TALLINNA TEHNIKAÜLIKOOL
TALLINN UNIVERSITY OF TECHNOLOGY

Mechatronics

COBOT TYPE ROBOT LEARNING SYSTEM

COBOT TÜÜPI ROBOTI ÕPPIMISSÜSTEEM

MASTER THESIS

Student: Marasinghe Mudiyanselage Danushka Ranjana Marasinghe

Student code: 177303MAHM

Supervisor: Professor Mart Tamre, Programme Director (Mechatronics)

Tallinn, 2019

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." ..... 201.....

Author: .....
/signature /

Thesis is in accordance with terms and requirements

"....." ..... 201....

Supervisor: .....
/signature/

Accepted for defence

"....."..... 201... .

Chairman of theses defence commission: .....
/name and signature/

Department of Electrical Power Engineering and Mechatronics

THESIS TASK

Student: Marasinghe Mudiyanselage Danushka Ranjana Marasinghe 177303

Study programme, MAHM02/13 Mechatronics

main specialty: Mechatronics

Supervisor(s): Mart Tamre, Professor, 620 3202

Consultants

Thesis topic:

(in English) *Cobot Type Robot Learning System*

(in Estonian) *Cobot Tüüpi Roboti Õppimissüsteem*

Thesis main objectives:

1. Perform background research on existing methods and determine the suitable approach
2. Design the teaching and replaying system using the determined software and hardware
3. Evaluate the functionality of the system

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Perform background research on previous work	25/02/2019
2.	Formulate a suitable methodology	15/03/2019
3.	Identify the required hardware and software	27/03/2019
4	Design the system	10/05/2019
5	Write the report and conclusion of the thesis	21/05/2019

Language: English **Deadline for submission of thesis:** "21."May 2019

Student: M.M. Danushka R Marasinghe ..... ".21" May 2019

/signature/

Supervisor: Professor Mart Tamre ..... "....."....201….a

/signature/

CONTENTS

PREFACE

This thesis was proposed by Professor Mart Tamre and the author has enthusiastically accepted the challenge of perusing it as my Masters Thesis. The author has a passion in Mechatronics, Industry 4.0, robotics, and researching on methods to make the solutions more user friendly and efficient and affordable. The work on the thesis has been done in Tallinn Estonia with the use of the hardware and software available at the computer labs in the Department of Mechatronics of Taltech University.

The thesis is about teaching ABB YuMi collaborative robot to perform movements using a RGB-D depth camera (Intel RealSense) by a human operator. These movements are recorded in a database and later use them to perform the tasks. User interfaces in both in the PC software and in the robot are used for easy operation.

The author would like to express his deepest gratitude to Professor Mart Tamre, and to the staff of the Mechatronics department for the immense support provided to me whenever I needed to conduct my studies as well as the thesis. Also, to Taltech University for accepting as a Masters student and providing all the facilities for a remarkable study experience.

A special thanks goes to my family for their backing to stay outside many miles away from home to my fiancée, and to my friends and Estonia for all the moral support given to me to fulfil my dreams to the best way possible.

Keywords: Collaborative robots, Depth Camera, Robot Learning, learning from demonstration, Master's Thesis.

List of abbreviations and symbols

- API - Application Programming Interface
- csv - Comma Separated Value
- MIPI - Mobile Industry Processor Interface
- RGB-D - Red Green Blue Depth
- R_x - Rotation along X Axis
- R_z - Rotation along Z Axis
- SDK - Software Development Kit

1 INTRODUCTION

Automation of the industry has been developing in an increasing pace where it has come to a point where full automation of a factory is a possibility. However, there is still a large set of manufacturing tasks that are tedious or strenuous for humans to perform. Some of these tasks, such as electronics or aircraft assembly, are difficult to automate because they require workers to collaborate in close proximity and adapt to each other's decisions and motions [1]. Collaborative robots on the other hand are designed to work along with the human operator where both perform simultaneous motions in close proximity to each other.

However, the collaborative robot needs to be programmed by a skilled person for the designated task to perform. Also, if the robot needs to be used for different motions or the human operator needs to change their motions the programming has to be changed again in the collaborative robot. In industry 4.0 the robots should be more autonomous, flexible, and cooperative, they will interact with one another and work safely side by side with humans and learn from them [2]. In the rapidly changing industry flexibility is a key to perform tasks efficiently.

In order to make the collaborative robots to be more adaptable and make them more suitable smart manufacturing industry 4.0 standards, the solution would be to develop a system using vision based or Augmented Reality (AR) based methodologies where the robotic operator or other robots can let a particular robot to learn from their motions and perform its tasks accordingly.

The solution will be a good opportunity for the industries who can make the cost reduction by faster adapting to new demands in the factory floor, reduction in cost by making the robotic operator to program the motions of the collaborative robot to perform different tasks, they can easily train new operators. The solution is also suitable for industries who is planning to adapt industry 4.0 methods who plan to automate their systems. Also, the solution is viable as a demonstration and learning tool where a simple user can program simple tasks and learn the capabilities of the robot. Further success of this technology can path way to have robots in home environments where a simple user can teach robots to perform their tasks where the opportunities are endless.

ABB YuMi collaborative robot, Intel RealSense depth camera was used the main hardware and Unity, Nuitrak, Visual Studio 2017 and RobotStudio was used as the main software platform for this thesis.

1.1 Motivation

Earlier days robots in the industries required a separate workspace, required high skilled programmers to program the robots and was dangerous to work alongside with the robots. However, upon the introduction of collaboration robots which are designed to work with human operators in the work field the robots became more humane friendly. With the introduction of industry 4.0 and customization of each production became more vital. Thus, robots need to be programmed for new products with minimum downtime. Further the use of collaborative robots is not limited to manufacturing industries. They are currently used in other sectors such as entertainment industries , hospitals etc. where the users are not very skilled on robot programming.

The author of this report is passionate on collaborative robots and to research on how to make them more user friendly and convenient for the end user. He sees an opportunity in finding an approach to program the robot without having specific programming knowledge or using any special wearable devices and replicate and repeat the movement on the robot.

1.2 Objective of the study

The objective of this research is to find a possible solution to create a Cobot type robot learning system where the collaborative robots in the industry can be programmed by perceiving the movements of a human operator and later the robot can perform the tasks without any programming requirement.

This research aims to

i. Research on the current methods and technologies available.
ii. Develop possible approaches to obtain the solution.
iii. Deterring the optimum solution.
iv. Finalize the suitable hardware and software to use
v. Implement the system
vi. Test and verify that the objective is met.

1.3 Thesis structure

This section will give a brief description on the chapters on this thesis so that the reader can get a better understanding on what to read.

Chapter 1 provides an introduction for the thesis followed by the motivation and objectives of the study. The objectives are further details in the body of the report.

Chapter 2 contains an analysis of previous studies conducted in related to the objectives of the thesis referring to the published data. the section further discusses the justification of the scope of the thesis.

Chapter 3 consists of the methodology of the thesis where it describes the proposed solution, selection of the hardware and software that will be used in the project. Furthermore, it provides details on three conceptual approaches to the solution and selection of the most suitable approach.

Chapter 4 comprises detailed explanation on how the selected approach is developed. This includes the process flow chart, the programming methods used in PC and robot, implementation of data communication methods and explanation of the user interface.

Chapter 5 provides the details on limitation of the projects, justifications of the project, suggestions for further developments and the summary of the project.

.

2 LITERATURE REVIEW AND BACKGROUND

This chapter describes the existing solutions and research done one the collaborative robots and human robot collaboration, how the robots can learn from human movements, the importance of such technology will be discussed.

This section will further analyze the existing literature that explains the previous methods that are closely related to the objective of this thesis and will use the ideas to describe a potential solution for the cobot type learning system.

2.1 Existing Methods

With the introduction of industrial robots, many studies were conducted to further simplify the operations, improve the interaction with the humans and to automate the processes. In order to reach this goal, robots must learn from the interaction with humans and learn human skills [3].

In 1999 collaborative control message-based framework was introduced where the human operator is treated as a limited source of information and reduces the need of continuous human involvement. It allows the robot more freedom in execution and allowing it to better function if the operator is inattentive or making errors [4].

There are two key factors for achieving effective human-robot collaborations. They are roles and responsibilities must be assigned according to the capabilities of both the human and the robot and the second factor is that it should be easy for the human to effect control of the robot [5].

Sönke et al, 2011 of ABB have presented an easily portable dual arm robot with flexible grippers that is designed to work next to and in cooperation with assembly workers which can be deployed in an environment that is a hybrid between a manual, and an automatic assembly line [6]. This was the introduction of dual arm 7 link per arm ABB YuMi collaborative robot.

Matthias et al., 2011 argues that Human robot collaboration in collaborative manufacturing settings require a particularly detailed examination of injury risks and standards that will define human robot collaboration from safety perspective [7].

2013, Jim Mainprice and Dimitry Bernson presented a framework that allows the human, and the robot to perform simultaneous manipulation safely to enable collaborative tasks in close proximity based on early prediction of the human motion [1].

Approaches involving capturing full-body motions of a human performer using an optical tracking device, which provides 3D locations of identified active markers that are currently in view were done by [8] where they presented a fully automatic technique for the generation and synchronization of kinematic models describing human and humanoid robot motion.

