pulse-Width-Modulation (PWM) technique.

The aim of this project is to create a remote control system for a robotic arm based on the Kinect sensor. The human movements were recorded and transferred to the robotic device. A well-priced control system was created which can be used for industrial purposes as well as for educational applications.

Keywords: Motion Capture, Human-Robot Interaction, Kinect, Robotic Arm

October/2017 **Yusuf Sarıkaya**

SYMBOLS

$\mathbf{T_i^{i-1}}$	**:** The matrix between two adjacent joints
$\boldsymbol{\alpha}$	**:** Angle between two vectors
$\bar{\mathbf{x}}$	**:** Mean Value
$\boldsymbol{\sigma}$	**:** Standard Deviation
A	**:** The length of the Robotic arm’s joints
$\boldsymbol{\theta}$	**:** Angle

ABBREVIATIONS

3D	: Three dimensional
OPENNI	: Open Natural Interaction
DOP	: Degree of freedom
UDMI	: Ultrasonic distance measurement instrument
2D	: Two dimensional
IR	: Infrared
CMOS	: Metal oxide semiconductor
RGB	: Red-Green-Blue
FPS	: Frames per second
BST	: Breakaway support technology
PWM	: Pulse with a modulation
IDE	: Integrated development environment
SDK	: Software Development Kit
WPF	: Windows presentation foundation
XAML	: Extensible application markup language

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

In the recent times the research about human robot interaction, which hold the attention of many researchers, is a contemporary issue because of the difficulties to build a system for interaction with robot.

There are some techniques to develop a control system for human robot interaction. For example image processing methods and voice recognition methods are ways to interact with a robot.

Industrials robots are used instead of the people in many applications such as handling, assembling and packaging of objects. Because Robots are able to perform these tasks more quickly than humans and they can easily transport and move objects that are often heavy or in a dangerous area.

Beside the area of industrial robots it is a significance in the bio-medical area as well. The topic of robotics engineering is getting more important with the target to support handicapped people in the daily life. In general, the topics of robotics is always connected with the idea to make the life easier for the humans in many different contexts.

In this way the control system of the robot should be simple in order to control the robot easier. Most of time the hand controllers such as joysticks are used for this and usually the user needs knowledge about the robotic system in this kind of controller. On the other hand the remote control systems by using a 3D camera is a simple application for users without knowledge about robotic systems because the user doesn't need to use an additional controller. The only necessary thing is that the user is situated in front of the camera. For this reason the remote control system of a 3D camera is developed in this project.

In present project and the developed system are planned to be well-priced, user friendly and the flexible in order to implement it to the other projects easily. Thus the well-priced microcontroller is chosen and the robotic arm is printed in this project. As well the methods, which are the dot product and the inverse kinematics, can be changed easily according to the project's desires.

To realize the following work the thesis is divided into three main parts. Firstly the theoretical foundations and the current state of research will be presented, in order to have a first overview about the topic. Therefore, the fundamental principles of forward kinematics and inverse kinematics are explained more in detail are part of the theoretical assumptions.

Furthermore the control system within the scope of the project will be explained more in detail. Especially the 3D camera with its individual components, the robot and the microcontroller will be presented.

The second major part of this thesis includes the implementation of the project. The main focus will be the implementation and the interaction of the individual parts such as the robot arm and the software. In the last part of the thesis the experimental results are presented and evaluated.

1.1. Literature review

Many projects in the past have used the imitation of gesture to control robots. In the paper “Motion Control of Robot by using Kinect Sensor” published in the Research Journal of Applied Sciences, Engineering and Technology, the researchers studied about the motion of the remotely controlled robotic arm by using the Kinect camera to take information about the human body joints data and by using a Arduino microcontroller to control the servos motor of the robot [1]. This project is using the software Open Natural Interaction (OPENNI) which is an open source software development kit (SDK) including special libraries for developing 3D image processing applications.

Another paper about the “Gestural Interaction for Robot Motion Control” is dealing with the topic of the implementation of the remote control robot system [2]. A mobile robot, which has eight degree of freedom (DOP) for the forearms and two wheels for mobility, are controlled by Kinect tracking skeleton data.

In the other research, “Implementation of Gesture Control in Robotic Arm using Kinect Module”, the Microsoft Kinect SDK includes an official Microsoft library for Kinect sensor, is used to build the frame of the skeleton image [3]. In the research published in quotation “Teleoperation Humanoid Robot Control System Based on Kinect Sensor”, a teleportation humanoid robot control system is designed, adapted from Kinect sensor [4]. The author of this paper developed the ultrasonic distance measurement instrument (UDMI) which is an effective and non-contact distance measuring instrument. Two

Kinect and the UDMI are used firstly to take the joints of the skeleton data from the user to receive secondly a feedback from the humanoid robot in order to reduce the missing operation.

The last two interesting papers, which are “Motion Control System for a Humanoid Robot based on Motion Capture using Kinect” and “Kinect-Based Humanoid Robotic Manipulator for Human Upper Limbs Movements Tracking”, are dealing with the real time motion control systems which are designed for using Kinect upper limbs body tracking. In these projects, after the calculation of the angles which are between the upper limbs, one of them is calculated with cosine interpolation for the manipulation of the robot [5]. However, to receive better results for the imitation of human gestures this calculation is not sufficient. The other one is calculated with the method of inverse kinematics model to manipulate the robot and there is problem when the users move their arm in front of any part of the body, the tracked joints of accuracy is not good in this research [6].

2. THEORETICAL PART

2.1.KINEMATICS

In this chapter, theoretical information on the topics of kinematics of robots and control of robots will be treated in detail.

The field of kinematics is a very important topic for robot systems. Components such as the robot end effector or the robot joint angles are determined by kinematics equations. Particularly in the industrial sector the robot manipulator systems must be very precise in order to complete the task of the system, such as assembly, handling and welding [7].

The field of kinematics is divided in two approaches, which are Forward Kinematics and Inverse Kinematics. These two approaches are used to determine different joint variables for the robot manipulator. Forward and inverse kinematics equations are based on the calculation of the joint coordinates and the joint angles.

In the present project, the kinematics equations are favored to calculate the joint angles of the robotic arm, because the kinematics allows the calculation to find the unknown variables of the robotic arm by using very few data and in the same time it provides a high precision in the movement of the robotic arm.

2.1.1. Forward kinematics

The principle of Forward kinematics is going to be explained generally at this point. On the contrary, the principle of inverse kinematics was used and implemented in the present case to the control system due to the unknown variables.

The end point coordinates of the robot such as gripper can be determined by using forward kinematic equations if the robot's joint angles and the robot's pieces length are known. While solving the kinematics equation there are also different solutions based on the robot configurations which are the Cartesian, cylindrical and spherical coordinates.

The robotic arm, which is used in this project, is able to move in cylindrical coordinates. Rotation in the base of the robotic arm and the movement in the joints of the robot can be given as an example.

The robotic arms are formed by various joints such as revolute, prismatic or slightly more complex joints as socket or spherical. The degree of freedom of the robotic arm is determined by considering all joints together. In the forward kinematics calculation all joints of the robot has to be considered piece by piece to provide the all movement of the robot in the calculation [8].

A coordinate system is placed for each joints in order to find the transformation matrixes that is the important to find the relations between the adjacent joints. Because the end effector position and the orientation depends on the relations between the joints which are mathematically given by the transformation matrix. The matrix between two adjacent joints can be shown by matrix T_i^{i-1} that describes the transformation from the coordinate system “i” to coordinate system “i-1”. For example the transformation matrix for the first, second and third joint are respectively T_1^0, T_2^1 and T_3^2. If all adjacent matrixes are multiplied with each other the final matrix will denote the forward kinematics which expresses the position and the orientation of the end frame in respect to the base frame which is shown in equation 2.1 [9].

$$\mathrm{T}_\mathrm{N}^0 = \mathrm{T}_1^0\mathrm{T}_2^1\mathrm{T}_3^2\ldots\mathrm{T}_\mathrm{N}^{\mathrm{N-1}} \quad (2.1)$$

The equation 2.1 is the transformation matrix for N joints of the robot. In addition the position and the orientation of the end effector can be described by three vectors o_N^0 which identify the origin of the end effector frame with respect to the base frame and the rotation matrix R_N^0. Thus the matrix, which includes the rotation and the origin coordinate of the end effector, define the homogenous transformation matrix.

$$\mathrm{T}_\mathrm{N}^0 = \begin{bmatrix} \mathrm{R}_\mathrm{N}^0 & \mathrm{o}_\mathrm{N}^0 \\ 0 & 1 \end{bmatrix} \quad (2.2)$$

In equation 2.2, the forward kinematics is defined by using a homogenous transformation matrix.

$$\mathrm{R}_\mathrm{N}^0 = \mathrm{R}_1^0\mathrm{R}_2^1\ldots\mathrm{R}_\mathrm{N}^{\mathrm{N-1}} \quad (2.3)$$

$$\mathrm{o}_\mathrm{N}^0 = \mathrm{o}_{\mathrm{N-1}}^0 + \mathrm{R}_{\mathrm{N-1}}^0\mathrm{o}_\mathrm{N}^{\mathrm{N-1}} \quad (2.4)$$

The equations 2.3 and 2.4 are the formulas to calculate the rotation matrix R_N^0 and the coordinate vectors o_N^0.

The forward kinematics can be calculated by using the above principles. First the homogenous matrix is calculated for each frame and then the calculated matrix is multiplied with each other.

2.1.2. Inverse kinematics

In forward kinematics the end effector position and the orientation can be calculated according to the joint variables which are known before the calculation. On the other hand the inverse kinematics finds the joint variables according to the end effector coordinate which is known before the calculation. Most of the time the approach of inverse kinematics is more difficult than the approach of forward kinematics, due to the fact that there can be more than one or no solution available by calculating with the principle of inverse kinematics.

The solution of the homogeneous transformation matrix, which is given in equation 2.5, shows the joints variables for the desired coordinates for the end effector. The joint variable “q” represents an angle if the joint type is revolute. if the joint type is prismatic, it represents the length of the robot parts [8].

$$T_N^0 = T_1^0(q_1)T_2^1(q_2)T_3^2(q_3)\ldots T_N^{N\text{-}1}(q_N) \qquad (2.5)$$

For example the robotic arm, which has N joints, the position and the orientation of the fourth joint frame depends on only first fourth joints variables.

The mathematical notation, which shows the relation between the fourth joint and the initial frame, is given by equation 2.6. The end effector coordinates according to the fourth joint is expressed by equation 2.7 [10].

$$T_{\text{fourth}} = T_1 T_2 T_3 T_4 \qquad (2.6)$$

$$T^k = T_4^{-1} T_3^{-1} T_2^{-1} T_1^{-1} T_{\text{fourth}} \qquad (2.7)$$

The equation 2.6 and 2.7 can be used in order to find the desired joint variables for the end effector. Thus the solution of the inverse kinematics starts with the equation 4.

$$T_1^{-1} T_{\text{fourth}} = T_2 T_3 T_4 \qquad (2.8)$$

By solving equation 2.8, the first joint angle can be calculated and then the other desired angles can be found with the use of a similar approach to provide the end effector's coordinates. Unfortunately this method for inverse kinematics sometimes needs a complex mathematical calculation. For this reason the other method, which is geometric approach, is preferred in the inverse kinematics problem because the solution of inverse kinematics needs less mathematical calculation.

3. 3D CAMERAS

In daily life the camera and its technology can be seen in lots of devices such as personal computer and mobile phones. The big usage area of the camera get the people and researcher excited. The user can easily take an image and record a video by using the 2D cameras. Especially in the research area the 3D camera, which can sense the depth of the environment beside of the normal RGB image, is a very popular device in order to use it. The 3D camera can be used for a lot of applications such as object recognition, 3D reconstruction of the environment, object tracking, human detection and gesture recognition. Moreover the applications of the 3D camera can be implemented to other applications like robotic.

There are different companies which develop 3D camera technologies and the Microsoft Kinect camera is the most preferred one in the 3D camera technology area because it is well-priced in comparison to other cameras and it is providing official libraries and documents with instructions while developing a software.

3.1. Kinect sensor

Until a few years ago the computers had a limited way to communicate with the environment; as well the users did not have many possibilities to give an input to the computer without touching any hardware. The most common ways are to use camera or a sound equipment system for taking data with a computer. This data could be saved or opened in the computer but the computer was not able to understand and to process these data to an active application. For example the distance can be estimated with sound

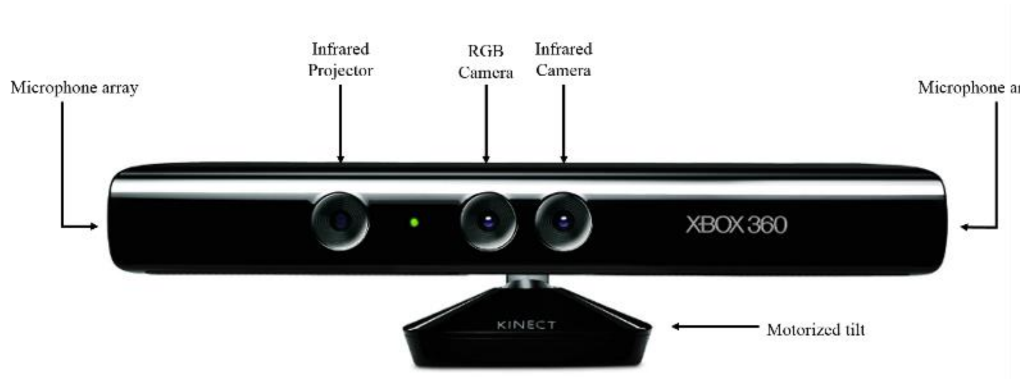


Figure 3.1: Microsoft Kinect Camera

information by using a microphone or the object recognition can be done by using camera. However it is quite difficult to analyze a picture exactly because this data is only a reflection from a 3D space to a 2D space [11].

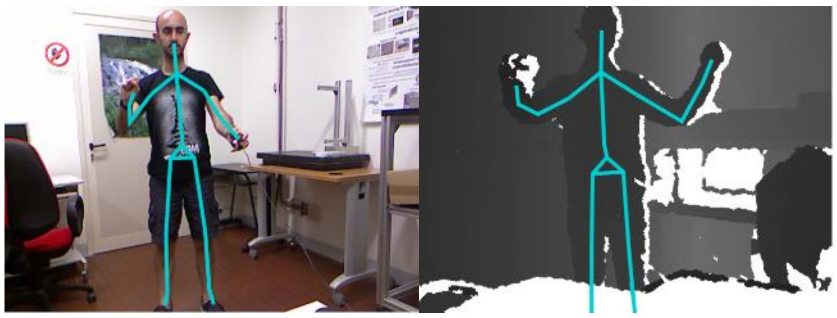
The solution of this issue has been provided by Microsoft Kinect Camera with the recognition of objects and humans in 3D space.

The Kinect Camera is formed with two cameras, one light source and four microphones in itself. It is shown in figure 3.1. The three-depth image is constructed by both, Infrared (IR) projector and IR Camera that consist of a Monochrome complementary metal oxide semiconductor (CMOS) sensor. This technology is taken the license by Prime Sense Company [12]. Beside of the 3D motion capture, the other possibility is the voice recognition with the microphones based on properties of waves like reflection and refraction.

The fundamental of the 3D depth sensing system is based on optics principles. The infrared light, which comes from the IR project of the Kinect device, is reflected from objects in the environment and then the reflected light is detected by IR camera. Thus, the distance between the camera and the object can be calculated by using the traveling duration of light in the air.