With the introduction of depth cameras significant research was conducted on Human-computer interaction and [9] built a system to recognize different body gestures and generate visual Human-Robot interaction interface, then the controlling signals of different body gesture modules are sent to the robot.

The technological advancement of the depth cameras allowed the researchers to move beyond the mere gesture recognition. [10] developed real-time human computer interface using the Kinect camera and achieved close to 100% practical recognition rates.

Augmented Reality is a novel approach into industry 4.0 worker support system where [11] used the robot assembly commands and worker assembly instructions can be virtually augmented for human workers intuitively and instantly.

2.2 Learning through demonstration

With the introduction of the collaboration robots into the industrial market extensive research was conducted on the safety of the robot as well as the human worker or operator and the skill requirement for the human worker to program the robot in terms of customizations. Following review describes the current research on this area.

The modern-day robotic applications bring robots into unstructured environments such as houses, hospitals, museums etc. These dynamic environments are uncertain, and it makes hard coding an unfeasible approach [12]. This article proposes the human operator to use kinesthetic teaching to teach the robot by physically involving with the robot movements.

Suleman et al., 2010 tried to use Artificial Neural Networks to perform learning of robot actions from demonstration where a (learner) robot learns an action by observing a demonstrator robot performing the same [13] . The method hacks into the sensor reading of the learner robot to obtain the data initially which is not very straightforward in the collaborative robot working environment.

Gu Ye et al., 2017 developed a framework by using a Baxtor robot, a Kinect RGB-D camera, motor driven turntable, 3D printed object. In this method the operator demonstrates the primitive actions and assembly tasks through kinesthetic teaching. In order for the robot to recognize the actions for each action, the robot arm is guided to the desired pose multiple times from different initial poses [14]. This method is able to teach limited number of movements and does have a drawback on the ability to program the movements of the robot easily for multi joint collaborative robots.

Zhao et al., 2016 used RGB-D sensor for hand tracking and Matlab to generate joint coordinates using inverse kinematics. This method provided hand tracking coordinates at some positions and did have jerks in z direction coordinates [15]. Therefore, the robot did not move to those coordinates when the values are not accurate.

2.3 Justification of the scope

There were continuous research and demonstrations on how to make the robots to collaborate and work efficiently and effectively with humans and how to quickly program robots. The above literature review suggests that the robots have gradually involved from heavy dangerous industrial robots to collaborative robots that are convenient and safe to use.

There were many researches have to make the robots understand that the movements of other robots or human operators. These studies were successful in their own scope but there seems to be a scarce for collaborative robots. there is an opportunity to create a marker less, easily to use teaching system for collaborative robots using newly available RGB-D cameras, and the collaborative robots can play those training algorithms at later stages. The idea is to make the system as user friendly as possible so that even an operator with a little knowledge can operate and teach collaborative robots the required movements.

3 METHODOLOGY

The chapter will focus on the detailed explanation of the implementation of the proposed solution. The first section of the chapter will explain the overview of the proposed solution.

The second, third and the fourth chapters respectively will provide an explanation of the hardware that was used, software that was selected and the communication parameters that was used in the project.

The final section of the chapter will explain the comparison of the feasibility of the alternative approaches that was tested prior to the selection of the desired method.

3.1 Proposed solution

As discussed in chapter 2 the collaborative type robot learning system once completed should be able to program the selected robot without any programming knowledge. In the solution the robot should be able to learn the movements of the operator. The operator is not wearing and special hardware. Once learnt the robot should be able to repeat those leant movements perform required tasks at any given time.

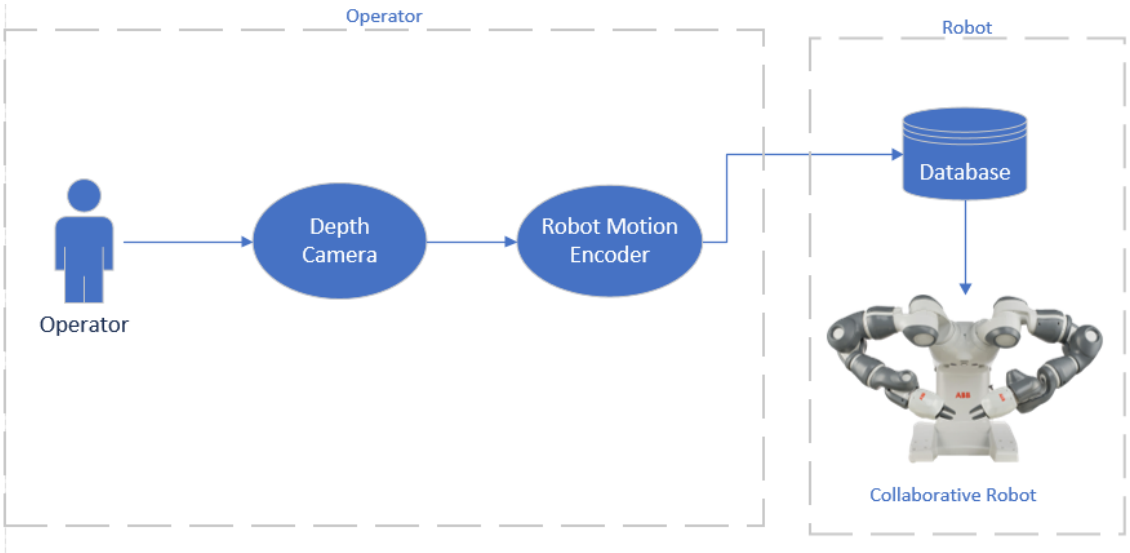

Figure 3-1 Overview of the Cobot type learning system

The proposed solution is described in the above Figure 3-1 describes the outline of the solution. since the system need to capture the operator's movements in the 3D space a depth camera is used for capturing that data. The selection process of the depth camera will be discussed in the section 3.2 below. A user interface is provided to the user to easily operate the training module of the overview where the live moments related to the movements of the robot is showed in real-time. The user interface will have different controls and will provide detailed explanation of that in the section 4.4.

The operator needs to stay in front of the depth camera and perform the calibration procedure to tune to his or her position data. Once the calibration process is completed the operator should connect with the robot. Upon the connection the operator can mimic the movements of the robot arms and perform the task that needs to be programmed to the robot. The encoding decoding mechanism will perform required inverse kinematics and hence the robot will also move according to the movements and will confirm that the task is successfully performed.

The data of the said movement will be saved in a database for future reference and that data can be used to perform the tasks of the robot without any usage of any special programming.

3.2 Hardware selection

The hardware selection is an important section of the solution. It consists of two critical hardware namely the collaborative robot and depth camera. The detailed description will be as below.

3.2.1 ABB YuMi Collaborative robot

The initial identification of the robots are industrial robots where it was required for a highly skilled programmer to program the robot and the robot had to be situated in a separate area where it was dangerous for any human operator to move closed during its operation. Also, if the robots are deployed in a dynamic setting, the operator often must spend significant time resituating himself each time he switches context [16]. The growing of the flexible manufacturing methods in the industries demanded versatile robots to the market. One requirement was the ability of bringing humans and robots closer together in the manufacturing working environment [17]. The

collaborative robots provide an efficient mechanism for adaptation, to adjust autonomy and human–robot interaction to fit situational needs and user capabilities [16].

There are many collaborative robots in the market and ABB YUMI IRB14000 was selected for the project. The detailed explanation is given in the following section.

ABB YuMi IRB14000 is a first-generation dual arm collaborative robot by ABB robotics [18] . The name YuMi is derived from the "You and Me "which as the name suggests developed to work closely with humans in industries [19] . YuMi robot is lightweight and does not need to be covered using any protective area resulting in a safer working environment for both worker and the robot.

Figure 3-2 ABB YuMi Collaborative robot [19].

Each arm of the robot has 7 axes offering more flexibility of movement. The overview of the arms and the positive and negative directions of the YuMi robot are shown in the Figure 3-3 below.

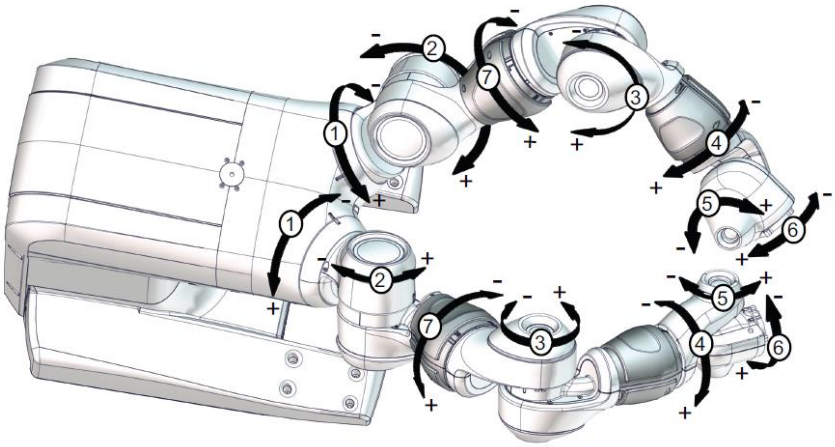

Figure 3-3 ABB YuMi Robot Axis overview [18].

As per the above illustration, the 7 axes of each arm can mimic the movements of a human arm to the best way possible. The below table explains the relationship of the axis of the Yumi robot with the joints of the human arm that has been deduced by the author of this project, type of motions of each axis and their degree of motions limits.