3.2. Color stream and Depth Stream

The Kinect camera provides some features to take images with the two main possibilities of color image taking and depth image taking. Many cameras as Kinect camera support the fundamental function which is RGB video format. The Kinect camera is able to



(a) (b)

Figure 3.2: (a) Color Image (b) Depth Image

capture the video in RGB format with 640x480 resolution at 30 frames per second (FPS) and also 1280x960 resolution at 12 FPS. The image, which is shown in 3.2(a), is captured with 640x480 resolution at 30 FPS. Furthermore the captured image or video can be used to create skeleton of the human body on the top of the image as shown in Figure 3.2(a).

Another characteristic of Kinect Camera is to take depth image and depth video which include the distance information of the object from frame of the Kinect Camera. The example of the depth image is shown in figure 3.2(b). The depth sense of the environment is the most important characteristic feature of Kinect Camera. Because the depth information of the environment allows the user to make many applications such as 3D reconstruction and object tracking.

There are 3 different available resolution to record a video by using depth feature of Kinect Camera. The all available resolutions which are 640x480, 320x240 and 80x60, use 30 FPS. In addition there are two possible range modes in order to compute the depth value. By the way the distance between Kinect Camera plane and the location of the object is given in millimeters by taking the Kinect Camera as a reference.

The first one is the standard mode which allow to compute depth value between 800mm and 4000mm properly and besides the second one, which is near mode, allows to compute the depth value in the range of 400mm and 3000mm. The out of range has to be considered to create the applications correctly. For example the depth value is unknown under 400mm or over 8000mm. The Kinect Camera refers the value too far between 4000mm and 8000mm for standard mode and as well the depth value is too far in the range of 3000mm and 8000mm for near mode. In addition the depth value is too near between 400mm and 800mm in the range mode [13].

The possibilities in the resolution and FPS for both RGB video and depth video and also the possibilities of the modes for depth video has to be chosen according to the demands in the application. Furthermore the available settings of the Kinect Camera shows the range of precision and accuracy.

3.3. Coordinate spaces

The coordinate frame of the Kinect is important to understand correctly the depth data which comes from inside of the Kinect Camera's vision area. If the Kinect Camera is considered as a human eye, the "z" direction is located to forward and back from the Kinect's eye and the "y" direction is located along to up and down from the eye point of

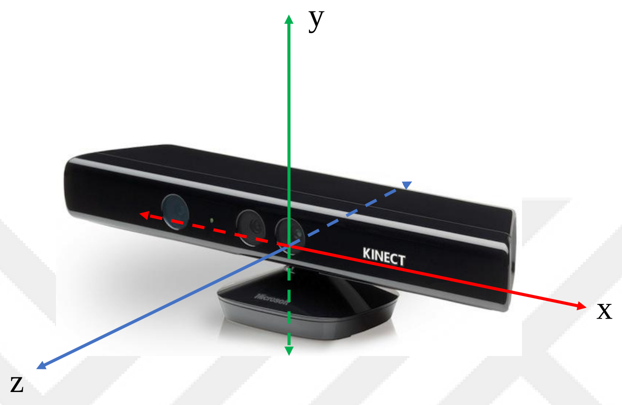


Figure 3.3: The coordinate frame for Kinect

the Kinect and lastly the "x" direction is located along to right and left from the eye point of the Kinect. The xyz coordinates can be seen in figure 3.3 clearly. Thus the correctly detected data by using the Kinect can be used in order to analyze this data.

Especially the Kinect coordinate system have importance while using the skeleton tracking because the distance of the skeleton points from Kinect Camera is captured based on the coordinate system of the Kinect Camera which is shown in figure 3.3.

4. IMPLEMENTATION

4.1. ROBOT ARM

The parts of the robot arm, which are used in the present project, were printed by using 3D printing technology and it is designed by Carlo Franciscone [14]. The robotic arm is an open source and can be downloaded as free software.

This robot arm is designed by considering the ABB IRB460 model industrial robot arm. In addition, the robot arm has 4 degree of freedom.

4.1.1. Printing all parts

The robot arm consists of the both 19 robot's parts for 3D printing and the other components such as servo motors, screws and nuts. In order to print the parts of the robot, Dimension BST 1200es model 3D printer, which has 254 x 254 x 305 mm volume for building, was used. The first thing is to upload the robot's parts to CatalystEX 4.1 software of the 3D printer in STL file format. The part list of the robot arm is shown in the appendix A and the non-printed parts of the robots arm is shown in Appendix A as well.

Before starting the printing, some parts of the robot has to be changed due to fact that some parts are not convenient to print. The main reason of this transformation are the

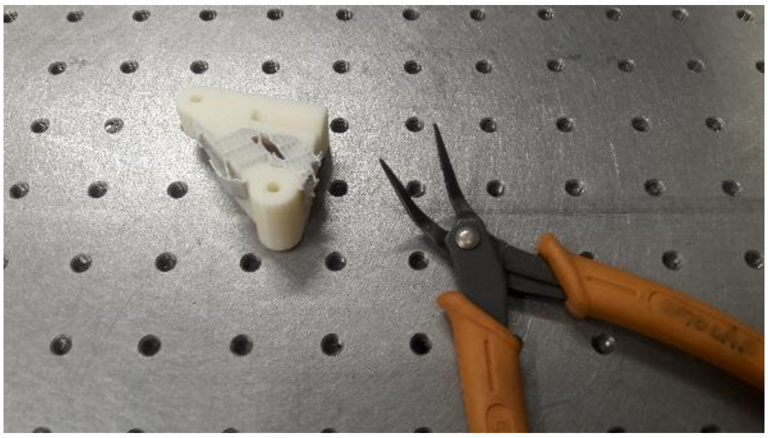


Figure 4.1: The part hole is covered with support material

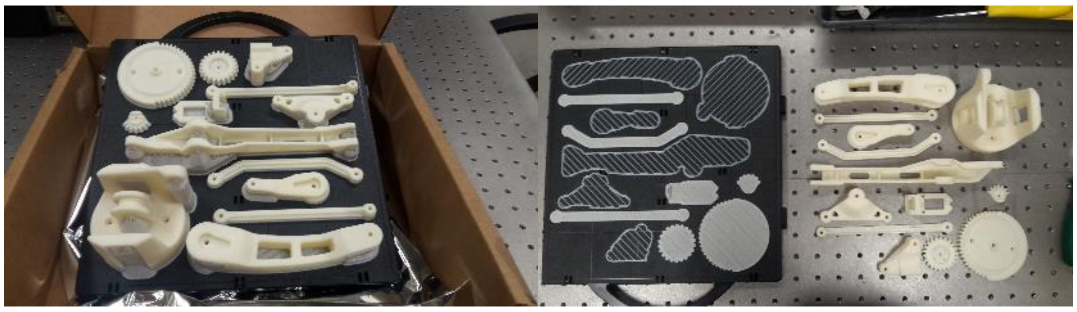
printer properties that are based on a breakaway support technology (BST) instead of a soluble support technology.

While the parts are under construction in the printer the modeling material which is ABSplus™ is melted to create the main material and in the same time the support material is used fix the main material in some points such as holes and girders.

The support material has to be extracted after the printing process such as figure 4.1. The extraction of the support material can cause a problem sometimes because the extraction of the support material is difficult in some points. The other reason of change the robot parts is the layer thickness availability of the 3D printer. The most thin available layer thickness is 0.254 mm [15]. This available layer thickness means that the part thickness cannot be thinner than 0.254 mm.

The diameter of a hole in the lower base part of the robot was changed from M3 screw's diameters to M2 screw's diameters due to the fact that the hole was disallowing the spheres way. In addition two holes in the main base part were changed. Before starting to print, unless the holes are changed, the extraction of the support material will be really difficult after printing. For this reason the holes depth was extended along the parts to extract the support material easily after printing. In addition the design program Solidworks CAD was used in order to change the robot's parts and then the changed parts were saved as STL file format as a condition for the printing process.

The software of the 3D printer which is CatalystEX 4.1 provide to simulate the printing process before printing the parts. Thus some information such as how many material will be used and how many hours the printing process will take can be estimated.



(a) (b)

Figure 4.2: (a) The parts of robot after printing (b) The parts of robot after extraction from support material

As well the position of the robot's components has to be in special order at the board of the 3D printer. The biggest part has to be in the right-up corner of the board and the smallest part has to be left-down corner of the board according to the beginning position of the 3D printer's head in order to decrease the printing process duration.

In the first printing process 13 parts of the robot were completed in 21 hours and in the second printing process 6 parts of the robot were completed in 9 hours. Totally 416 cm^3 model material and 79 cm^3 support material was used in the present project.

The main material is attached to the support material and the printer's plate when the printing process finish. Thus the support material has to be extract from the main parts. The support material is highly fragile by using a small screwdriver and pinchers. Shortly after the printing, the main material, the support material and the printer's plate are shown in figure 4.2(a). The same parts which are divided from each other are shown in figure 4.2(b).

Unfortunately, separating the parts from one to another is difficult and there may be problems in pulling out the material. The first problem came up while extracting the leg of the main base cover from the support material. The main base cover's leg didn't have enough thickness to print strongly due to the fact that the printer's layer thickness is not sufficient to print materials in an appropriate thickness. The leg consists of two layer main material that means 0.5 mm main material. Thus the leg of the main base cover is fragile even if the pushing force to the leg is slight broken leg was glued to the main base cover. In addition there was the extraction of the support material problem in both, one of the claw finger and the main base.

4.1.2. Assembly of parts

After printing all parts of the robot, the non-printed parts which are shown in Appendix-A, for the robot has to be supplied in order to start the construction of the robot arm. Unluckily some of the non-printed parts couldn't be supplied and they were replaced with the other materials which are able to do the same work as the original material.

Firstly 6.3 mm diameter spheres were used instead of 6 mm spheres in order to provide the rotation in the base part of the robot arm. Normally 25 spheres were planned to use but more than 25 spheres were used to fill the spheres path. In addition the other part, which is 6x17 mm bearing in normal design, was changed with 4x16 mm bearing in this project. Because the necessary bearing couldn't find and this replacement was implemented to solve the problem quickly.

The replacement in the bearings caused another problem due to the difference in the size between the two bearings. In the normal design of the robot, the main base flat part and the swivel element part is kept with each other by using the M6 nut and M6 screw. However the 4x16 bearing is not compatible with the M6 screw. Thus the M4 screw and M4 nut was used to assemble the main base flat part and the swivel element part. The problem was to use the M4 nut because the hole in the swivel element part was constructed for a M6 nut and when the M4 nut was used, M4 nut could move inside of the hole and also it was not possible to interfere to the hole due to the fact that the nut place in the swivel part was closed with other part.

For this reason it was difficult to find the M4 nut when the M4 screw was tried to fix to the nut. Eventually the M4 screw and the M4 nut was fixed each other after some trying.

The other problem which came up was the construction of the horns of the servo motor, which are providing the movement in the arm. The arm parts have a smaller space to fix than the horns of the servo motors. For this reason the servo motor's horns were made suitable for the parts of the arm of the robot by cutting and rubbing them with emery according to the size of the hole in the parts.

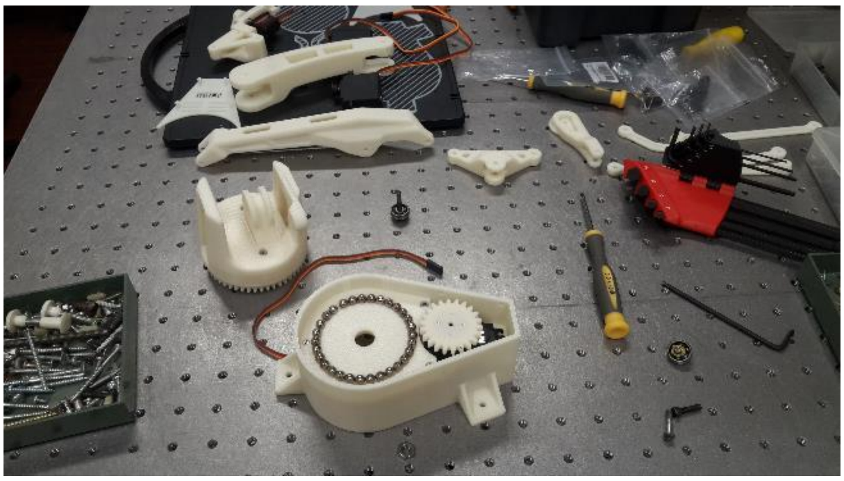
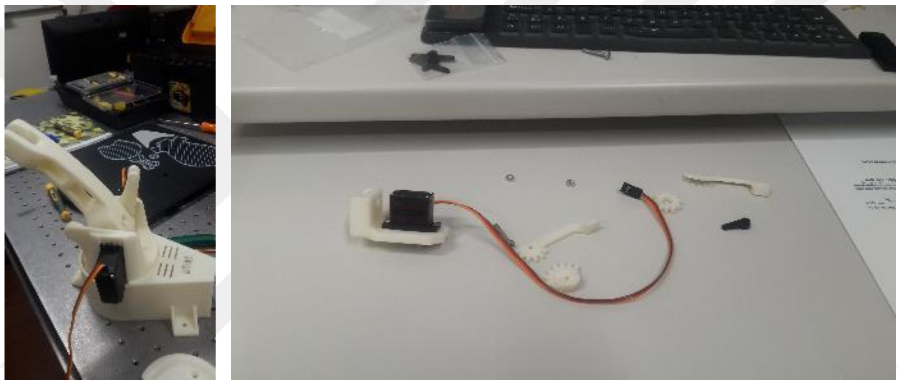


Figure 4.3: Assembling of the base

After preparing all parts of the robot the assembling of the parts started as following the steps are showing:

- First of all one servo motor has to be attached to the main base. Then the plate of the main base is attached to its place with 30 spheres on the path of the plate. In addition the drive plate of the servo motor and the gear are put together to the servo motors head. The assembled base is shown in figure 4.3.

- The base part and the gear part are fixed to each other by using two M3 screws and nuts in order to attach this part to the main base. Thus the base part of the robotic arm is completed.
- The main arm and the vertical arm part are put to their place in the base part and these three parts are attached to each other by using a 33 mm long M4 rod. Then the second servo motor is attached to the sides of the base part as shown in figure 4.4(a). Thus the rotation in the main base part is provided basically.
- The 135 mm link part is attached to the vertical arm part according to the end and



(a) (b)

Figure 4.4: (a) robot's base with main arm (b) the parts of the robot's gripper

the beginning point of the 135 mm link part. The other 13mm link with angle is attached to its place in the base part.

- The unattached side of the 135 mm link and the 135mm link with angle are attached to the horizontal arm part and the triangle link part respectively by using a M3 screw. In the same time the horizontal arm, the triangle link and the main arm are attached to each other by using a M4 rod.
- The 147 mm link part is attached to the triangle's hole which is available after attaching the other parts. Then the triangle part for the front of the robotic arm is attached to the end of the 147 mm link part and the horizontal arm part by using a M3 screw.

- Lastly the MG90s servo and the gripper part, which is shown in figure 4.4(b), are attached to each other and the created gripper is attached to the frontal triangle part of the robotic arm.