Table 3-1 YuMi's motion parameters for each axis [18].

Axis	**Type of motion**	**Movement of the human arm**	**Degree of motion**
Axis 1	Rotation Motion	Shoulder R_z	-168.5° to +168.5 °
Axis 2	Bend Motion	Shoulder Rx	-143.5° to +43.5 °
Axis 7	Rotation Motion	Elbow Rz	-168.5° to +168.5 °
Axis 3	Bend Motion	Elbow Rx	-123.5° to +80.0 °
Axis 4	Rotation Motion	Wrist Rz	-290.0° to +290.0 °
Axis 5	Bend Motion	Wrist Rx	-88.0° to +138.0 °
Axis 6	Rotation Motion	Hand Rz	-229.0° to +229.0 °

As explained in the above Table 3-1 the movement of each axis of the arm involved 5 motions and each axis has a limit of motion in degrees but are more flexible than the movements angles of the human hand. Additionally, it was impossible to directly relate the movement of each joint in the human arm to the movement of axis in YuMi arm as it has a greater number of joints. Therefore, the author had to relate the rotation around different axis (X, Y, Z) of the human arm to the Yumi arm as explained in the table.

Below Table 3-2 shows the default joint target values in degrees for each arm.

Table 3-2 Home position joint values in YuMi

Axis	Home position coordinates in degrees	
	Left arm	Right arm
1	0	0
2	-130	-130
3	30	30
4	0	0
5	40	40
6	0	0
7	135	-135

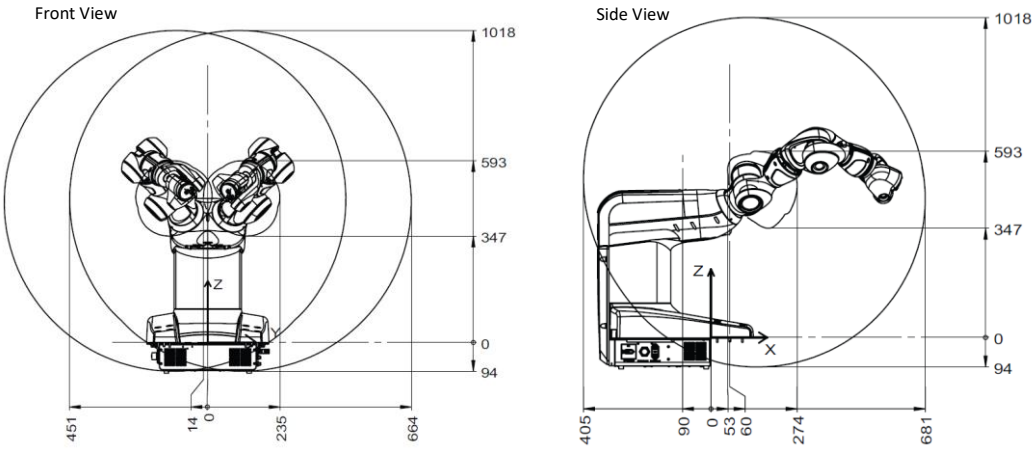

Figure 3-4 Working range of the YuMi Robot [18]

The working range for each arm of the Yumi robot are shown in Figure 3-4. Each arm has movements of approximately 664 mm along the Y axis, 1018 mm along Z axis and 681 mm along X axis. These values are important in determining the boundaries when matching the movement of the human arm with the robot arm. This will be further discussed in section 4.2 of this report.

Figure 3-5 ABB FlexPendant [20].

FlexPendant is a handheld operator unit that is connected to the robot. Figure 3-5 shows an image of a FlexPendant. The device has a user interface and it helps the user to perform tasks on the robot such as running programs, change settings of the robot, modify or create programs, jogging the manipulator etc.

3.2.2 Depth Camera

As described in Figure 3-1, the operator is moving his/her hand and those movements needs to be captured in 3D space without wearing any additional device. Therefore, a normal camera will not be sufficed. The hardware has to be an RGB-D camera. It is a specific type of depth sensing device with an RGB camera, that are able to augment the conventional image with depth information in a per-pixel basis [21]. The commercial RGBD cameras were introduced to the market by Microsoft Kinect and currently there are more compact advanced and affordable cameras in the market.

Table 3-3 Comparison of available RGBD Cameras [22]

Device	Technology	Range (m)	Resolution	Frame Rate (fps)	Field of View	Availability
Kinect V1	Structured Light	0.8-4.0	640 x 480	30	57^o x 43^o	No production
Kinect V2	Time of Flight	0.5-4.5	512 x 424	30	70^o x 60^o	No production
Intel RealSense D435	Active Stereoscopy	0.2-4.5	1280 x 720	90	85.2^o x 58^o	Available

As per the comparison of the available RGB-D cameras in Table 3-3, the Kinect V1 and V2 are out of production which means that the support is limited. The intel RealSense camera has better range, resolution, frame rate and field of view. Therefore, upon analyzing the pros and cons the Intel RealSense D435 camera was selected for this project.

Intel RealSense D435 is a USB3.0 Type C powered camera with a pair of depth sensors, an RGB sensor and an infrared projector [23]. The camera has a powerful 28 nanometer processor with capability to compute real-time depth images and accelerate output, it has a dedicated color image processor and active infrared projector to boost depth data [24].

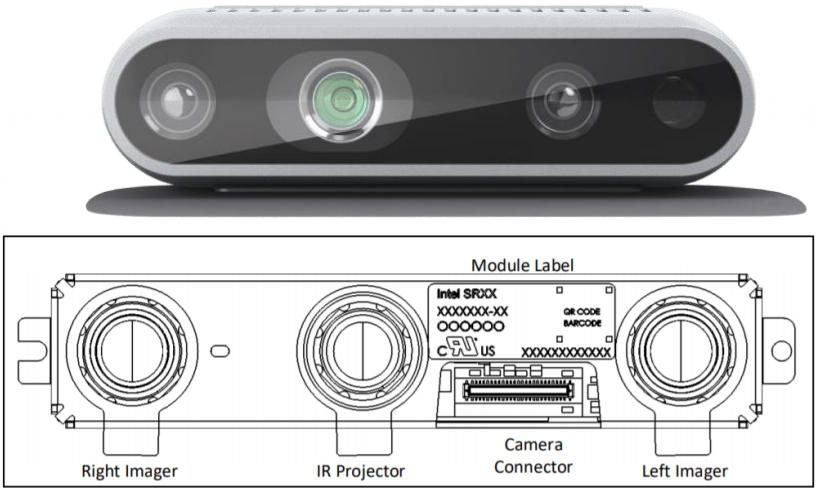

Figure 3-6 Intel RealSense D435 camera and sensor details [23].

The top image of the Figure 3-6 shows the actual view of the camera. It is compact with the dimensions of 90mm x 25mm x 25mm. The bottom image shows the included hardware camera modules. The camera can be connected to the PC using a USB 3.0 connection and the camera has a USB Type C interface. The camera is affordable and lightweight. The camera supports 3D scanning, Skeletal and people tracking, object measurement, augmented reality, virtual reality etc.

Depth camera placement is a very important aspect in order to get accurate results. Since it's a depth camera the following criteria must be met.

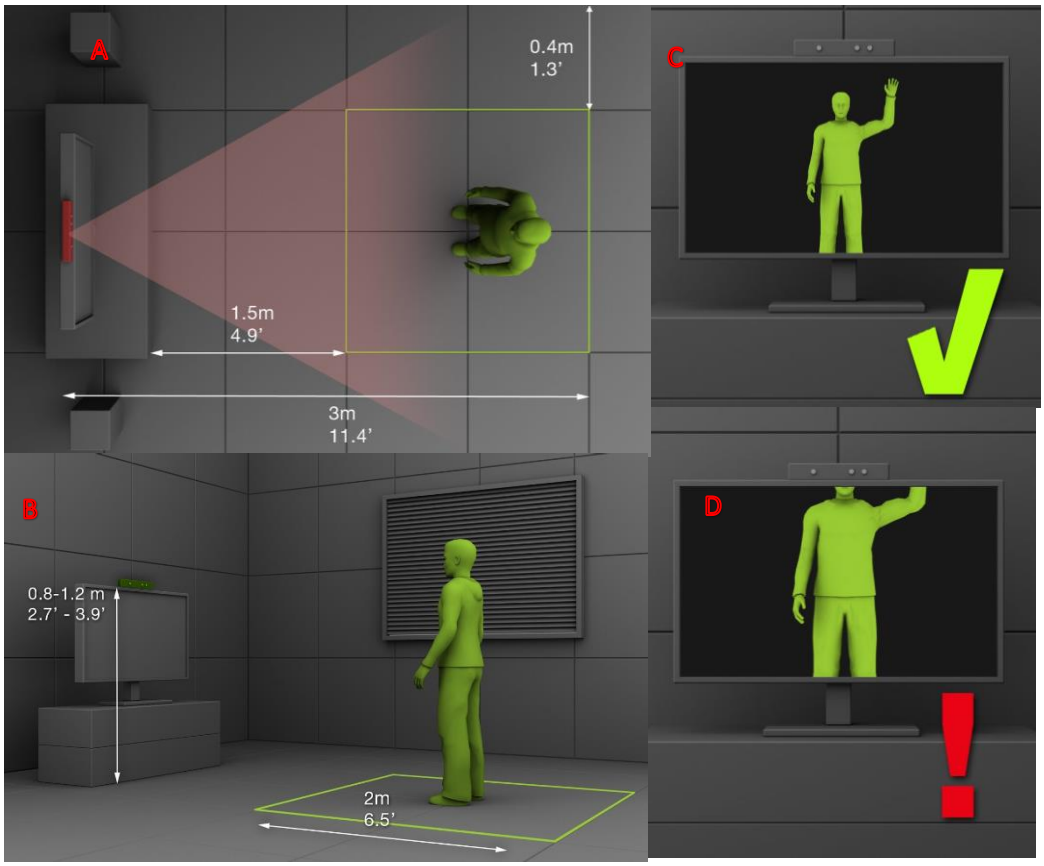

Figure 3-7 Depth camera placement instructions [25]

As shown in Figure 3-8 A the maximum distance of the user from camera is 3m and minimum is 1.5m. There should be minimum of 0.4m from a wall and should be sufficient space for the user to move without touching the walls.