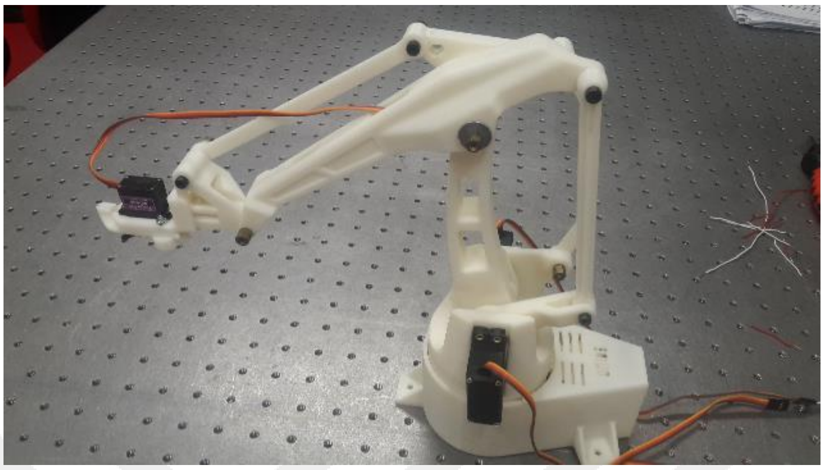


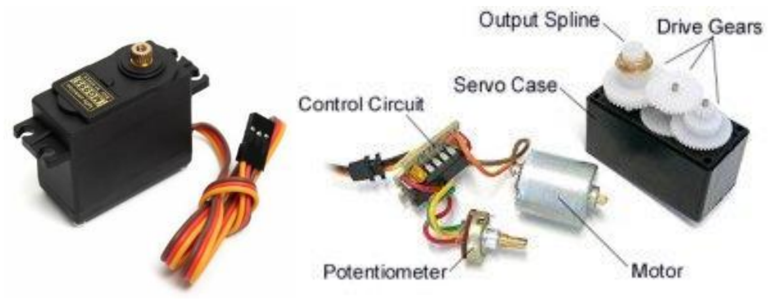
Figure 4.5: Completed Robotic Arm

After following all steps for creating the robotic arm, the robotic arm, which is shown in figure 4.5, is completed and it is ready in order to use it in this project.

4.1.3. Servo motors for the robot

Servo motors, which are electronic devices to provide the motion, can be used in many areas such as robots, small cars and home electronics in order to adjust the object position with high precision [16]. For example in the daily life the servos are used in DVD players, radio devices, computer fans and coffee machines.

Servo motors are particularly popular in research projects, because of their favorable price and they are easy to obtain via the internet. In this project the servo motors Tower Pro MG995 and MG90s were used in order to control the movement in the joints of the robotic arm. The functional principle of servomotors can be learned with a view to the internal structure of the servos. Generally the internal structure of the servo motor is formed with the help of a motor, potentiometer, control circuit and gears. One servo motor external view and detailed internal structure can be seen Figure 4.6(a) and 4.6(b) respectively.



(a) (b)

Figure 4.6: (a) a servo motor (b) servo motor internal structure in detail[16]

Servo motors are controlled by sending an electrical pulse with a modulation (PWM) signal from the control wire which is frequently the yellow one. Servo motors can be brought into the desired position by changing the pulse duration in the signal.

In this project the Arduino microcontroller is used and programed to give this signal to the servo motors. In addition the servo motor is able to turn 180° from the initial point to the end of the servo motor.

In the present project, three MG995 servomotors are used to ensure robot-based rotation and robotic arm joints. Further, a servomotor MG90 is used to control the movement of the gripper of the robot.

4.1.4. Control Circuit

In order to control the robotic arm and to provide the communication between the robot and the computer, Arduino Uno microcontroller was used.

Before the entire system was started, a single servo motor test was performed to simplify the control at the beginning without adding any additional hardware. This was done in order to recognize all possible errors in the system immediately. During the test run no errors occurred, so that the commissioning of the whole system could be started.

The second step for creating the system was to add the second servo motor in order to check the working conditions of two servo motors running at the same time. When the second servo motor was connected to the Arduino microcontroller, the servomotors and the Arduino began to behave strangely. The reason for this misbehavior was that the

Arduino microcontroller received the necessary working voltage via a USB cable from the Computer and the servo motors via the Arduino 5v pin.

The servo motors can behave extraordinary when the Arduino is plugged to the computer, because the servo motors draw high current when the servo motors started initially.

This high power request can cause the dropping of the voltage on the Arduino board and so the Arduino resets itself [17]. In the present project, this problem has been solved by adding a high value capacitor between the ground and the 5 V pin of the Arduino. Also, an 820 Ohm resistor was placed in parallel with the capacitor to discharge the capacitor after the loading

While using servo motors more than two, the Arduino is not able to give the necessary voltage to the all servo motors, because the Arduino has only 5 volts in spite of the fact that generally servo motor's operating voltage is between 4.8 volts and 6 volts. Thus external power supply was used to provide the necessary voltage for four servo motors.

Another important point is to connect the Arduino ground and the external power supply ground in the common ground, using an external power supply.

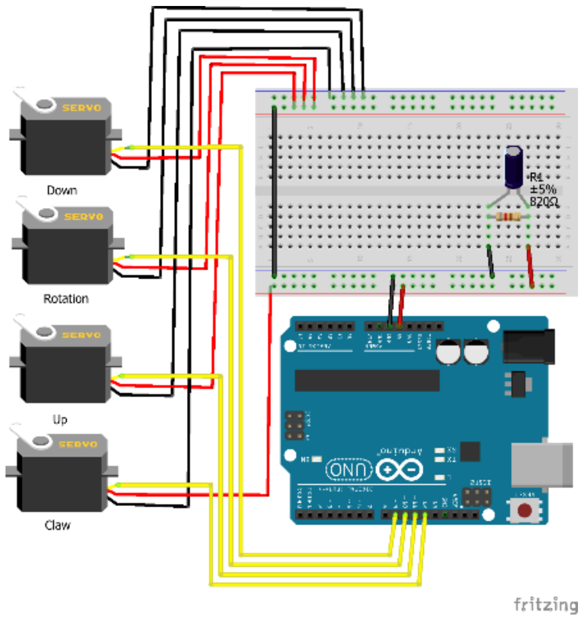


Figure 4.7: Control circuit for the robotic arm

The control circuit in this project, shown in Figure 4.7, consists of a breadboard, four servomotors, a capacitor, a resistor, an Arduino microcontroller and some connecting cables to unite the components together.

The external power supply, which is not shown in figure 4.7, should be such that the ground of the external power supply matches the ground of the Arduino. The voltage line of the external power supply may be connected to only three power cables of the servomotors presented by the red cables. Only the gripper servo motors were powered with an Arduino 5V-pin. The system requirement is respected for each connection. Thus the control circuitry of the system is ready to write the code to the Arduino and to use the servomotors for our needs.

4.2. SOFTWARE

In this project, Microsoft Visual Studio 2015 is used in C# computer language to develop the software, which is for controlling the robotic arm. Besides of Microsoft Visual Studio, Arduino software, based on C/C++ programming language, is used in order to program Arduino microcontroller.

4.2.1. Arduino (Robot software)

Arduino is a microcontroller that controls analog and digital systems. Nowadays, this is often used because it is inexpensive and it is an open source.

The programming of Arduino is also quite simple, if the user can use one of the programming languages C or C ++. In addition, there are several Arduino boards available for specific applications. In this project, Arduino Uno, the most widely used with many documentaries and sources, has been chosen as controller for these reasons [18].

To install the Arduino, first the requirements tools, an Arduino Uno card and an A-to-B connector USB cable must be available. Then the integrated Arduino development environment (IDE) can be downloaded from the official Arduino website. When downloading, the IDE is carefully selected based on the operating system of the computer. In this project the version 1.6.4 Arduino IDE for Windows OS is used. After the Arduino IDE has been installed, the Arduino hardware must be connected to the computer with a USB cable, and the port name of the Arduino hardware can be checked via the Device Manager, right-clicking on the computer icon.

The Arduino Microcontroller should be displayed in the "Ports (COM & LPT)" list, as all the devices currently plugged in are visible for the user. As soon as the name appears in the device list, the same name must be recognizable and selected under the Arduino IDE tool part. In addition it is important to select the correct name in the port, the board name of Arduino must also be selected correctly to identify the Arduino board in the computer. All these installations and identification steps make it possible to write and execute the code to use the Arduino applications.

Figure 4.8 shows the Arduino Uno pin structure with a USB port. Furthermore, the DC (DC) jacks, the 14x digital IN / OUT pins and the 6x analog IN pins are visible.

Although the Arduino is ready for programming, a startup cable, a USB cable, a resistor and some capacitors are required to create a whole working application. The reason for the additional components is that Arduino is a board, which cooperates with software and hardware.

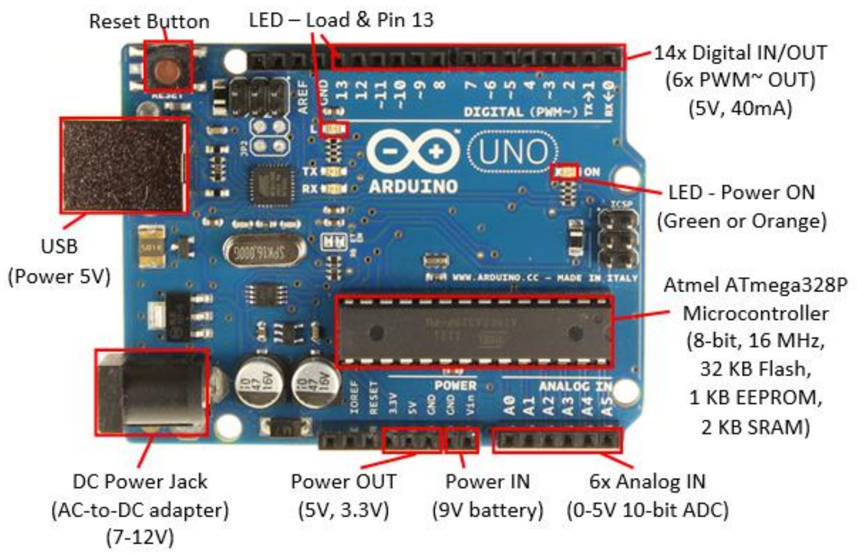
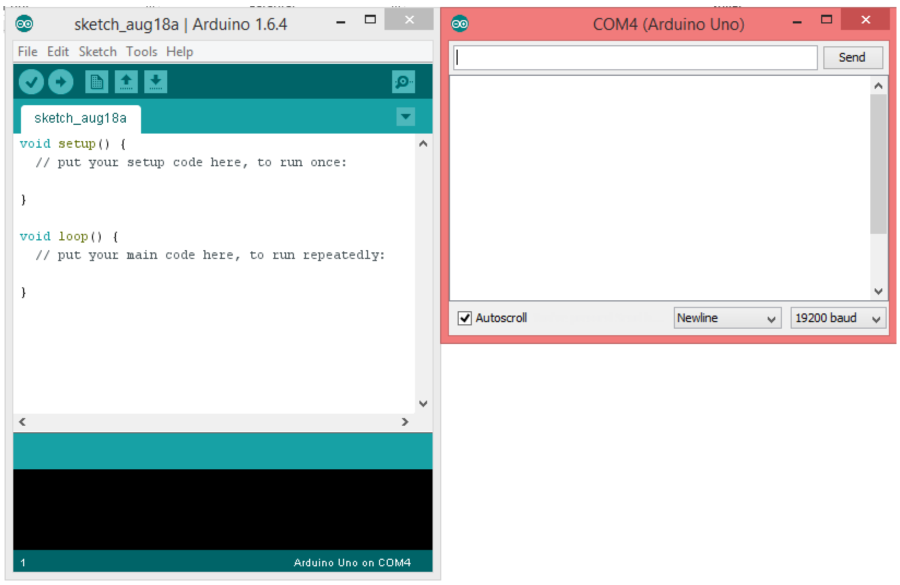


Figure 4.8: Arduino Uno pin structure

Arduino is used in order to control especially the servo motors and to communicate with the computer in the control system of the present project. Therefore, it can be considered as a bridge to transfer the data from the computer to the servo motors of the robotic arm.

The communication between the servomotors and the computer is carried out by means of the Arduino microcontroller by means of a serial communication method. For this reason, only one servomotor was initially controlled with the serial monitor, which is responsible for writing inputs or reading the output. As shown in Figure 4.9, this is available in the additional Arduino IDE monitor.



```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

☑ Autoscroll   Newline   19200 baud

Figure 4.9: Arduino IDE and Serial monitor interface

The angles which are in the limited range of 0 to 180 degrees can be written as input due to the operating range of the servomotor. To write the inputs, the values that come from the connected tools to the Arduino are also visible. At this point, the Arduino can read data from the external environment. For the present project, this point can be regarded as significant since the aim of the project is to use the Arduino as the communication channel between the computer and the servomotors of the robot.

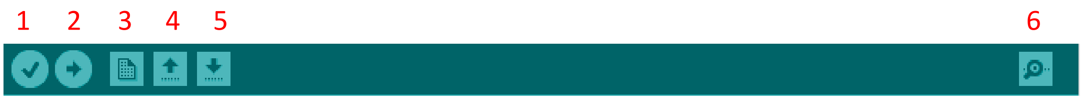


Figure 4.10: Top bar of the Arduino IDE

To open the additional serial monitor, the serial monitor must be clicked from the top bar of the Arduino IDE, shown in Figure 4.10 marked with number 6. In addition, Figure 4.10 shows the first button for compiling the Arduino code and check the code. The second button is used to upload the code to the Arduino hardware. The third button opens a new page for writing a new code. The fourth key is used to open an existing code from the computer, and the fifth key is used to store the code that is currently open in the work area.

The baud rate, which is the change in the number of symbols per second, must be selected from both sides of the Arduino and the computer to establish the correct connection. The available baud rate values are 2400, 4800, 9600, 19200. In this project a baud rate of 19200 was used. In the Arduino code, the serial connection must be started as the serial connection code with the selected baud rate is written into the Arduino IDE.

After the connection between the computer and Arduino has been made, the path must be found to obtain the angles for the joints of the robot, and then the recorded angles must be sent to the servomotors to provide the desired movement in the robot. The Arduino code written in Appendix B can receive both the angles from the computer and send the angles to the servomotors.

In the following, the Arduino code is explained in detail in order to understand the logic behind the program. The code can be divided into three parts and is explained step by step.

The first part of the program contains the necessary libraries, the variables and the identification of the variable type, such as string, integer. Servomotors can also be specified with servo command.

The second part of the program contains the "Setup" function, which is delivered with Arduino by default. The code written inside the "Setup" function is executed only once. This happens when the Arduino board starts to work.

The third part of the program contains the standard function of the Arduino, which is called the "loop" function. It is a cycle that can be understood from its name of the function. The "loop" function allows the repeated execution of the code in the "loop" function. After general information about the program parts, the code lines can be explained, quasi, what the code assignment is exactly.

At the beginning of the Arduino code shown in Appendix B begins with library codes required to write certain codes. In contrast to other programming tools, Arduino IDE generally helps the user to write simple codes, such as reading data entered from the computer screen or printing data on the computer screen without adding a library head to the Arduino codes.

In order to control the servo motors, only the library associated with the servo motor is added in this way. The "Servo" library is also included in Arduino IDE packages. In addition to the "Servo" library, the “ServoEaser” library is added because of the servo control. The reason for this is that the servo movement is so fast that the robot arm cannot be controlled improperly without the servo motors being relieved. The open source library "ServoEaser" is an additional library. The "ServoEaser" library is downloaded [19]. Therefore, this library must be added to the Arduino IDE to define the library. To add the library, you must copy and paste the downloaded “ServoEaser” folder into the Arduino library folder, where ~ \ Documents \ Arduino \ libraries are the default location on Windows. After you have added all necessary libraries, you must define the variables at the beginning of the Arduino code. The most important variables are angles, which are defined as string and servo motors, which are defined as servo.

The second part of the program, which includes the "Setup" function, starts with the code of the serial port, which starts the serial connection between the Arduino and the computer with a baud rate of 19200. The "Setup" function executes the code only once as described above. The pin numbers of the servomotors connected to the Arduino hardware must be defined in this section with the "attach" code. In this way, the Arduino will be able to understand which servo motor is connected where. The servo motor initial value is defined in the "Setup" function. In addition, the servo bucket function for each servo motor is activated in this section.

After explaining the "setup" function part, the most important part of the "loop" function is the function "loop" contains the codes that perform the actual tasks of the program. The "loop" function begins with the codes that update the servo bucket function in each cycle. Then there is the statement “if ...” with which the condition can be controlled. When the serial connection is available, the program reads the data coming from the computer over a serial connection in the present state and stores the recorded data as characters.

Inside of the “if serial connection is available” statement, there is another “if…else…” statement. The second “if” condition is used to divide the data which includes the angles due to the fact that the angles comes all together in one string. The “if” condition finds the star “*” figure in the beginning of the string and then the string is divided by using the slash “/” figure to find the angles. For example the string format is like this “*angle1/angle2/angle3/angle4”. Thus the angles are ready to write on the servomotors. In order to write the servo motor’s angles there is additional function in the end of the

Arduino Code. Actually the code for writing angles to the servo motors includes the easing codes which provide the movement of the robot smoothly.

Lastly inside of the “loop” function there is another code that allow to map the calculated angles to desired angles for servo motor. Because the limit of the human joint’s angle is different than the limit of the robotic arms joint’s angles. For this reason the range of the calculated human joint’s angles has to be mapped to the range of the robotic arm joint’s angles. For example the elbow of the human joint’s angle is calculated in the range of 50 and 170. However the same possible angle for robotic arm is in the range of 20 and 154. In this example the range is transformed from 50-170 to 20-154.

In the present project the robotic arm is controlled with two method that are dot product of the vectors and inverse kinematics of the robotic arm. The Arduino code is almost same for these two different method. The only changing part in the code is the range of the angles for mapping.

4.2.2. Microsoft Visual Studio (Kinect software)

The program Microsoft Visual Studio was used to write the code in the C # programming language and also to control the user interface. The user interface not only serves to visually follow the process but also contains the algorithms and libraries in the background of the program. This allows the robot arm to be controlled and the Kinect camera functions can be used.

4.2.2.1. Configuration for Kinect camera

First, some adjustments has to be made to use the Kinect camera in our applications. The Kinect has to be on a stable surface without any vibration or noise. After allocating the position of the camera, the hardware and the libraries will be downloaded. The libraries can be downloaded in the Microsoft Official Library and as well as an open source library. In the present project the Microsoft Kinect SDK 1.8 is used for the setting of the application. Furthermore, the operating system Windows 10 is used during the whole project.

Microsoft Kinect SDK version 1.8 is downloaded from the official Microsoft Web site. Before starting the installation, make sure that the Kinect camera is not connected to one of the USB ports on the computer. After the installation of Kinect SDK v1.8 has been successfully completed, the Kinect sensor can be connected to a USB port of the computer and to an external power source. Now the Kinect should work properly. In addition, the status of Kinect can be checked by checking the device list in Device Manager.

After installing and connecting the device, the program "Kinect for Windows Developer Toolkit v1.8" is downloaded and installed. This includes sample source code, tools, and other resources for the user.

The applications in this work are developed with Microsoft Visual Studio 2015. Therefore, the installed Kinect libraries must add Visual Studio references to use these libraries in the source code. First of all, a new Windows Presentation Foundation (WPF) application is created, as it is more suitable for 2D and 3D visual applications.

After creating WPF, the xaml, which is more used for the design of the visual diagram of the program, appears. In addition, the libraries can be added to the references written at the under of the Solution Explorer menu.

To add libraries to references, just click on the right mouse button while the "References" and "Add References" button is selected. The reference list is then displayed, and Microsoft.Kinect.dll must be selected to add it to the library. Now the Kinect libraries are available, but it is still not possible to write a code. In addition, these libraries must be written at the beginning of the code, which contains all necessary libraries

4.2.2.2. Skeleton tracking

The Microsoft Kinect camera provides the ability to capture the depth information of the environment. However, the depth data do not automatically contain the joint coordinates of the human body. In order to take over the joint coordinates of the human body, the Kinect camera's skeleton tracking function must be activated by using the Microsoft Kinect SDK.

The Kinect Camera with SDK v1.8 is able to detect six people and is as well able to track one or two people in the same time. In other words there are two possible functions which can be described as “tracking” and “taking the position only” to take information about the user. The tracked user data contains the skeleton of the user which represent 20 joints of the human body such as head, hand and foot that are shown in figure 4.11 [20]. The joints of the user are represented in a 3D coordinate space of the Kinect Camera. In skeletal tracking, the joints are expressed in meters, instead of millimeters, which are indicating the depth.

The Kinect Camera is taken as a reference point (0, 0, 0) and in this way one of the twenty joints is shown point (x, y, z). According to the Kinect Camera, “z” value increase. In

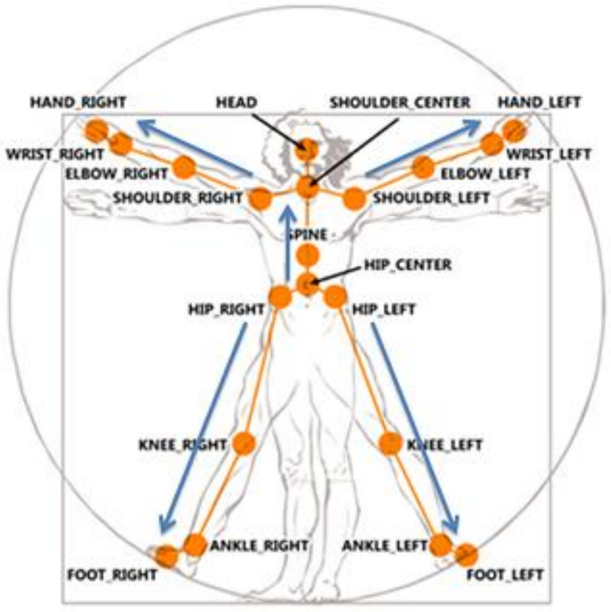


Figure 4.11: Twenty joints of user body

addition, the user must be in the area of the Kinect vision so that the joint data can be clearly recorded. The user can be in a standing or sitting position to be recognized by Kinect Camera. In the sitting position, however, only the upper body with 10 joints, the upper limbs and the head is recorded. The user must move easily to be recognized by Kinect Camera.

After the all adjustments which are expressed in the configurations for the Kinect camera, some code has to be written to the code block in the WPF application in order to use the skeleton tracking feature of the Kinect Camera. It should be noted that the WPF application has two code blocks. One code block is used to develop the interface of the program and another code block, which includes the most important part of the program, is used to utilize the Kinect Camera feature, to calculate the joint angles of the user and to provide the serial connection between the computer and the Arduino. In addition the programming language of one code block is C# and the other one is XAML.

In the beginning of the code, which is shown in Appendix C, the variables are defined for the skeleton tracking. Then the Kinect Camera’s features such as skeleton tracking are tried to activate in the program [21]. The condition of the activation of the Kinect Camera depends on the configuration of the Kinect Camera. If the Kinect camera is ready to take an image, the skeleton stream can be enable. In the same time the depth image and color image can enable in order to show in the interface of the program. After the color image, the depth image and the skeleton stream enables. The program is able to detect the user which is in the field of Kinect Camera’s vision. Thus, the user's joint points are recognized to produce the skeleton in the interface program.

To describe the skeleton drawing in detail, first the lines are generated from one detected joint point to another recognized joint point, with respect to the natural skeletal figure of the human body and using the "CreatFigure" code. Then, the created lines are added to the "grid" on the XAML side to display them in the program window. The XAML code of the program is given in Appendix D.

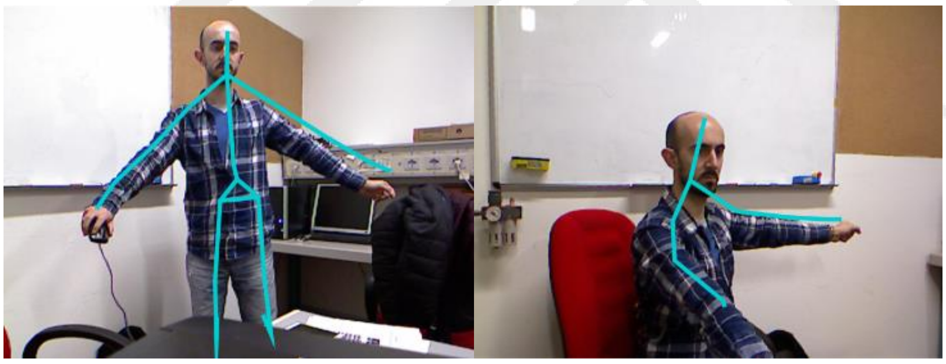


Figure 4.12: (a) Skeleton tracking of the standing user (b) Skeleton tracking of the seated user

The created skeleton is shown in Figure 4.12 (a) for the standing user and in Figure 4.12 (b) for the sitting user. By default, skeleton tracking is enabled for the standing user in the current project.

A "checkbox" is added to the program's interface to allow skeleton tracking for the seat position. When "Checkbox" is selected, skeletracking is active for the seated position. The skeletal tracking of the standing position makes it possible to draw the whole skeleton of the user, which is limited by 20 joints. However, the sitting position allows only the upper extremities and the head, which is limited by 10 joints.

4.2.2.3. Hand Pointer

The skeleton tracking, which is provided by using Microsoft SDK, allows to find the hand coordinates of the user but it doesn't allow to detect the hand movement such as an open or closed position of the hand. Detecting an open or closed position of the hand it is necessary to control the robotic arm's gripper. For this reason the example, which is published by Microsoft, is used to provide the recognition of the hand movement [22].

In order to use the hand detection, first the library "Microsoft.Kinect.Toolkit" is added in the beginning to the program which is given in Appendix C. In addition the file "KinectInteraction180_64.dll" is downloaded and added to the folder which includes the source files of the program. After the configuration of the libraries, the grip and release of the "hand pointer" function is added to the "KinectRegion" which is a XAML code part. The button, which is put completely transparent in the interface of the program, is used to detect the movement of the user hand's movement. When the hand of the user is closed, the button is clicked. Two functions which are "button_click" and "button_notclick" are implemented to the program. When the hand is closed, the function "button_click" is active. Then it sets the gripper's angle of the robotic arm to 10 degrees. Otherwise the gripper of the robotic arm servo motors sets 80 degrees. In conclusion this method is used in order to detect the movement of the hand and it is implemented to the both inverse kinematics program and dot product program.

4.2.2.4. Calculation of the joint angles

The calculated joints, which are showing the joints of the human body in the range of the camera, are used to calculate the angle that are shown in figure 4.13. Point A is the joint

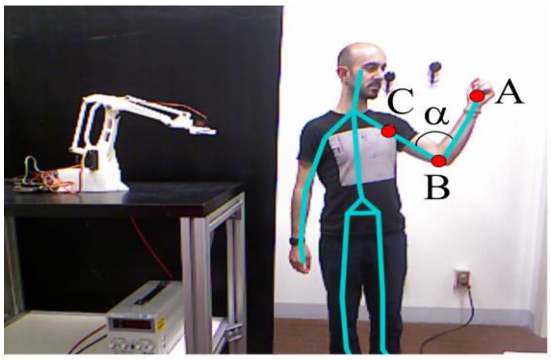


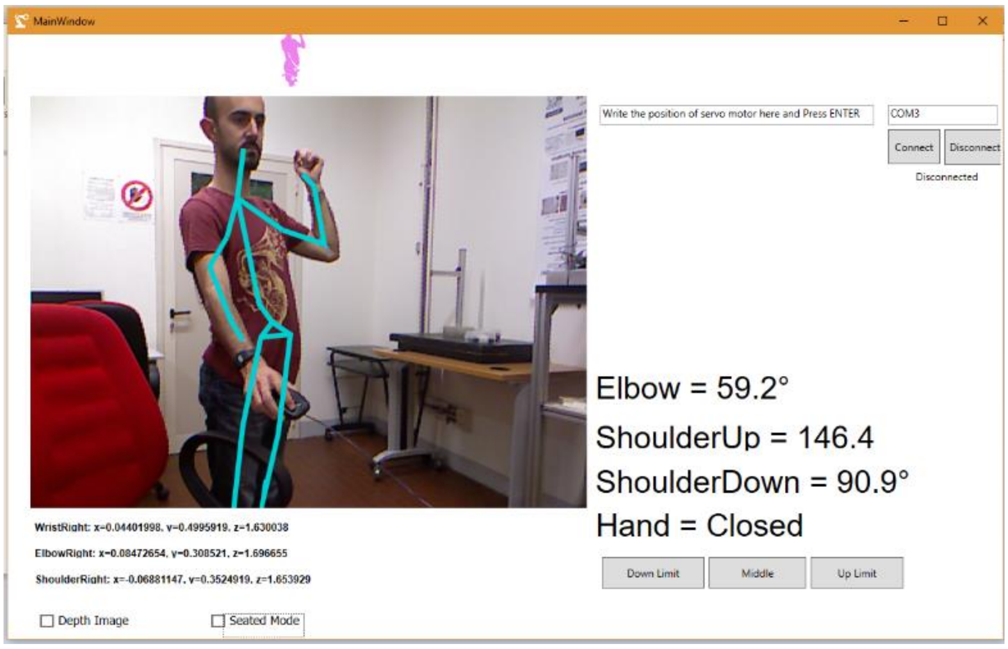
Figure 4.13: Angle between the vectors

of the right wrist, point B is the joint of the right elbow and point C is the joint of the right shoulder.

These joints are taken with the help of the skeleton tracking and then they are shown in the program's display screen in real time. In the present case, actually two vectors can be created and then the angle can be calculated easily by using the dot product of two vectors. Dot product is given with equation 4.1.

$$\overrightarrow{BA} \cdot \overrightarrow{BC} = |\overrightarrow{BA}| \cdot |\overrightarrow{BC}| \cdot \cos\alpha \tag{4.1}$$

In three dimensional space, $\overrightarrow{BA}$ and $\overrightarrow{BC}$ vectors are calculated by using A, B and C points, which are taken from the skeleton tracking and then the vectors length is calculated. Now we have all information to take the angle from the equation. All these mathematical calculations are executed in the background of the program in real time.