As per Figure 3-8 B the ideal height of the camera is about 1.2m from the ground and should be parallel.

As per Figure 3-8 C and D, the total upper body should be visible in the camera.

Further, there should not be direct sunlight on the field of view of the camera and dark clothing should be avoided for better tracking of the user.

3.3 Software selection

Since the hardware selection is completed the next step would be to select the compatible software for our use case which will be discussed deeply in this section.

3.3.1 Middleware software

When the RealSense sensor is connected to the software it is required to have a middleware software obtain the depth information and convert it into skeletal tracking, obtain joint coordinates and hand gesture recognition. Two middleware software were used for this purpose namely RealSense SDK2.0 and NuiTrack SDK.

RealSense SDK2.0 is a cross platform operating system independent software that connect the PC with the camera. The APIs in the SDK lets the developed applications to access the functionalities of the camera.

Nuitrack SDK is a cross platform 3D body (skeletal) tracking middleware that supports both Kinect and Intel RealSense depth cameras. The system has features such as full skeletal body tracking, user masks, 3D points cloud, gesture recognition and face tracking [26]. The middleware also supports cross platform such as Android, Unity3D, C# development which provides the versatility to develop the application for many use cases.

The main feature that used for the project is the ability of accurate skeletal model tracking. Skeletal model is where the skeleton of a person in front of the depth camera is represented as a set of joints with its position and orientation and connected with imaginary bone. The movements of the model are according to the movements of the operator in real-time.

Figure 3-8 Nuitrack Skeletal tacking model [26].

As illustrated in the above Figure 3-8 the Nuitrack sdk can track the position and orientation of 19 joints in the body. The SDK also capable of recognizing 6 types of gestures namely waving, swipe left, right, up, down and push. The reason to choose this sdk for the project is its ability of accurate skeleton tracking, gesture recognition and the compatibility of multiple depth camera types. Unity version 2018.3.12f1 (64-bit) was used for this project.

3.3.2 Unity

Unity is a cross platform real time platform that enables to build 3D immersive experiences. Unity can be used to build user interfaces fast and spontaneously. Both middleware software discussed above are fully supported for Unity. Therefore, it is possible to perform the required skeleton tracking, let the user control the functionalities, build the interactive user interface faster.

The overview of Unity software is shown in Figure 3-9 below.

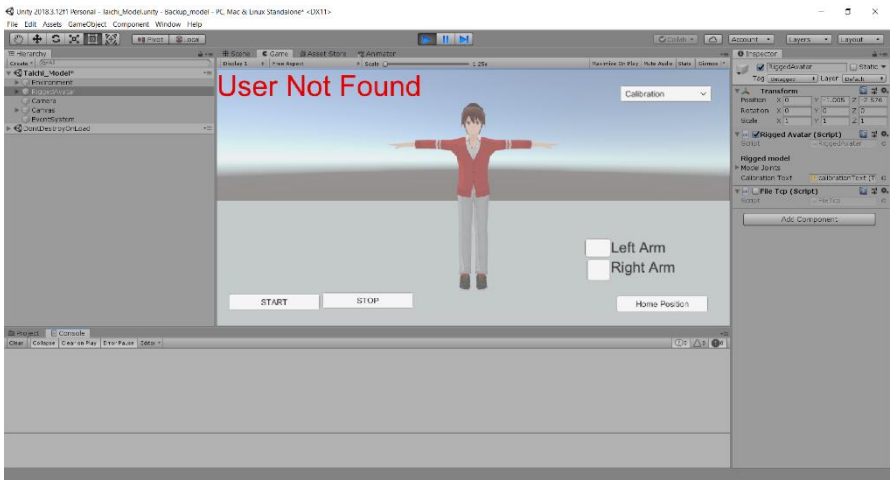

Figure 3-9 Overview of Unity

Coordinate system in Unity is shown in **Error! Reference source not found.** below. As per the c oordinates Y is acting upwards, Z is forward and X is sideways.

Figure 3-10 Coordinate System in Unity

3.3.3 RobotStudio®

RobotStudio is a software built by ABB Robotics. ABB robotics is the same manufacturer who build the YuMi robot. The RobotStudio software is built on the ABB Virtual Controller, an exact copy of the real software that runs on the ABB robot [27].

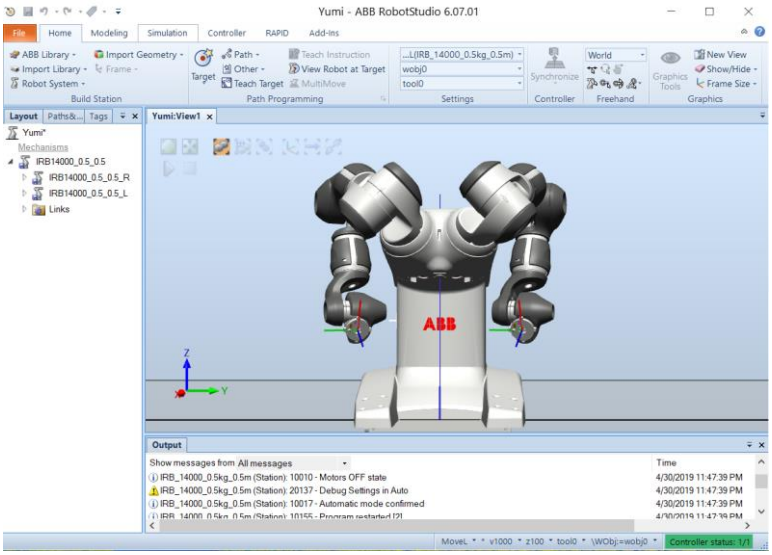

Figure 3-11 Overview of RobotStudio and coordinate system.

As shown in Figure 3-11 the software offers a virtual control of any ABB robot for configure, program and test prior to physical implementation. This helps the programmer to test all the aspects of the robot that results in reduction of risk in the industrial environment, does not need to shut down any productions for programming or testing and increase the productivity in the factory. RobotStudio version 6.07.01 is used for the project.

Coordinate system in RobotStudio as shown in Figure 3-11 has two coordinate systems. World coordinate and tool coordinate. In World Coordinates the Z axis is upwards, X axis is forward and Y axis is sideways. In default tool coordinates the X axis is upwards, Z axis is forwards and Y is sideways.

RAPID is a high-level programming language developed by ABB to program ABB robots. The RAPID instructions are there to control every aspect of the robot such as configuration, movements of the robot, activating IOs, packet communication etc. RAPID was used in the project to receive the data sent by Unity and perform relevant instruction in the YuMi robot to store the data or move the robot as programmed.

3.3.4 Visual Studio 2017

Visual Studio 2017 is a development environment by Microsoft™ and it was used in the project to program required scripts C# language to interface the RealSense camera, Nuitrack SDK, Unity and to send and receive data from RobotStudio.

3.4 Connectivity

As described in the overview of the system in Figure 3-1 the depth camera connects to the PC and the robot connects to the PC.

The connectivity between RealSense camera and the PC is done via USB 3.0 connection. The camera will not work properly without the 3.0 connection. As per the specification without the USB 3.0 connectivity the MIPI data reading time will slow down by 4 folds, will increase the noise in the feed, will not work at the maximum framerate due to lack of bandwidth [28]. Therefore, the manufacturer insists to connect the camera to a USB 3.0 port in the PC.

The connectivity between the PC and the ABB YuMi robot is handled via a TCP/IP connection. The Local Area Network connection is implemented between the PC and the robot. The server for the TCP/IP connection is done in the robot and the client is the PC. The details of the TCP/IP packet and how the communication is implemented will be further discussed in section 4.3 below.