```
Elbow = 59.2°
ShoulderUp = 146.4
ShoulderDown = 90.9°
Hand = Closed

WristRight: x=0.04401998, y=0.4995919, z=1.630038
ElbowRight: x=0.08472654, y=0.308521, z=1.696655
ShoulderRight: x=-0.06881147, y=0.3524919, z=1.653929
```

Figure 4.14: The interface of the program for dot product method

First the vectors are created as vector 1, 2, 3, 4. Vector 1 is from right the wrist to the right elbow, vector 2 is from the right elbow to the right shoulder and vector 3 is from the right shoulder to the shoulder center and vector 4 is from the right shoulder to the right hip. By using these four vectors, the angles, which are the right elbow angle, the right armpit and the right shoulder are calculated by using the function called "Angle". The angle function, which is given in Appendix C, is developed in this project to calculate the angle from two vectors.

The calculated joint angles are shown in the interface of the program which is created by using XAML code which is given in Appendix D. The interface program is created to give the possibility to the user to follow the angles and to control the serial connection between the computer and the Arduino. The created program is shown in figure 4.14.

The interface program includes a big RGB image, which can change optionally to depth image taking (in left side of the program's interface).

Below the image, there are three "labels" to show the coordinates of the right wrist, the right elbow and the right shoulder of the user. In the right corner of the program there are two "buttons" to connect or disconnect the serial connections and as well the sending and receiving angles are tractable from the label that is located close to the connection buttons. The four labels are showing the calculated angles of the joints and the position of the hand that is closed or open.

4.2.2.5. Calculation of the robot's kinematics

In this method, only hand coordinates were used, which were recorded using skeleton tracking to calculate the inverse kinematics. Using only one hand coordinate causes a high work of the calculations in the background of the program. The program, which was developed in the present project is able to both calculate the joints angles of the robot according to the user's right hand coordinate and send the calculated angles to the robot by using the Arduino microcontroller in real time.

While making the inverse kinematics calculation of the robot the geometric approach was used to find the angles in the robot's joints. When the calculated angles were used to move the robotic arm the gripper of the robotic arm would move the desired coordinate. Thus the gripper of the robotic hand would track the user's right hand in its workspace area. The robot's workspace area acts like a mirror of the user's right hand workspace area. However the difference between the robot's workspace limit and the user's hand workspace limit has to be considered.

In order to start the calculation of the inverse kinematics based on the geometric approach, the geometric notation of the robotic arm has to be created. The geometric notation of the robotic arm, shown in figure 4.15, includes the joint angles and the point P(x, y, z), which shows the gripper coordinates. Thus the inverse kinematic equations of the robotic arm can be created by using the geometric notation. In addition while creating the equations, a Polar coordinate system is used and then the coordinate system is switched to the Cartesian coordinate system.

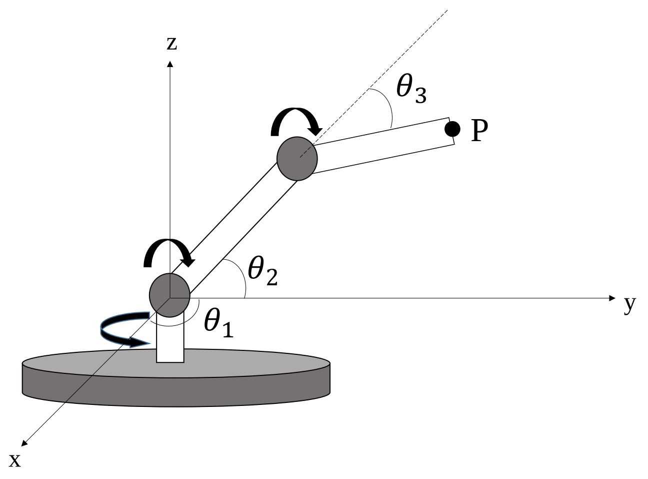


Figure 4.15: Geometric notation of the robotic arm

$$x = R\cos\theta_1 \tag{4.2}$$

$$y = R\sin\theta_1 \tag{4.3}$$

$$z = A_2\sin\theta_2 + A_3\sin\theta_3 \tag{4.4}$$

$$R = A_2\cos\theta_2 + A_3\cos(\theta_2+\theta_3) \tag{4.5}$$

The inverse kinematics equations based on geometric approach are created by using a geometric notation of the robotic arm in order to find the angles which are $\Theta_{1,}$ Θ_2 and Θ_3. The known variables are point P(x, y, z) and the lengths of the robot parts to calculate the angles. The Equation 4.2, 4.3, 4.4 and 4.5 are used to leave alone the joint angles of the robotic arm. In addition the equations have to be transformed from polar coordinate to Cartesian coordinate.

While making these calculations, the aim is to find the angles type of the variables x, y and z. Thus in order to find Θ_1, the equations 4.2 and 4.3 are used to leave alone the Θ_1. The equation for Θ_1 is shown in equation 4.6.

$$\theta_1 = \tan^{-1}\frac{y}{x} \tag{4.6}$$

In figure 4.16, the geometric notation of the robotic arm is shown in detail to create more trigonometric equations for decreasing the unknown variables. Especially the figure 4.16 is important to find Θ_3.

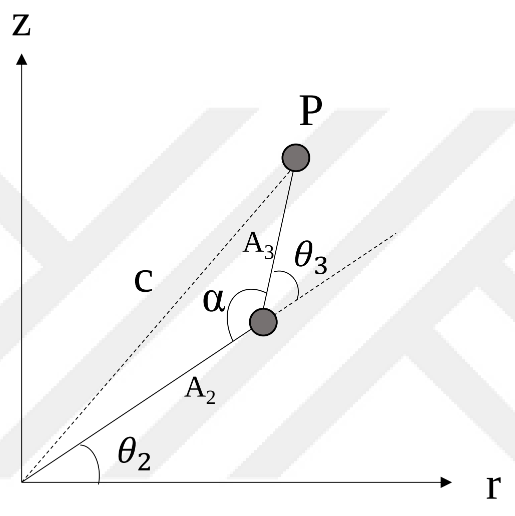


Figure 4.16: Geometric notation of robotic arm in z-r coordinates

$$c^2 = R^2+z^2 \tag{4.7}$$

$$c^2 = A_2^{\ 2}+A_3^{\ 2}-2A_2A_3\cos\alpha \tag{4.8}$$

The equations 4.7 and 4.8 are created by using the geometric notation of the robotic arm. The cosine theorem is used to find the equation 4.8 and then α is left alone due to the fact that Θ_3 and α complete the 180 degree that is explained in equation 4.9. After finding the cosα from equation 4.8, the solution of α can be located to the equation 4.9 and so Θ_3, which is shown in equation 4.10, is gotten.

$$\theta_3 = 180\text{-}\alpha \tag{4.9}$$

$$\theta_3 = 180 - \cos^{-1}\left(\frac{A_2{}^2+A_3{}^2-(x^2+y^2+z^2)}{2A_2A_3}\right) \tag{4.10}$$

In this way, Θ_1 and Θ_3 are found by using the geometric principles. The Θ_2, which is the only missing angle, can be found by using the same principles that have been used already for Θ_1 and Θ_3.

$$z = A_2 \sin\theta_2 - A_3\left(\frac{A_2{}^2+A_3{}^2-(x^2+y^2+z^2)}{2A_2A_3}\right) \tag{4.11}$$

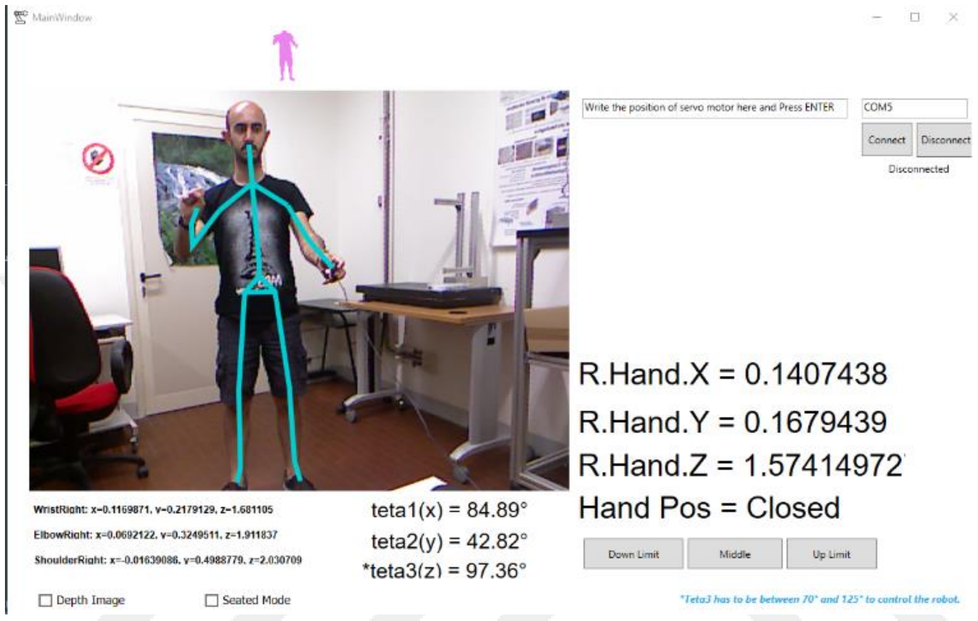
$$\theta_2 = \sin^{-1}\left[\frac{z+A_3\left(\frac{A_2{}^2+A_3{}^2-(x^2+y^2+z^2)}{2A_2A_3}\right)}{A_2}\right] \tag{4.12}$$

The calculated Θ_3 is put in the equation 4.4 to find Θ_2 as can be seen in the equation 4.11 and then the Θ_2, which is shown in equation 4.12, is left alone. Thus, all angles for the robotic arm are provided to implement them in the software.

The interface of the program developed for the inverse kinematics method is very similar to the interface program for the dot product method. As well, the background of the program, which includes the code that provides the skeleton tracking, has the same appearance. Only the function, which includes the inverse kinematics solutions, is different than the program of the dot product method.

Three functions, which are inverse kinematics 1, 2 and 3, are created in the program in order to calculate Θ_1, Θ_2 and Θ_3. The inverse kinematics functions are given in Appendix C. After the calculation of the angles, the angles are converted to “string” and then all of them are put in a label. In addition for the position “open hand” it is defined with 80 degrees and for the position “closed hand” it is defined with 10 degrees. As well the defined hand position is put close to the other angle values in one label. In the end the created label will be “teta3 + "/" + teta1 + "/" + teta2 + "/" + handStatus + "*"”. In the chapter of the Arduino software it will be explained more in detail why the label should be like this. Thus the main objective of the inverse kinematics is provided.

In addition while sending the label to the Arduino, a “DispatcherTimer” which is the default timer function in the Visual Studio, is added. The function of the timer is to send the label in a controlled manner. The timer is set in a rhythm of 2 seconds. Then the serial port’s sending function is implemented to the timer function. In this way the angles will be send in every 2 seconds to the Arduino.



Write the position of servo motor here and Press ENTER

COM5

Disconnected

WristRight: x=0.1169871, y=0.2179129, z=1.681105

ElbowRight: x=0.0692122, y=0.3249511, z=1.911837

ShoulderRight: x=-0.01639086, y=0.4988779, z=2.030709

teta1(x) = 84.89°

teta2(y) = 42.82°

*teta3(z) = 97.36°

R.Hand.X = 0.1407438

R.Hand.Y = 0.1679439

R.Hand.Z = 1.57414972

Hand Pos = Closed

*Teta3 has to be between 70° and 125° to control the robot.

Figure 4.17: The interface of the program for inverse kinematics method

After the inverse kinematics functions other desired functions are implemented to the program. The interface of the program, which is shown in figure 4.17, is showing this. In the right side of the interface program, the right hand’s x, y and z coordinates are expressed according to the frame of the Kinect Camera. The calculated angles from the right hand’s coordinates can be seen under of the RGB image. The other features in the interface program are the same as explained in the interface program of the dot product method.

5. EXPERIMENTAL RESULT

The experimental part expresses the performance of the systems which is made in the present project. While doing the experiment to analyze the system and to take data, the whole system has to be considered with all pieces. Namely these pieces are the Arduino microcontroller, the programs for the systems and the robotic arm since all parts have an interaction with each other and influence each other.

The target of the first experiment is to detect the performance of the program which is written in order to find the joint angels of the human body by using the dot product method. The experiment is done by using three different arm positions which have presented by three different elbow angles. The user arm is fixed to a stick to decrease the error which is caused by the user. The user's elbow is taking a position of 90°, 180° and a position where the arm is flexed as far as possible and is pulled up to the upper arm. The user's elbow positions are shown in figure 5.1.

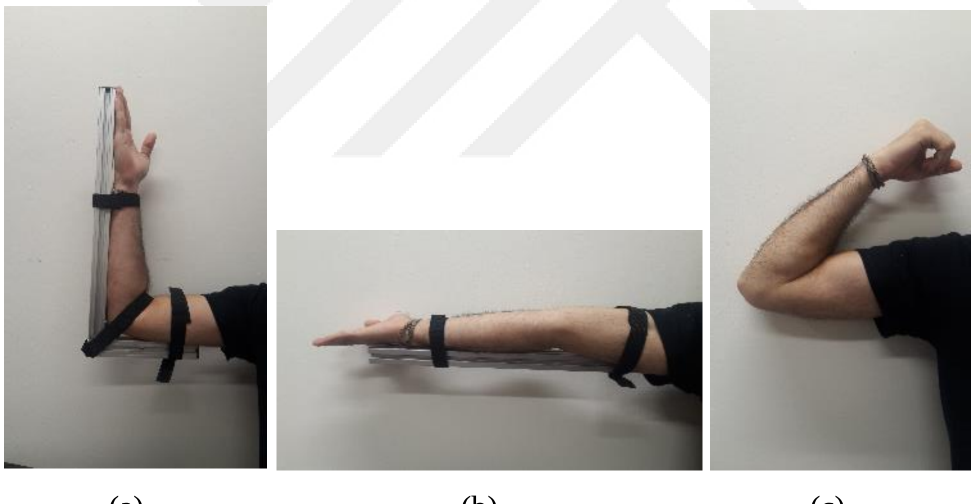

(a) (b) (c)

Figure 5.1: (a) the user arm is in 90 degree position (b) The user arm is 180 degree position (c) The user arm is flexed as much as possible

The user remains in front of the camera and aligns his arm clearly in the direction of the Kinect camera. The program calculates the angles using the dot product method. The arm of the user is moving in the area of the Kinect Camera's vision for each specified elbow angle in order to record the angles from different region of the Kinect Camera's vision. The angles are recorded to the text file in the computer by using the same program which calculates the angles. 900 values are saved for each elbow position. The graph of the elbows angles are shown in figure 5.2.

In some points the failures can be seen in the graph due to the fact that the arm stays in front of the user or behind the user. Thus the taken angle values are different than the expected angles values which are 90° and 180° at some points in the graph.

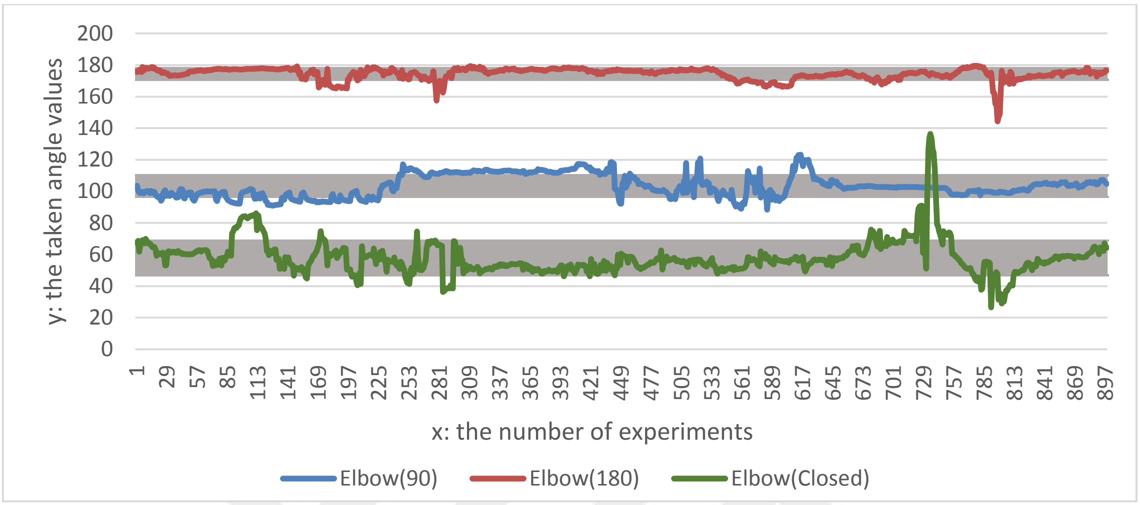


Figure 5.2: Graph of the elbow angles

The mean value, the standard deviation and the median of the taken angle values, which are shown in table 1, are calculated in order to analyze the data set more in detail. Basically the mean value can be explained as an average value of the data set and the formula of the mean value is shown in equation 5.1. The median is the middle value when the data set is made ascending sort. The standard deviation, which is shown in equation 5.2, measures the variance for the data set [23]. The standard deviation is an important element for the consistency of the system. For this reason the standard deviation is an expected small value in the present project.

$$\bar{x} = \frac{\sum x}{n} \tag{5.1}$$

$$\sigma = \frac{\sum (x-x^2)}{n-1} \tag{5.2}$$

"$\bar{x}$" represents the median value in equation 1 and "x" represents each angle value in equation 5.1 and 5.2. "σ" represents the standard deviation value in equation 5.2 and "n" shows the total number of the angle values in equation 5.1 and 5.2.

The angle for the position where the elbow is closed is found 58°±11. Thus the closed elbow can be from 47 to 69. The reason of the inconsistency is that the user's closed elbow occupies a small area in the vision of the Kinect camera and makes the program unable to make a correct skeleton detection.

Table 5.1: Mean, Standard Deviation and Median for three different Elbow Angles

	Mean	Standard Dev.	Median
Elbow(Closed):	57,87	10,99	56,32
Elbow(90):	103,39	6,95	102,57
Elbow(180):	174,41	3,78	175,44

The expected angle values are calculated with the values of 103°±7 and 174°±4 instead of 90° and 180° respectively. This inconsistency is due to the same reason as it was for the closed elbow, namely that the elbow occupies a small area in the Kinect camera vision and the program is not able to perform a correct skeleton recognition.

However the standard deviation of the elbow position of 180° is not as big as the standard deviation which occurs for the position of the closed elbow. This result shows that a better skeleton detection is obtained by using the skeleton tracking program, if the user's arm is shown more clearly in the vision of the Kinect Camera. In addition the error between the expected angle values and the taken angle values is 6° for a 90° elbow position and 4° for a 180° elbow position. This error is not so small but it is acceptable and it can be considered chronic error.

The dot product method, which is used to calculate directly the joint angles by using the right arm's joint coordinates, is tested with the calculation of the mean, the standard deviation and the median values. Unfortunately, the inverse kinematics method cannot be tested by using the same test for the dot product because in the inverse kinematics method the angles are not calculated in the same way like in the dot product method. For this reason, the inverse kinematics method is tested with creating an experimental environment, which is shown in figure 5.3, to verify the repeatability and accuracy of the remote control system.

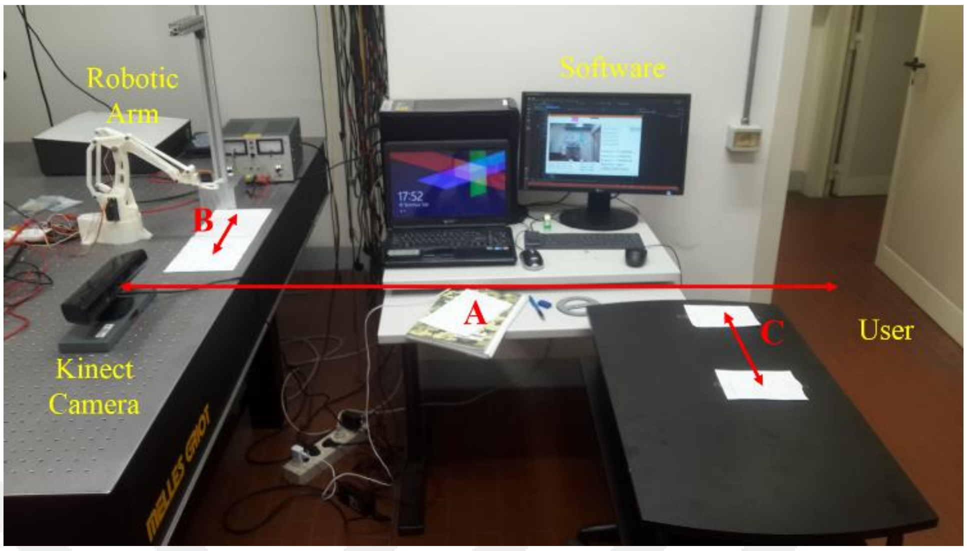


Figure 5.3: Experimental Setup for inverse Kinematics method

The user is standing in front of the Kinect Camera in a particular distance. The A, B and C distance is 162 cm, 15 cm and 31 cm respectively in figure 5.3. While the program is working, the user moves the right hand between the papers which are shown with the distance of C. The camera is detecting the movement the user's right hand. In the same time the inverse kinematics method program is calculating angles and sending these angles to the servo motors by using the Arduino. Finally the robotic arm moves in the field of distance B according to user movement.

While making this experiment, a pen is attached to the gripper of the robotic arm. The pen is drawing a point to the paper, which is located in the field of B, when the user put his right hand to the paper, which is located in the field of C.

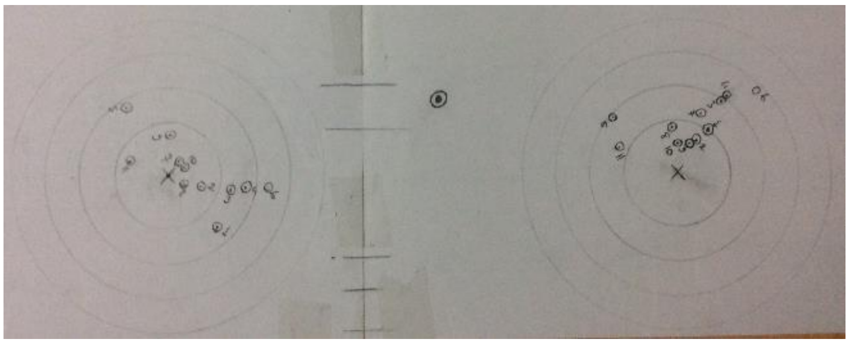


Figure 5.4: Accuracy and repeatability of the robotic arm

The experiment is repeated eleven times. The taken points, which are shown in figure 5.4, represents the accuracy and the repeatability of the robotic arm. The expected point is shown with cross sign in the figure 5.4. The results express the acceptable accuracy and the repeatability because the robotic arm’s movement has 1 degree accuracy due to its design. In the background of the program is a 5 degree waiting time to decrease the robot’s vibration.

6. CONCLUSION

The remote control system was developed successfully in the present project by using two different methods which are the dot product method and the inverse kinematics method. The main object was to develop the program in order to provide the motion control of the robotic arm remotely.

First of all the Kinect Cameras features were trained to start to take information about the user's joints. Then by using the feature of skeleton tracking of the Kinect Camera the joint coordinates of the user were provided in real time. After receiving successfully the data from the Kinect camera, the robotic arm's parts were built in a 3D printer. The robotic arm was built by using the printed parts and also the non-printed parts such as the servo motors and the screws.

In this way the data which was taken with the Kinect camera has to be sent to the robotic arm in order to control the robotic arm's movement by the user's movement. Thus the Arduino microcontroller was used to provide the connection between the computer and the robotic arm.

The dot product method and the inverse kinematics method were used to calculate the angles in order to move the robotic arm according to the user's movement. These methods were implemented to the program which was developed in Visual studio. In the same time the Arduino software was developed to send the calculated angles from the computer to the servo motors of the robotic arm.

The control system was created by the following the steps. Then some experiment were applied to the created control system in order to verify how successful the control system works. The results of the experiment showed that the inverse kinematics method is better than dot product method in order to control the robotic arm remotely because the accuracy of the inverse kinematics method was showing better results. As well the program of the inverse kinematics method is user-friendly because the user is using the only the right hand instead of the right arm's joints.

In the end of the present project the developed system came to the same level of the literature in this field. The implementation of the inverse kinematics to the remote control system and dot product method was applied in the same project. Thus the present project gave the possibility to compare the two different methods with each other.

In addition the present project was developed with well-priced unites and the control system can be implemented to an industrial robot system easily.

For future work, the robotic arm can be changed by an industrial robot to provide more accuracy in the movement of the robotic arm. Especially the second version of the Kinect Camera can be used because it provides almost 10 percent more camera vision which means a brighter view in detecting. It is able to track as well 26 joints instead of 20 joints. The second version of the Kinect Camera can detect the hand movement such in an open and closed hand position. In addition it can be used in order to improve the detection of the joint points of the user.

REFERENCES

[1] M. A. Hussein, S. A. Ahmed, F. A. Elmisery, and R. Mostafa, “Motion Control of Robot by using Kinect Sensor,” *Res. J. Appl. Sci. Eng. Technol.*, vol. 8, no. 11, pp. 1384–1388, 2014.

[2] G. Broccia, M. Livesu, and R. Scateni, “Gestural interaction for robot motion control,” *Eurographics Ital. Chapter Conf. 2011*, pp. 61–66, 2011.

[3] R. K. Megalingam and D. Menon, “Implementation of Gesture Control in Robotic Arm using Kinect Module,” no. Icadme, pp. 30–31, 2015.

[4] W. Song, X. Guo, F. Jiang, S. Yang, G. Jiang, and Y. Shi, “Teleoperation humanoid robot control system based on kinect sensor,” *Proc. 2012 4th Int. Conf. Intell. Human-Machine Syst. Cybern. IHMSC 2012*, vol. 2, pp. 264–267, 2012.

[5] Y.-H. Seo and J.-S. Park, “Motion Control System for a Humanoid Robot Based on Motion Capture using Kinect,” *Iaser*, pp. 2–3, 2011.

[6] M. Z. Al-faiz and A. F. Shanta, “Kinect-Based Humanoid Robotic Manipulator for Human Upper Limbs Movements Tracking,” no. February, pp. 29–37, 2015.

[7] A. A. Mohammed and M. Sunar, “Kinematics modeling of a 4-DOF robotic arm,” *Proc. - 2015 Int. Conf. Control. Autom. Robot. ICCAR 2015*, no. May 2015, pp. 87–91, 2015.

[8] Spong, “Robot dynamics and control,” *Automatica*, vol. 28, no. 3, pp. 655–656, 1992.

[9] C. Közkurt and M. Soyaslan, “Euler Bilekli Scara Robot Kolu Ġçin Kinematik Analiz Yazılımı GeliĢtirilmesi,” no. May, pp. 16–18, 2011.

[10] T. Asfour and R. Dillmann, “Human-like Motion of a Humanoid Robot Arm Based on a Closed-Form Solution of the Inverse Kinematics Problem,” *Conf. Intell. Robot. Syst.*, no. October, 2003.

[11] R. Miles, *Start Here! Learn the Kinect API*. 2012.

[12] Z. Zhang, “Microsoft kinect sensor and its effect,” *IEEE Multimedia*, vol. 19, no. 2. pp. 4–10, Feb-2012.

[13] D. Catuhe, *Programming with the Kinect for Windows SDK*. 2012.

[14] C. Franciscone, “EEZYbotARM MK2,” 2016. [Online]. Available: http://www.eezyrobots.it/eba_mk2.html. [Accessed: 30-Aug-2017].

[15] “Dimension® BST/SST 1200es 3D Printer User Guide,” 2008.

[16] F. Reed, “How servo motors work?,” *Jameco*, 2014. [Online]. Available: http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html. [Accessed: 13-Sep-2017].

[17] Simon Monk, “Arduino Lesson 14. Servo Motors,” *Adafruit Learn. Syst.*, p. 14, 2013.

[18] E. R. Melgar, C. C. Díez, and P. Jaworski, *Arduino and Kinect Projects: Design, Build, Blow their Minds (Technology in Action)*. 2012.

[19] T. E. Kurt, “ServoEaser library for Arduino,” 2011. [Online]. Available: https://github.com/todbot/ServoEaser.

[20] Microsoft, “Skeletal Tracking,” *Kinect for Windows SDK Documentation*, 2016. [Online]. Available: https://msdn.microsoft.com/en-us/library/jj131025.aspx. [Accessed: 11-Oct-2017].

[21] Microsoft, “Combining skeleton and color stream,” 2013. [Online]. Available: https://social.msdn.microsoft.com/Forums/en-US/9905ce20-439e-44fe-891f-99bcdaee0c84/need-help-with-combining-and-overlaying-the-skeleton-and-color-streamimage?forum=kinectsdk.

[22] P. D. Ignatiuz, “Kinect status and setup the Kinect for interaction,” 2013. [Online]. Available: https://www.codeproject.com/Tips/701338/Kinect-status-and-setup-the-Kinect-for-interaction.

[23] J. Martinez, “Mean, Median, Mode & Standard Deviation,” 2015. [Online]. Available: http://www.compton.edu/facultystaff/jmmartinez/docs/Math-150-Spring-2015/Stat-Ch3-Formulas.pdf.

APPENDICES

APPENDIX A

Printed parts of the robot

1. Part1_base
2. Part2_MainArm
3. Part3_Varm
4. Part4_link135
5. Part5_link135Angle
6. Part6_horarm
7. Part7_trialink
8. Part8_link147
9. Part9_trialinkfront
10. Part10_gearservo
11. Part11_gearmast
12. Part12_mainbase
13. Part13_lowerbase
14. Part14_clawbase
15. Part15_clawfinger_dx
16. Part16_clawgeardrive
17. Part17_clawfinger_sx
18. Part18_clawgeardriven
19. Part19_drivecover

Non-printed parts of the robot

1. Three MG995 servo motors
2. One MG90s servo motor
3. 30 spheres at 6.3 mm diameter
4. One 4x16 bearing
5. Some amount M3 screw
6. Some amount M4 screw
7. Two M4 threated rod
8. Some amount M4 nuts

APPENDIX B (Arduino)

```
#include <Servo.h>
#include "ServoEaser.h"

String readString;
String angle1; // angles
String angle2;
String angle3;
String angle4;

int servoFrameMillis = 100;  // minimum time between servo updates

Servo myservo1, myservo2, myservo3, myservo4; // servo motors
ServoEaser servoEaser1, servoEaser2, servoEaser3, servoEaser4;

int ind1, ind2, ind3, ind4; // , locations
int oldn1, oldn2,oldn3, oldn4;
int n1, n2, n3, n4;

//
void setup()
{
  Serial.begin(19200); //baud rate value

  myservo1.attach(9); //Elbow
  oldn1=87;
  myservo1.write(oldn1);

  myservo2.attach(10); //ShoulderUp //Rotation
  oldn2=90;
  myservo2.write(oldn2);

  myservo3.attach(11); //ShoulderDown
  oldn3=127;
  myservo3.write(oldn3);

  myservo4.attach(12); //HandStatus //Grip
  oldn4=80;
  myservo4.write(oldn3);


  // begin with a framerate, no starting position, we don't know
  servoEaser1.begin( myservo1, servoFrameMillis ); //Elbow
  servoEaser2.begin( myservo2, servoFrameMillis );  //ShoulderUp
  servoEaser3.begin( myservo3, servoFrameMillis ); //ShoulderDown
  servoEaser4.begin( myservo4, servoFrameMillis ); //Hand
```