3.5 Conceptual approaches to the solution

At the initial phase of the project it was brainstormed on how to implement the solution. There were three alternatives were suggested. Namely forward kinematics approach, inverse kinematics approach and work object coordinate approach.

i. Forward kinematics approach is when the operator makes the necessary movement in front of the depth camera, the skeleton tracking will determine the rotation along X, Y and Z for shoulder, elbow, wrist and hand in degrees. The values will then be assigned to the axis in the arms to YuMi robot as described in Table 3-1. Once the angles are calibrated with the home position of the robot arm the expectation is to move the robot in correspondence with the angles of the joints of the human hand.

ii. Inverse Kinematics approach is when the operator moves his / her hand in front of the depth camera, the skeleton tracking will determine the X, Y and Z coordinates of the wrist joint. The coordinates will assign to the end effector of the robot arm. In this approach once the human wrist position is calibrated with the home position of the robot arm end effector, the expectation is that all the axis of the robot will move to the position of the end effector using inverse kinematics.

iii. Work object coordinate approach is when the operator's hand position is registered in the depth camera the X Y and Z coordinates will be recorded. When the human hand is moved to a 3D point cloud within the working range area of the robot at end point will be determined as a work object coordinate. Therefore, in the program the robot arm will be taught to move to that particular position using inverse kinematics.

The recoding of the movement of the robot and perform its operations are planned to conduct in two stages. The first stage is recording the movements of the operator with the data that are readable by the YuMi robot. For this stage either angles or the coordinates will be recorded in real time in relation to the robot movement. These data will be stored in a separate file inside the robot file system with reference to the corresponding motion.

The second stage is to replay the data at later stages in the robot to perform their designated tasks. The movement data saved earlier can be accessed via the robot program and continuously

performed the designated task. For this stage there is no need of any additional hardware or software.

3.5.1 Analysis of the approaches.

Although there are three approaches for the desired outcome it is vital to select the best method in order to proceed with the project.

In order to select the best approach following factors needs to be considered.

i. Each arm of the ABB YuMi has 7 axis joints. Therefore, for forward kinematics each joint should have a joint angle.
ii. The human arm only has three joints namely shoulder, elbow and the wrist that can be analogies to the robot arm.
iii. YuMi robot has kinematic errors or collision detection in case the robot cannot reach the desired position.
iv. Error of the skeleton model and operator movement.

In the forward kinematics approach the Euler angles (rotations around X Y and Z axis) from the each joint of the skeleton model was considered.

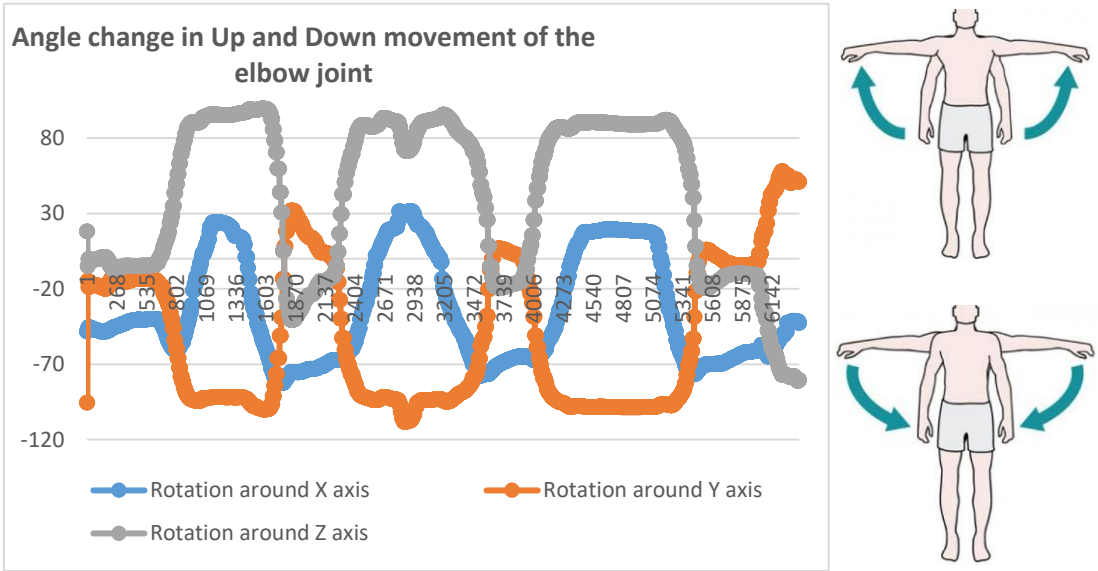

Figure 3-12 Angle change of up and down movement of elbow joint

In order to test the feasibility of the system the author checked the angle movements of each joints of the human arm using the skeleton model. In the above Figure 3-12 the author moved the arms as shown in the right side of the figure and the rotations along the X, Y and Z axis of the angles were recorded. As shown the elbow joint movement shows all three rotations has a change in the angle. Although the movement is consistent with the human hand movement it was difficult to find a relationship to the movement of the robot axis 7 and 3. When tested almost every time the YuMi robot stopped movements immediately with kinematics error.

In the inverse kinematics approach, the X, Y and Z coordinates of the wrist joint is used instead of the angles. In the skeleton model the wrist is moved from forward to up position and the X, Y and Z coordinates were monitored. The corresponding 3D plot is shown in Figure 3-13 below.

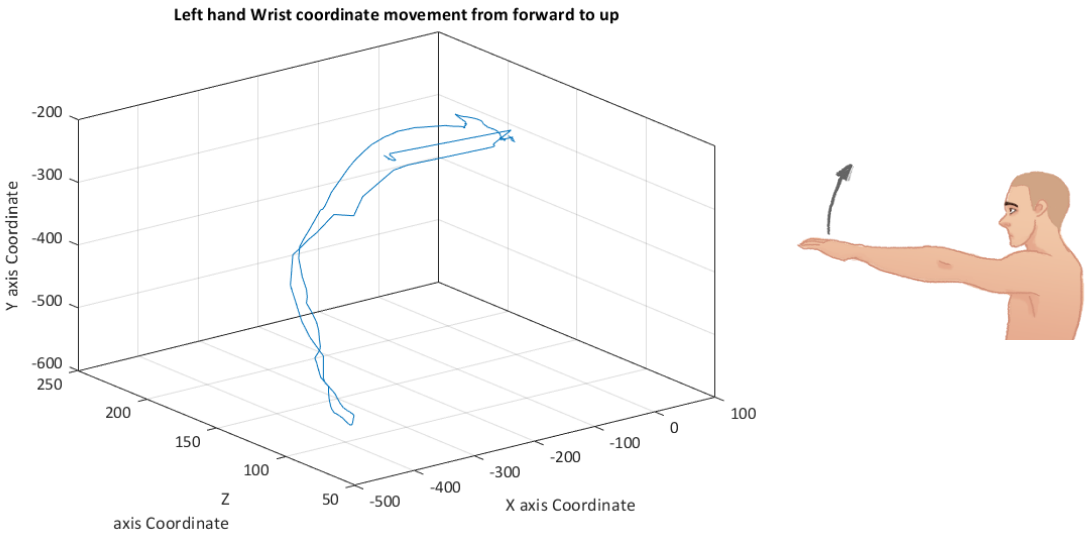

Figure 3-13 - Left Hand Wrist movement from forward to up [29].

As per the above figure the wrist movement of the author is mapped almost accurately in the skeleton model. When these corresponding coordinates are set to the YuMi robot's end effector coordinate the required angles for the axis of the robot arm joints are calculated automatically using inverse kinematics. Thus, resulting fewer errors on the robot movement.

In the work object coordinate approach the coordinated of the hand of the author is matched to the hand in the skeleton model. When the desired work object position is reached the coordinates are saved and the robot arm will move to the desired work object position. However, the accuracy of the position is highly depended on the measurement accuracy of the skeleton model obtained by the depth camera.

Table 3-4 Summery of analysis of alternative solutions

Alternative solution	**Effect of the criteria (out of 5)**			**Total**
	Error of the skeleton model	**Kinematics error of the robot**	**Matching skeleton data with axis data**	
Forward Kinematics Model	2	1	1	4/15
Inverse Kinematics Model	3	3	5	11/15
Work-object Coordinate model	1	3	3	7/15

The parameters discussed above for each solution is summarized in the below Table 3-4. Each criterion is rated out of 5 where the highest suitable is 5. The total is selected out of 15. Therefore, Inverse kinematics model is selected to continue with the project.

4 DEVELOPING THE SELECTED APPROACH

This chapter will discuss in detail on how the selected approach was developed. The first section of the chapter will provide an overview of the system with a detailed flowchart. This is important to provide a detailed outline of the system that was developed.

The second section of the chapter will explain on the works of the skeleton model and how it was implemented. The third section explains the architecture of the TCP/IP packet that was used for communication.

The forth section will explain the user interface and its functionality. The fifth section will provide a detailed description on the programming of the YuMi robot and how it will interact with the user interface.

The sixth and seventh sections will respectively describe how the saving and replay of the movement are done and the accuracy of the results.

4.1 Flowchart

The flowchart the detailed description of the logical step by step process on how the collaboration robot learning system will perform. The operation has two section. They are namely, motion recording and playing the motion in cobot.

4.1.1 Motion recording

Motion recording is the main section of the program. This section is responsible for capturing the movement of the operator using the depth camera, steps to be taken in the user interface to calibrate, capture the movement, send to the robot, operation of the robot on how to start and save the movement data. The flowchart is shown in the Figure 4-1 below.

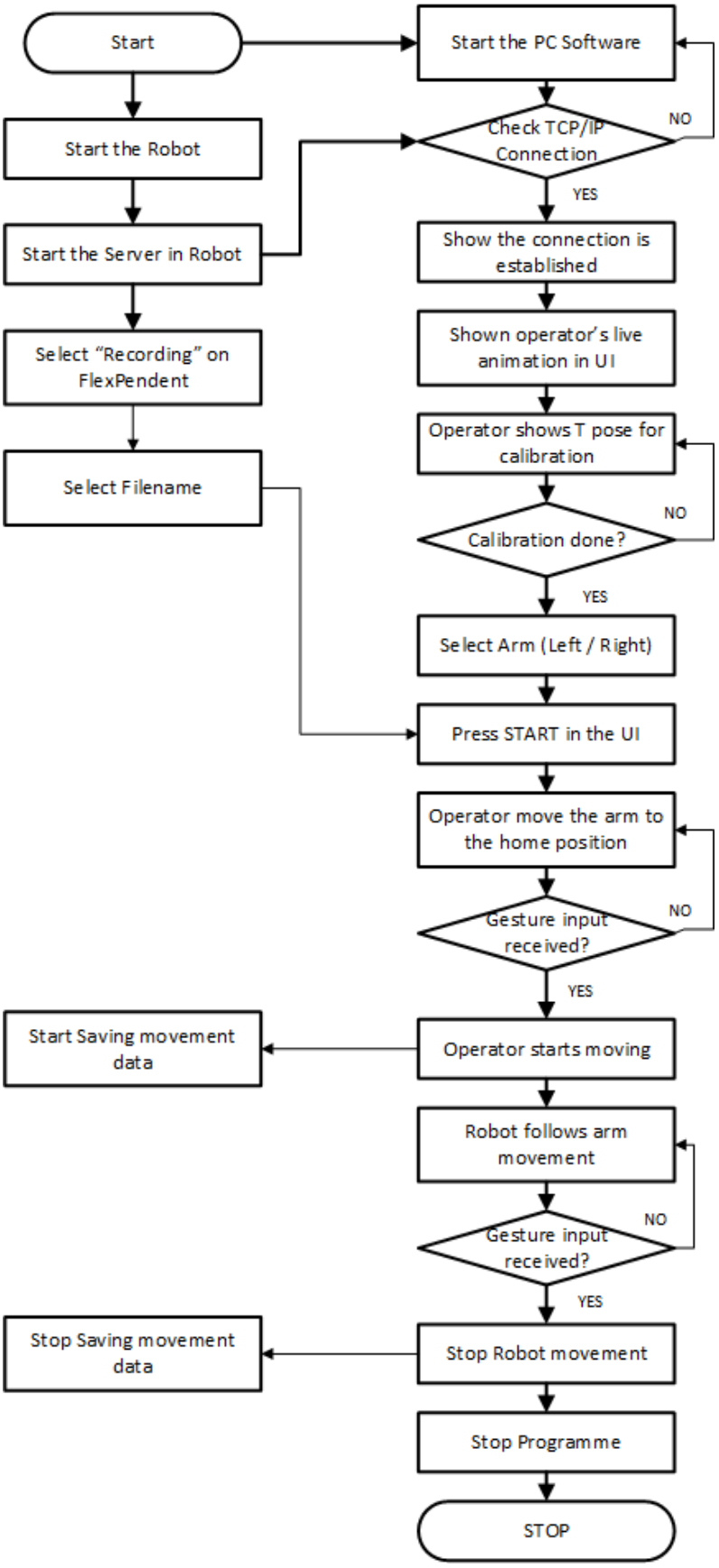

Figure 4-1 Flow Chart for Motion Recording

In order to start the process both collaboration robot and the UI PC software needs to be started. When the installed program in the robot is set to run it will start the socket server and will wait for the client in the PC software to respond. Once the connection is established it will be shown in the interface and the animation character will appear on the screen. The character will depict the movements of the operator. In the robot there will be a menu on the FlexPendant and needs to select “Recording”.

Next step is the calibration which will be further explained in the below section 4.2. Once the calibration is completed the operator should select which arm should be moved. Once the selection is done the operators needs to press Start on the user interface. Next the user needs to move the selected hand to the home position and use the swipe right gesture from the other hand to start the movement. Once the start movement is set the robot will also start recording the movement data for later use.

Accordingly, the robot arm will move to the movement of the user arm. Once the desired movements are obtained the user can do the swipe right gesture from the other hand to stop the movement of the robot and stop saving movement data in the robot. At the end the movement data will be saved in a database file in the robot.

4.1.2 Playing the motion in the robot

Playing the motion is the next section in the program. This is run from the robot and does not need any PC software. Once the program is saved using the process explained in section 4.1.1 the movement can be played from the robot. The flow chart is shown in Figure 4-2 below.

Initially the robot needs to be started and run the program installed in the robot. Once the program started there will be a section menu appear on the FlexPendant. The user needs to select the “Play” option and then select the file relevant to the movement that needs to be used. Once the file is selected the user needs to select single play mode or continuous play mode. In this process there is no requirement of any PC program or depth camera.

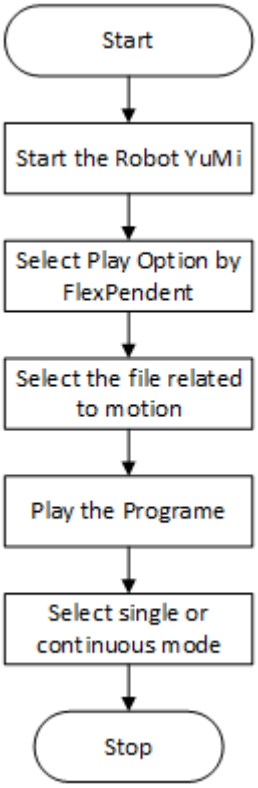

Figure 4-2 Flow chart for Playing the motion in the robot

4.2 Skeleton model

As described in section 3.3.1 and section 3.3.2 Nuitrack middleware, Intel RealSense SDK, and Unity were used to capture the movements of the operator using skeleton tracking.

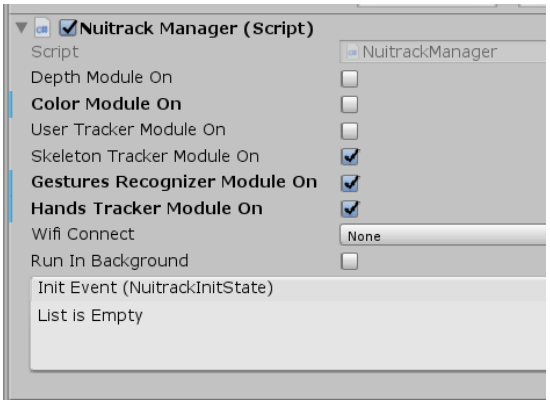

Figure 4-3 NuiTrack middleware manager in Unity

Once the Nuitrack SDK is installed there is a prefab in Unity called NuitrackScrips as shown in Figure 4-3. It enables the programmer to easily select the features that needs to be enabled in the program. In this case the author has enabled the skeleton tracking module, gesture recognizer module and hand tracker module. These selected features will continuously run in the background.

Following code in C# editor needs to be used in order to activate the skeleton model and sense the presence of the user in front of the camera.

```
        if (CurrentUserTracker.CurrentSkeleton != null)
         { ProcessSkeleton(CurrentUserTracker.CurrentSkeleton);
             userFound = "User found";        }
```

Once the user is detected “User found” will be displayed on the User Interface. If not “User Not Found” is displayed.

If the depth camera is places in front of the user, it is required to calculate the position of the whole skeleton model using the torso joint position and to make sure the model will move at the correct direction it is required to rotate it along Y axis by 180 degrees. This is done in the following code.

```
Vector3 torsoPos = Quaternion.Euler(0f, 180f, 0f) * (0.001f *
skeleton.GetJoint(nuitrack.JointType.Torso).ToVector3());
transform.position = torsoPos;
```

The skeleton model that was created using the Nuitrack middleware is shown in Figure 4-4.

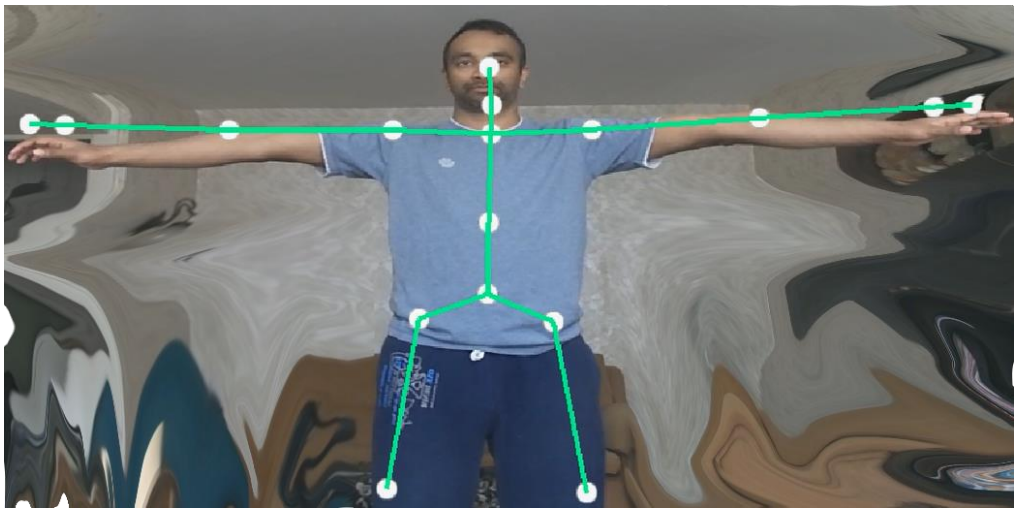

Figure 4-4 Skeleton tracking

The skeleton model able to track up to 19 joints in the human body. But for the project the data is obtained only from the following joints with reference to Figure 3-8.