```
}

//
void loop()
{
  servoEaser1.update();
  servoEaser2.update();
  servoEaser3.update();
  servoEaser4.update();


 if (Serial.available())  {
    char c = Serial.read();  //gets one byte from serial buffer
    if (c == '*') {
      //do stuff


      ind1 = readString.indexOf('/');  //finds location of first ,
      angle1 = readString.substring(0, ind1);   //captures first data String
      n1 = angle1.toInt();
      n1 = map(n1, 50, 170, 20, 154); //Elbow

      ind2 = readString.indexOf('/', ind1+1);   //finds location of second ,
      angle2 = readString.substring(ind1+1, ind2+1);   //captures second data String
      n2 = angle2.toInt();
      n2 = map(n2, 120, 156, 15, 165); //ShoulderUp

      ind3 = readString.indexOf('/', ind2+1 );
      angle3 = readString.substring(ind2+1, ind3+1);
      n3 = angle3.toInt();
      n3 = map(n3, 40, 126, 95, 160); //ShoulderDown

      ind4 = readString.indexOf('/', ind3+1 );
      angle4 = readString.substring(ind3+1);
      n4 = angle4.toInt(); //HandStatus

      Servowrite(n1,n2,n3,n4);
    Serial.print("Last pos. of servos: ");
    Serial.print(myservo1.read());
    Serial.print(myservo2.read());
    Serial.print(myservo3.read());
    Serial.println(myservo4.read());

      readString=""; //clears variable for new input
      angle1="";
      angle2="";
      angle3="";
      angle4="";
    }
    else {
      readString += c; //makes the string readString
    }
 }
 oldn1=n1;
 oldn2=n2;
 oldn3=n3;
 oldn4=n4;
}

void Servowrite(int n1,int n2, int n3, int n4){
  if(-7<n1-oldn1<7){
  //if(n1 != oldn1){
    servoEaser1.easeTo( n1, 1800);
  }
  if(-7<n2-oldn2<7){
  //if(n2 != oldn2){
    servoEaser2.easeTo( n2, 1800);
  }
  if(-7<n3-oldn3<7){
  //if(n3 != oldn3){
    servoEaser3.easeTo( n3, 1800);
  }

  if(n4 != oldn4){
    servoEaser4.easeTo( n4, 1800);
  }
     }
```

APPENDIX C (Visual Studio)

```
using Microsoft.Kinect; //kinect libraries
using System.IO;
using System.IO.Ports; //serialport libraries
using Microsoft.Kinect.Toolkit; // for kinect status kinectsensorchooser

namespace Kinect_arduino_interaction
{
    /// <summary>
    /// Interaction logic for MainWindow.xaml
    /// </summary>
    public partial class MainWindow : Window
    {
        SerialPort sp = new SerialPort(); //FOR SERIALPORT

        private KinectSensor kinectDevice;
        private readonly Brush[] skeletonBrushes;

        private WriteableBitmap depthImageBitMap;
        private Int32Rect depthImageBitmapRect;
        private Int32 depthImageStride;
        //private DepthImageFrame lastDepthFrame;

        private WriteableBitmap colorImageBitmap;
        private Int32Rect colorImageBitmapRect;
        private int colorImageStride;
        //private byte[] colorImagePixelData;

        private Skeleton[] frameSkeletons;
        public string str_angle_1;
        public string str_angle_2;
        public string str_angle_3;
        public string handStatus = "80";

        System.Windows.Threading.DispatcherTimer dispatcherTimer = new
System.Windows.Threading.DispatcherTimer();

        private KinectSensorChooser sensorChooser;


        public MainWindow()
        {
            InitializeComponent();

            // initialize the sensor chooser and UI
            this.sensorChooser = new KinectSensorChooser();
            //this.sensorChooser.KinectChanged += SensorChooserOnKinectChanged;
            this.sensorChooserUi.KinectSensorChooser = this.sensorChooser;
            this.sensorChooser.Start();

            skeletonBrushes = new Brush[] { Brushes.DarkTurquoise };

            KinectSensor.KinectSensors.StatusChanged += KinectSensors_StatusChanged;
            this.KinectDevice = KinectSensor.KinectSensors.FirstOrDefault(x => x.Status
== KinectStatus.Connected);
            dispatcherTimer.Tick += new EventHandler(dispatcherTimer_Tick);
            dispatcherTimer.Interval = TimeSpan.FromSeconds(1);
            dispatcherTimer.Start();

            KinectRegion.AddHandPointerGripHandler(this.button, this.button_Click);
            KinectRegion.AddHandPointerGripReleaseHandler(this.button, this.buttonnot);
            var regionSensorBinding = new Binding("Kinect") { Source = this.sensorChooser
};
            BindingOperations.SetBinding(this.kinectRegion,
KinectRegion.KinectSensorProperty, regionSensorBinding);

        }

        //Timer
        private void dispatcherTimer_Tick(object sender, EventArgs e)
        {
            // code goes here
            if (sp.IsOpen & str_angle_1 != null)
            {
                string stringtotal = str_angle_1 + "/" + str_angle_2 + "/" + str_angle_3 + "/" +
handStatus + "*";
                sp.Write(stringtotal);
                if (sp.BytesToRead > 0)
                {
                    LabelOutput.Content = "Sending angles :" + stringtotal +
Environment.NewLine + sp.ReadExisting() + Environment.NewLine +
LabelOutput.Content;
                }

            }

        }

        public KinectSensor KinectDevice
        {
            get { return this.kinectDevice; }
            set
            {
                if (this.kinectDevice != value)
                {
                    //Uninitialize
                    if (this.kinectDevice != null)
                    {
                        this.kinectDevice.Stop();
                        this.kinectDevice.SkeletonFrameReady                        -=
kinectDevice_SkeletonFrameReady;
                        this.kinectDevice.ColorFrameReady -= kinectDevice_ColorFrameReady;
                        this.kinectDevice.DepthFrameReady                           -=
kinectDevice_DepthFrameReady;
                        this.kinectDevice.SkeletonStream.Disable();
                        this.kinectDevice.DepthStream.Disable();
                        this.kinectDevice.ColorStream.Disable();
                        this.frameSkeletons = null;
                    }

                    this.kinectDevice = value;

                    //Initialize
                    if (this.kinectDevice != null)
                    {
                        if (this.kinectDevice.Status == KinectStatus.Connected)
                        {
                            this.kinectDevice.SkeletonStream.Enable();

this.kinectDevice.ColorStream.Enable(ColorImageFormat.RgbResolution640x480Fps3
0);

this.kinectDevice.DepthStream.Enable(DepthImageFormat.Resolution640x480Fps30);
                            this.frameSkeletons            =             new
Skeleton[this.kinectDevice.SkeletonStream.FrameSkeletonArrayLength];
                            this.kinectDevice.SkeletonFrameReady                  +=
kinectDevice_SkeletonFrameReady;
                            this.kinectDevice.ColorFrameReady                     +=
kinectDevice_ColorFrameReady;
                            this.kinectDevice.DepthFrameReady                     +=
kinectDevice_DepthFrameReady;
                            this.kinectDevice.Start();

                            DepthImageStream depthStream = kinectDevice.DepthStream;
                            depthStream.Enable();

                            ColorImageStream colorStream = kinectDevice.ColorStream;
                            colorStream.Enable();

                            DepthImage.Source = this.depthImageBitMap;
                        }
                    }
                }
```

```
        }
    }

    void                    kinectDevice_DepthFrameReady(object                    sender,
DepthImageFrameReadyEventArgs e)
    {
        using (DepthImageFrame depthFrame = e.OpenDepthImageFrame())
        {
            if (depthFrame != null)
            {
                short[] depthPixelDate = new short[depthFrame.PixelDataLength];
                depthFrame.CopyPixelDataTo(depthPixelDate);
                depthImageBitMap.WritePixels(depthImageBitmapRect,    depthPixelDate,
depthImageStride, 0);
            }
        }
    }

    void                    kinectDevice_ColorFrameReady(object                    sender,
ColorImageFrameReadyEventArgs e)
    {
        using (ColorImageFrame frame = e.OpenColorImageFrame())
        {
            if (frame != null)
            {
                byte[] pixelData = new byte[frame.PixelDataLength];
                frame.CopyPixelDataTo(pixelData);
                this.colorImageBitmap.WritePixels(this.colorImageBitmapRect,  pixelData,
this.colorImageStride, 0);
            }
        }
    }

    void                    kinectDevice_SkeletonFrameReady(object                    sender,
SkeletonFrameReadyEventArgs e)
    {
        using (SkeletonFrame frame = e.OpenSkeletonFrame())
        {

            if (frame != null)
            {
                Polyline figure;
                Brush userBrush;
                Skeleton skeleton;


                LayoutRoot.Children.Clear();
                frame.CopySkeletonDataTo(this.frameSkeletons);
            for (int i = 0; i < this.frameSkeletons.Length; i++)
            {
                skeleton = this.frameSkeletons[i];

                if (skeleton.TrackingState == SkeletonTrackingState.Tracked &&
kinectDevice.SkeletonStream.TrackingMode == SkeletonTrackingMode.Default)
                {
                    userBrush = this.skeletonBrushes[i % this.skeletonBrushes.Length];

                    //draw head and shoulders
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.Head,
JointType.ShoulderCenter, JointType.ShoulderLeft, //JointType.Spine,
                                                        //  JointType.ShoulderRight,
JointType.ShoulderCenter, JointType.HipCenter
                                                        });

                    LayoutRoot.Children.Add(figure);

                    figure = CreateFigure(skeleton, userBrush, new[] {
JointType.ShoulderCenter, JointType.ShoulderRight, });
                    LayoutRoot.Children.Add(figure);

                    //draw-body
                    figure = CreateFigure(skeleton, userBrush, new[] {
JointType.ShoulderCenter, JointType.Spine, JointType.HipCenter, });
                    LayoutRoot.Children.Add(figure);

                    //draw-hip line
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.HipLeft,
JointType.HipRight });
                    LayoutRoot.Children.Add(figure);

                    //draw-left leg
                    figure = CreateFigure(skeleton, userBrush, new[] {
JointType.HipCenter, JointType.HipLeft, JointType.KneeLeft, JointType.AnkleLeft,
JointType.FootLeft });
                    LayoutRoot.Children.Add(figure);

                    //draw-right leg
                    figure = CreateFigure(skeleton, userBrush, new[] {
JointType.HipCenter, JointType.HipRight, JointType.KneeRight,
JointType.AnkleRight, JointType.FootRight });
                    LayoutRoot.Children.Add(figure);

                    //draw-left arm
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.ShoulderLeft, JointType.ElbowLeft, JointType.WristLeft, JointType.HandLeft });
                    LayoutRoot.Children.Add(figure);

                    //draw-right arm
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.ShoulderRight, JointType.ElbowRight, JointType.WristRight, JointType.HandRight });
                    LayoutRoot.Children.Add(figure);

                }
                else if (kinectDevice.SkeletonStream.TrackingMode == SkeletonTrackingMode.Seated)
                {
                    userBrush = this.skeletonBrushes[i % this.skeletonBrushes.Length];

                    //draw-head and shoulders
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.Head, JointType.ShoulderCenter, JointType.ShoulderLeft, });
                    LayoutRoot.Children.Add(figure);

                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.ShoulderCenter, JointType.ShoulderRight, });
                    LayoutRoot.Children.Add(figure);
                    //draw-left arm
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.ShoulderLeft, JointType.ElbowLeft, JointType.WristLeft, JointType.HandLeft });
                    LayoutRoot.Children.Add(figure);

                    //draw-right arm
                    figure = CreateFigure(skeleton, userBrush, new[] { JointType.ShoulderRight, JointType.ElbowRight, JointType.WristRight, JointType.HandRight });
                    LayoutRoot.Children.Add(figure);


                    /* // get the joint
                     Joint rightHand = skeleton.Joints[JointType.HandRight];

                     // get the individual points of the right hand
                     rightX = rightHand.Position.X;
                     rightY = rightHand.Position.Y;
                     rightZ = rightHand.Position.Z;
                     labelrightHandX.Content = rightX;*/
                }
            }
```

```
                if (frame != null)
                {
                    if (this.frameSkeletons == null)
                    {
                        // Allocate array of skeletons
                        this.frameSkeletons = new Skeleton[frame.SkeletonArrayLength];
                    }

                    // Copy skeletons from this frame
                    frame.CopySkeletonDataTo(this.frameSkeletons);

                    // Find first tracked skeleton, if any
                    skeleton = this.frameSkeletons.Where(s => s.TrackingState ==
SkeletonTrackingState.Tracked).FirstOrDefault();

                    if (skeleton != null)
                    {
                        // Obtain the right Wrist joint; if tracked, print its position
                        Joint WristRight = skeleton.Joints[JointType.WristRight];
                        Joint ElbowRight = skeleton.Joints[JointType.ElbowRight];
                        Joint ShoulderRight = skeleton.Joints[JointType.ShoulderRight];
                        Joint ShoulderCenter = skeleton.Joints[JointType.ShoulderCenter];
                        Joint HipLeft = skeleton.Joints[JointType.HipLeft];

                        double[] vector1 = new double[] { WristRight.Position.X -
ElbowRight.Position.X, WristRight.Position.Y - ElbowRight.Position.Y,
WristRight.Position.Z - ElbowRight.Position.Z };
                        double[] vector2 = new double[] { ShoulderRight.Position.X -
ElbowRight.Position.X, ShoulderRight.Position.Y - ElbowRight.Position.Y,
ShoulderRight.Position.Z - ElbowRight.Position.Z };
                        double[] vector3 = new double[] { ShoulderRight.Position.X -
ShoulderCenter.Position.X, ShoulderRight.Position.Y - ShoulderCenter.Position.Y,
ShoulderRight.Position.Z - ShoulderCenter.Position.Z };
                        double[] vector4 = new double[] { ShoulderRight.Position.X -
HipLeft.Position.X, ShoulderRight.Position.Y - HipLeft.Position.Y,
ShoulderRight.Position.Z - HipLeft.Position.Z };
                        double angle_calc_1 = Angle(vector1, vector2);
                        double angle_calc_2 = Angle(vector2, vector3);
                        double angle_calc_3 = Angle(vector2, vector4);

                        //labelAngle1.Content = "Angle = " + angle_calc.ToString("#.##");

                        labelAngle1.Content = "Elbow = " + angle_calc_1.ToString("#.#") +
"°";
                        labelAngle2.Content = "ShoulderUp = " +
angle_calc_2.ToString("#.#") + "°";
                        labelAngle3.Content = "ShoulderDown = " +
angle_calc_3.ToString("#.#") + "°";
```

```
                    str_angle_1 = angle_calc_1.ToString("#");
                    str_angle_2 = angle_calc_2.ToString("#");
                    str_angle_3 = angle_calc_3.ToString("#");



                    //sp.Write(angle_calc.ToString("#"));

                    // WristRight coordinates
                    if (WristRight.TrackingState == JointTrackingState.Tracked)
                    {
                        labelWristRight.Content = "WristRight: " + "x=" +
WristRight.Position.X + ", " + "y=" + WristRight.Position.Y + ", " + "z=" +
WristRight.Position.Z;

                    }
                    else
                    {
                        labelWristRight.Content = "WristRight: " + ", " + "x=Not_Tracked"
+ ", " + "y=Not_Tracked" + ", " + "z=Not_Tracked";
                    }
                    // ElbowRight coordinates
                    if (ElbowRight.TrackingState == JointTrackingState.Tracked)
                    {
                        labelElbowRight.Content = "ElbowRight: " + "x=" +
ElbowRight.Position.X + ", " + "y=" + ElbowRight.Position.Y + ", " + "z=" +
ElbowRight.Position.Z;