- Head
- Neck
- Torso
- Left and right collar
- Left and right shoulder
- Left and right elbow
- Left and right wrist
- Left and right hand

4.2.1 Calibration of the skeleton model

In Nuitrack sdk it is required to conduct a t-pose calibration prior to start of any measurement. The t-pose is where the user arranges his/her pose as per the below Figure 4-5.

Figure 4-5 T-pose for calibration

During the t-pose calibration the rotation matrices of all joints are equal to identity matrix [25].

In the PC software the joint data will not be calculated until the calibration is done. The logic works that the user needs to be at the t-pose for 3 seconds. Once the calibration is completed “Calibration Done” will be displayed on the UI. The code that is responsible for calibration is shown below.

```
private bool CalibratingHand(nuitrack.Joint joint)
    {
        bool calibrationDone = false;
        calibrationText.enabled = true;
        calibText = "Move your hands for T position for Calibration";
        var startTime = DateTime.UtcNow;
        while (DateTime.UtcNow - startTime < TimeSpan.FromSeconds(3))
            {
                calibrationDone = true;
                calibText = "Calibration done";            }     }
```

4.2.2 Home Position of the hands.

The X, Y and Z coordinates of the left- and right-hand joints were used to determine the location movement as described in section 2.3. In order to calculate a fixed distance between the body and the hand the initial positions were determined using nonmoving joint in the body which is the waist as its analogies to the world coordinate positions of the YuMi robot as shown in Figure 3-11 where the world coordinate origin is located at the center of the robot.

$$X_{HP} = X_{Wrist} - X_{Hand} \quad (4.1)$$

Where X_{HP} – X or Y or Z coordinate of the Home position,

X_{Waist} - X or Y or Z coordinate of the waist,

X_{Hand} - X or Y or Z coordinate of the hand.

As described in equation (4.1) the coordinates of the home position are when the operator positions the human arm similar to the home position of the YuMi robot as shown in Figure 4-6.

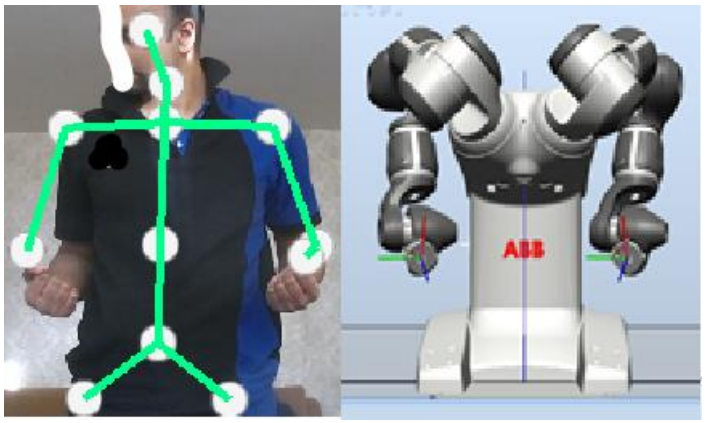

Figure 4-6 Home positions of the Operator and YuMi robot

Once the operator has put the arm into the home position the software will calculate the X, Y and Z coordinates of the position of the hand of that stance for 5 seconds and will take the average. This value is then subtracted from the result in equation 4.1 to get the absolute home position. The calculation is shown as below,

$$X_O = X_{HP} - X_{Ave} \quad (4.2)$$

Where X_O – X or Y or Z coordinate of the Origin Home Position,

X_{Ave} - X or Y or Z coordinate average values,

These values are close to zero. Therefore, any movement of the wrist from this position are calculated as from this origin.

4.2.3 Gesture controls

The YuMi robot had two arms and the developed program has the ability of recognize the movement of both hands of the operator and control both hands of the robot simultaneously. However due to practical issues such as controlling functions in the user interface when both hands are in use and one how to start and stop the recording, gesture control supported by Nuitrack middleware is used.

Therefore, when one hand is used to program the other hand is used for gestures. The Nuitrack system supports 6 types of gestures namely, waving, swipe left, swipe right, swipe up, swipe down and push.

For our project two types of gestures are being used. Swipe right to start the movement and swipe left to stop the program. The program will continuously monitor the gesture input during the runtime as described in section 4.1.

4.3 TCP/IP communication

One important aspect of the developed program is the communication between robot and the PC software. The YuMi robot has an ethernet interface and TCP/IP protocol was selected for communication. In the design YuMi robots acts as the server and the PC software acts as the client.

4.3.1 Packet design

TCP/IP packet was designed so that the server and the client will have a same data which was received and sent. And, to interpret the data sent by the PC software and decode to program it both PC software and RobotStudio. The packet design is explained in Table 4-1 below.

Table 4-1 TCP/IP Packet design

54 bytes →																
A 1	B 1	C 2	D 1	E 7	F 1	G 7	H 1	J 7	K 1	L 7	M 1	N 7	O 1	P 7	Q 1	R 1
A		Start Byte (Send character “S”)														
B,D,F,H,K,M,O and Q		Comma “,”														
C		The Control Message														
E, G and J		X, Y and Z coordinates of left wrist														
L, N and P		X, Y and Z coordinates of right wrist														
R		End Byte (Send character “E”)														
Sample Packet		S,01,-152.89,+307.35,-229.63,+040.27,+024.14,+005.92,E														
Packet data format		String														

As described in the above table, the byte stream will start with S character to notify the receiving end that it is the start of the packet. The receiving end will start unpacking data after reading the starting character. If the starting character is not read the data packet is discarded.

Commas were sent in between each instruction as it will be easy for save in a csv format in the robot for later use.

Next 2 bytes are the control bytes which is used to send controlling messages to the robot and PC such as start the program, right hand movement, left hand movement, stop, reset etc. The types of the control messages and their message IDs are explained in Table 4-2 below.

Table 4-2 Control Message description

Control Message	Function
01	Left hand movement
02	Right hand movement
03	Both hands movement
04	Return to Home position
05	Start
06	Stop
07	Reset

Next 3 instances of 7 bytes consists of X.Y and Z coordinate values of left wrist. Another 3 instances of 7 bytes contain the X.Y and Z coordinate values of right wrist. Each coordinate value had three integer number and 2 decimal number

The coordinate values can positive or negative depending on the position. So, in normal circumstances there is a minus sign when the value is negative. However, in order to preserve the total length of the byte stream constant the sign is sent on both negative and positive occasions. This was achieved by following the below command in c#.

```
jointsString.Append(initialPositionX.ToString("+000.00;-000.00;+0")
```

As shown in the above code extract the position data is converted to string and the sign (positive or negative) and the number length (3 integer and 2 decimal) is fixed for any value. The sign will be positive when the value is zero.

4.3.2 Client-side programming

The client side the PC software. Therefore, the TCP/IP communication is written in C# language. The client is running as a background thread and the thread communication is initiated as per the below code

```
clientReceiveThread = new Thread(new ThreadStart(ListenForData));
clientReceiveThread.IsBackground = true;
clientReceiveThread.Start();
```

There are two functions to receive data and send data. In the receive data function TcpClient is initiated for the IP address and port number. Then the stream data is read by opening a socket connection. The received byte array is then encoded using ASCII to read as a string. In the send data function the string message that was passed by other functions is converted into ASCII and send via the TCP/IP socket.

4.3.3 Server-side programming

The server for the TCP/IP connection is initiated at RobotStudio and programmed using RAPID language. There are three functions written to handle the communication. They are to create the socket connection, send receive encode and decode the data, send data and close the socket connection. Server creation is handled by the following code in RobotStudio

```
SocketCreate server;
SocketBind server,ServerIp,ServerPort;
SocketListen server;
SocketAccept server,client\ClientAddress:=clientIp\Time:=WAIT_MAX;
```

The SocketCreate, SocketBind, SocketListen and SocketAccept are standard functions in RAPID for create a TCP/IP connection. The IP address and the Port number should be the same as the numbers entered in the client side.

The next function is important as it will decode all the data received via the connection. The data stream will follow the packet format discussed in Table 4-1. Below is the code for this function.

```
SocketReceive client,\RawData:=data,\NoRecBytes:=noOfBytes;

IF noOfBytes=52 THEN
    UnpackRawBytes data,1,startHeader,\ASCII:=1;
    UnpackRawBytes data,2,controlMsg,\ASCII:=2;
    UnpackRawBytes data,5,leftHand{1},\ASCII:=7;
    UnpackRawBytes data,13,leftHand{2},\ASCII:=7;
    UnpackRawBytes data,21,leftHand{3},\ASCII:=7;
    UnpackRawBytes data,29,rightHand{1},\ASCII:=7;
    UnpackRawBytes data,37,rightHand{2},\ASCII:=7;
    UnpackRawBytes data,45,rightHand{3},\ASCII:=7;
    UnpackRawBytes data,52,endHeader,\ASCII:=1;
    ClearRawBytes data;
```

Once the byte stream is received the program will check whether the received size is 52 bytes. Then if it is true the program starts to decode using UnpakcRawBytes function. All the decoded values are then stored in string variables as per the packet definition. Later the received raw byte stream is cleared for the next iteration.