                    }
                    else
                    {
                        labelElbowRight.Content = "ElbowRight: " + ", " +
"x=Not_Tracked" + ", " + "y=Not_Tracked" + ", " + "z=Not_Tracked";
                    }
                    // ShoulderRight coordinates
                    if (ShoulderRight.TrackingState == JointTrackingState.Tracked)
                    {
                        labelShoulderRight.Content = "ShoulderRight: " + "x=" +
ShoulderRight.Position.X + ", " + "y=" + ShoulderRight.Position.Y + ", " + "z=" +
ShoulderRight.Position.Z;

                    }
                    else
                    {
                        labelShoulderRight.Content = "ShoulderRight: " + ", " +
"x=Not_Tracked" + ", " + "y=Not_Tracked" + ", " + "z=Not_Tracked";
                    }
                }
                }
            }
        }
    }

    private Polyline CreateFigure(Skeleton skeleton, Brush brush, JointType[] joints)
    {
        Polyline figure = new Polyline();

        figure.StrokeThickness = 6;
        figure.Stroke = brush;

        /* for (int i = 0; i < joints.Length; i++)
         {
             figure.Points.Add(GetJointPoint(skeleton.Joints[joints[i]]));
         }
         */
        for (int i = 0; i < joints.Length; i++)
        {
            figure.Points.Add(GetJointPoint2(skeleton.Joints[joints[i]]));
        }

        return figure;
    }

    /*private Point GetJointPoint(Joint joint)
    {
        CoordinateMapper cm = new CoordinateMapper(kinectDevice);

        DepthImagePoint point = cm.MapSkeletonPointToDepthPoint(joint.Position,
this.KinectDevice.DepthStream.Format);
        //ColorImagePoint point2 = cm.MapSkeletonPointToColorPoint(joint.Position,
this.KinectDevice.ColorStream.Format);
        point.X *= (int)this.LayoutRoot.ActualWidth /
KinectDevice.DepthStream.FrameWidth;
        point.Y *= (int)this.LayoutRoot.ActualHeight /
KinectDevice.DepthStream.FrameHeight;

        return new Point(point.X, point.Y);
    }*/

    private Point GetJointPoint2(Joint joint)
    {
        CoordinateMapper cm = new CoordinateMapper(kinectDevice);

        //DepthImagePoint point = cm.MapSkeletonPointToDepthPoint(joint.Position,
this.KinectDevice.DepthStream.Format);
        ColorImagePoint point = cm.MapSkeletonPointToColorPoint(joint.Position,
this.KinectDevice.ColorStream.Format);
        point.X                    *=                   (int)this.LayoutRoot.ActualWidth                    /
KinectDevice.ColorStream.FrameWidth;
        point.Y                    *=                   (int)this.LayoutRoot.ActualHeight                   /
KinectDevice.ColorStream.FrameHeight;

        return new Point(point.X, point.Y);
    }


    private                    void                    WindowClosing(object                    sender,
System.ComponentModel.CancelEventArgs e)
    {
        this.sensorChooser.Stop();
    }

    private void Depth_Image_Checked(object sender, RoutedEventArgs e)
    {
        DepthImage.Visibility = Visibility;
    }

    private void Depth_Image_Unchecked(object sender, RoutedEventArgs e)
    {
        DepthImage.Visibility = Visibility.Hidden;
    }

    private void CheckBoxSeatedModeChanged(object sender, RoutedEventArgs e)
    {
        if (null != this.kinectDevice)
        {
            if (this.checkBoxSeatedMode.IsChecked.GetValueOrDefault())
            {
                this.kinectDevice.SkeletonStream.TrackingMode                              =
SkeletonTrackingMode.Seated;
            }
            else
            {
                this.kinectDevice.SkeletonStream.TrackingMode                              =
SkeletonTrackingMode.Default;
            }
        }
    }
    //
    private static double Angle(double[] vec1, double[] vec2)
    {
        if (vec1 == null)
            return 0;
        if (vec2 == null)
```

```
                return 0;
            if (vec1.Length != vec2.Length)
                return 0;

            double vec1Lenght = 0;
            double vec2Lenght = 0;
            double dot = 0;

            for (int x = 0; x < vec1.Length; x++)
            {
                dot += vec1[x] * vec2[x];
                vec1Lenght += vec1[x] * vec1[x];
                vec2Lenght += vec2[x] * vec2[x];
            }

            double vecLenght = Math.Sqrt(vec1Lenght) * Math.Sqrt(vec2Lenght);
            double angle = Math.Acos(dot / vecLenght) * 180 / Math.PI;

            return angle;
        }

        /*////////////////////// THE BELOW CODE IS FOR SERIAL COMMUNICATION
//////////////////////*/

        private void degree_nine_Click(object sender, RoutedEventArgs e)
        {
            //sp.Write(str_angle);
            sp.Write("110/138/83/80*");
        }
        private void degree_three_Click(object sender, RoutedEventArgs e)
        {
            sp.Write("50/120/40/80*");
        }
        private void degree_twelve_Click(object sender, RoutedEventArgs e)
        {
            sp.Write("170/156/126/80*");
        }

        private void Connect_Click(object sender, RoutedEventArgs e)
        {

            try
            {
                String portName = comportno.Text;
                sp.PortName = portName;
                sp.BaudRate = 19200;
                sp.Open();
                status.Text = "Connected";
                TextBox1.Clear();
            }
            catch (Exception)
            {

                MessageBox.Show("Please give a valid port number or check your
connection");
            }
        }

        private void Disconnect_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                sp.Write("110/138/83/80*");
                /* if (sp.BytesToRead > 0)
                 {
                    System.Threading.Thread.Sleep(1000);
                    LabelOutput.Content = "Sending angles: 110/140/80*" +
Environment.NewLine + sp.ReadExisting();
                 }*/
                sp.Close();
                status.Text = "Disconnected";
            }
            catch (Exception)
            {

                MessageBox.Show("First Connect and then disconnect");
            }
        }

        private void TextBox1_TextChanged(object sender, TextChangedEventArgs e)
        {

        }
        private void OnKeyDownHandler(object sender, KeyEventArgs e)
        {
            if (e.Key == Key.Return)
            {
                sp.Write(TextBox1.Text);
                LabelOutput.Content = TextBox1.Text + Environment.NewLine +
LabelOutput.Content;
                TextBox1.Clear();
            }
        }

        /* private string Send(string data,int waitForReplayMS=2000)
```

```
         {

            try
            {
               sp.Write(data);
               System.Threading.Thread.Sleep(waitForReplayMS);

            }
            catch (Exception ex)
            {
               throw ex;
            }
         }*/
        ///////////////////////////////////*SERIAL COMMUNICATION*/////////////////////////
        /////////////////////////////The Below Code for Using Hand Data/////////////////////////////

        private void button_Click(object sender, RoutedEventArgs e)
        {
           labelAngle4.Content = "Hand = Closed";
           handStatus = "10";
        }
        private void buttonnot(object sender, RoutedEventArgs e)
        {
           labelAngle4.Content = "Hand = Open";
           handStatus = "80";

        }
    }
}

///////////////////////////////////////inverse kinematics///////////////////////////////
private static double inverse_kinematic1(double x, double y)
        {
           //double t = y / x ;
           double radians = Math.Atan2(y,x);
           double angle = radians * (180 / Math.PI);
           return angle;
        }

        private static double inverse_kinematic2(double x, double y, double z)
        {
           double a2 = 0.147 * 6;
           double a3 = 0.240 * 6;
           double s3, c3;
           c3 = ((a2 * a2) + (a3 * a3) - ((x * x) + (y * y) + (z * z)) ) / (2 * a2 * a3);
           s3 = Math.Sqrt(1 - (c3 * c3));
           double radians = Math.Atan2( z, Math.Sqrt((x * x) + (y * y))) - Math.Atan2((a3 *
s3), (a2 + (a3 * c3)));
```

```
    double angle = radians * (180 / Math.PI) + 90;
    return angle;
}

private static double inverse_kinematic3(double x, double y, double z)
{
    double a2 = 0.147 * 6;
    double a3 = 0.240 * 6;
    double s3, c3;
    c3 = ((a2 * a2) + (a3 * a3) - ((x * x) + (y * y) + (z * z))) / (2 * a2 * a3);
    s3 = Math.Sqrt(1 - (c3 * c3));
    double radians = Math.Atan2(s3, c3);
    double angle =180 - radians * (180 / Math.PI);
    return angle;
}
```

APPENDIX D (XAML)

```
<Window x:Class="Kinect_arduino_interaction.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:k="http://schemas.microsoft.com/kinect/2013"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Kinect_arduino_interaction"
        mc:Ignorable="d"
        Title="MainWindow" Height="734" Width="1160">
    <Grid Margin="0,0,2,0">

        <Grid Height="480" Width="640"  Margin="26,71,2.303,0">
            <k:KinectRegion x:Name="kinectRegion" Margin="-24,-70,-489,-150" >
                <Grid>
                    <Button x:Name="button" Click="button_Click" 
HorizontalAlignment="Center" VerticalAlignment="Center" Width="1151" 
Height="703" HorizontalContentAlignment="Left" BorderBrush="{x:Null}" 
Foreground="{x:Null}" Margin="2,0,0,-3">
                    </Button>
                </Grid>
            </k:KinectRegion>
            <Image x:Name="ColorImage" VerticalAlignment="Top" 
HorizontalAlignment="Left" />
            <Image x:Name="DepthImage" Visibility="Hidden" VerticalAlignment="Top" 
HorizontalAlignment="Left"/>
            <Grid Height="480" Width="640" x:Name="LayoutRoot" 
Background="Transparent"/>
        </Grid>
        <Grid HorizontalAlignment="Left" Height="48" Margin="26,4,0,-8" 
VerticalAlignment="Top" Width="640" Grid.Row="2" Grid.RowSpan="2">

            <CheckBox x:Name="Depth_Image" Content="Depth Image" Height="28" 
VerticalAlignment="Center" HorizontalAlignment="Left" Width="145" 
Checked="Depth_Image_Checked" RenderTransformOrigin="0.524,1.095" 
FontFamily="Tahoma" FontSize="14" Cursor="Hand" BorderThickness="2" 
Unchecked="Depth_Image_Unchecked" Margin="10,9,0,10" Grid.Row="1"/>
            <CheckBox Content="Seated Mode" Height="28" VerticalAlignment="Center" 
Margin="207,9,325,10" x:Name="checkBoxSeatedMode" 
Checked="CheckBoxSeatedModeChanged" 
Unchecked="CheckBoxSeatedModeChanged" FontFamily="Tahoma" FontSize="14" 
Cursor="Hand" BorderThickness="2" RenderTransformOrigin="0.492,1.206" 
Grid.Row="1"/>
        </Grid>
        <Grid Margin="3,71,0,76" Grid.Column="1" Grid.RowSpan="2" >
            <Button x:Name="degree_nine" Content="Middle" HorizontalAlignment="Left" 
Margin="135,530,0,-86" VerticalAlignment="Top" Width="112" Height="36"
```

```
Click="degree_nine_Click"/>
        <TextBlock x:Name="status" HorizontalAlignment="Left" Margin="373,84,0,0"
TextWrapping="Wrap" Text="Disconnected" VerticalAlignment="Top" Height="29"
Width="94"/>
        <TextBox x:Name="comportno" HorizontalAlignment="Left" Height="23"
Margin="341,10,0,0" TextWrapping="Wrap" Text="COM3" VerticalAlignment="Top"
Width="126"/>
        <Button x:Name="Connect" Content="Connect" HorizontalAlignment="Left"
Margin="341,38,0,0" VerticalAlignment="Top" Width="60" Height="40"
Click="Connect_Click"/>
        <Button x:Name="Disconnect" Content="Disconnect"
HorizontalAlignment="Left" Margin="406,38,0,0" VerticalAlignment="Top"
Width="70" Height="41" Click="Disconnect_Click"/>
        <TextBox x:Name="TextBox1" HorizontalAlignment="Left" Height="23"
            Margin="10,10,0,0" TextWrapping="Wrap" Text="Write the position of servo
motor here and Press ENTER" VerticalAlignment="Top" Width="315"
            TextChanged="TextBox1_TextChanged"
KeyDown="OnKeyDownHandler"/>
        <Button x:Name="degree_three" Content="Down Limit"
HorizontalAlignment="Left" Margin="12,530,0,-86" VerticalAlignment="Top"
Width="118" Height="36" Click="degree_three_Click"/>
        <Button x:Name="degree_twelve" Content="Up Limit"
HorizontalAlignment="Left" Margin="252,530,0,-86" VerticalAlignment="Top"
Width="107" Height="36" Click="degree_twelve_Click"/>
        <Label Name="LabelOutput" Content="" HorizontalAlignment="Left"
Height="257" Margin="12,38,0,0" VerticalAlignment="Top" Width="315"/>
      </Grid>
      <Label x:Name="labelWristRight" Content="WristRight: x=***,y=***,z=***"
HorizontalAlignment="Left" Margin="26,10,0,0" VerticalAlignment="Top"
Height="22" FontFamily="Arial" FontWeight="Bold" Width="400" Grid.Row="1"/>
      <Label x:Name="labelElbowRight" Content="ElbowRight: x=***,y=***,z=***"
HorizontalAlignment="Left" Margin="26,40,0,0" VerticalAlignment="Top"
Height="22" FontFamily="Arial" FontWeight="Bold" Width="400" Grid.Row="1"/>
      <Label x:Name="labelShoulderRight" Content="ShoulderRight:
x=***,y=***,z=***" HorizontalAlignment="Left" Margin="26,70,0,0"
VerticalAlignment="Top" Height="22" FontFamily="Arial" FontWeight="Bold"
Width="400" Grid.Row="1"/>
      <Label x:Name="labelAngle1" Content="Elbow = ***"
HorizontalAlignment="Left" Margin="2.697,381,0,0" VerticalAlignment="Top"
Width="314" Height="52" FontSize="36" FontFamily="Arial" Grid.Column="1"/>
      <Label x:Name="labelAngle2" Content="ShoulderUp = ***"
HorizontalAlignment="Left" Margin="2.697,438,0,0" VerticalAlignment="Top"
Width="332" Height="46" FontSize="36" FontFamily="Arial" Grid.Column="1"/>
      <Label x:Name="labelAngle3" Content="ShoulderDown = ***"
HorizontalAlignment="Left" Margin="2.697,489,0,0" VerticalAlignment="Top"
Width="398" Height="52" FontSize="36" FontFamily="Arial" Grid.Column="1"/>
      <Label x:Name="labelAngle4" Content="Hand = ***"
HorizontalAlignment="Left" Margin="3,539,0,0" VerticalAlignment="Top"
```

```
Width="398" Height="52" FontSize="36" FontFamily="Arial" Grid.RowSpan="2"
Grid.Column="1"/>
        <k:KinectSensorChooserUI HorizontalAlignment="Center"
VerticalAlignment="Top" Name="sensorChooserUi" Margin="553,0,75.303,0"/>
        <k:KinectUserViewer k:KinectRegion.KinectRegion="{Binding ElementName =
kinectRegion}"
                PrimaryUserColor="Violet" UserColoringMode="HighlightPrimary"
HorizontalAlignment="Left"
                VerticalAlignment="Bottom" Height="66" Width="100"
Margin="291,0,0,485"/>

    </Grid>
</Window>
```

RESUME

Name Surname	**:** Yusuf Sarıkaya
Date / Place of Birth	**:** 18.05.1990 / Istanbul
Email	**:** yusufsrk@gmail.com

Educational Background

High School	**:** Fenerbahçe High School, Istanbul, 2004/2008
Bachelor Degree	**:** Hacettepe University, Physics Engineering, Ankara, 2008/2014
Master Degree	**:** Marmara University, Mechatronics Engineering, 2015/…