Last function is to close the server connection. It is required to close the socket connection then the program loop ends. It is done using the following code.

```
SocketClose server;
SocketClose client;
```

Error Handling has been done in both server side and client side so that in case to connection interruption the program will auto retry for connection and after timeout it will display an error message informing the user that the connection is failed.

4.4 User interface

This section will describe the user interfaces and their functionalities. There are two under interfaces namely for the PC software and for the robot.

4.4.1 User interface in PC software

The user interface of the PC software is an important aspect of the software. It shows the operator on how to movements are tracked and how to operate the functions as described in section 4.1.1.

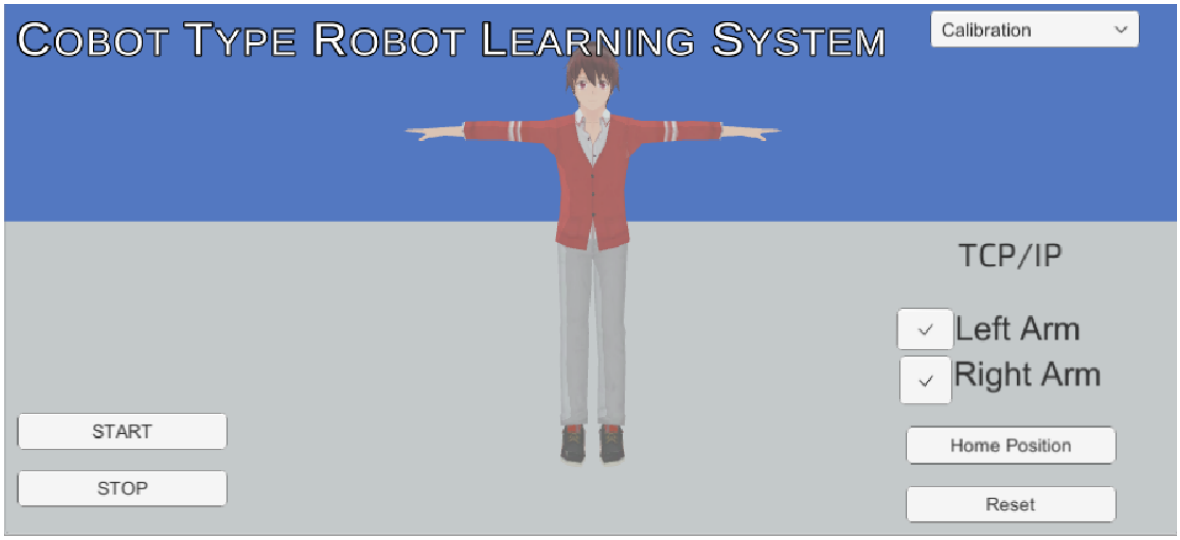

Figure 4-7 User Interface of the PC software

The above Figure 4-7 shows the user interface of the PC application. It runs on Windows platform and tested on windows 10 64 bit. The TCP/IP icon will turn to green once the connection is established with the server in the robot.

The character image shown on the screen would animate according to the movement of the operator once he or she stands in front of the camera. The operator needs to be inside the camera viewing specifications as described in Figure 3-7 in order to get an accurate and correct movement.

The “Start”, “Stop” “Home Position”, “Reset”, “Left Arm” and “Right Arm” buttons are selectable when the program is in running mode.

Operator recognition is a feature in the user interface where it will be displayed the availability of the user in front of the camera.

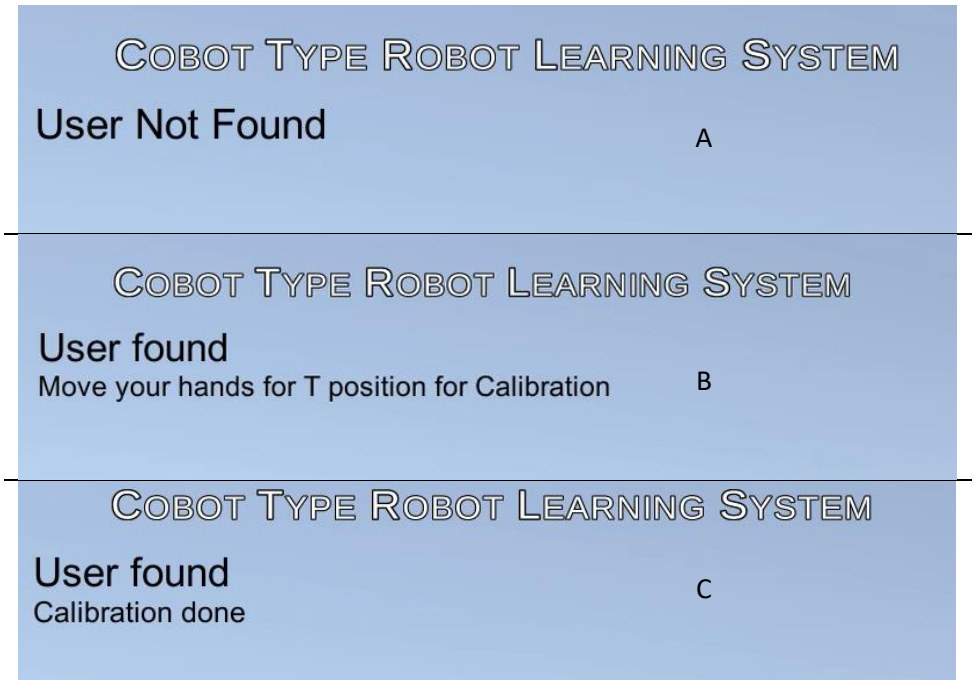

Figure 4-8 PC User Interface messages

The Figure 4-8 shows a part of the PC user interface when messages are displayed. The Figure 4-8-A displays the message “User Not Found” when the camera has not detected the operator present. When the camera was able to detect the skeleton model of the operator “User Found” will be displayed as shown in Figure 4-8-B

Calibration is done in order to get more accurate tracking details from the camera. The user will be notified as per the Figure 4-8-B to move the hands to the T position. Once the calibration is completed it will be notified to the operator as per Figure 4-8-C.

4.4.2 User interface in robot

When to ABB robot is powered on and start the program the user interface will appear on the FlexPendant. There are three functionalities that the use can select depending on the mode the operator wants to work on. The below flowchart in Figure 4-9 explains how the user interface will work.

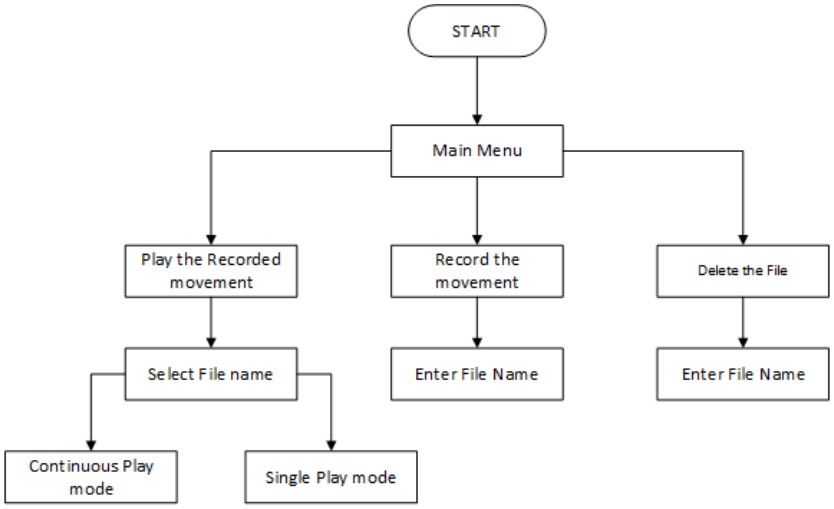

Figure 4-9 Flowchart of the RobotStudio UI

When the robot is started, and the program is in run mode the main menu will appear on the Flexpendant as shown in Figure 4-10.

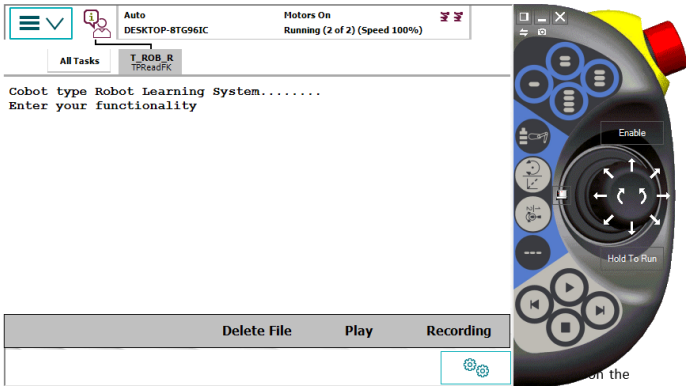

Figure 4-10 Main Menu in the robot UI

In the main menu there are three selections that the user can selection depending on the mode the want. The functions are namely “Delete File”, “Play” and “Recording”.

Recording mode is used when working with the PC software to train the robot of the movements as explain in the main flowchart in section 4.1.1. once this recording mode is selected there will be a prompt the enter a filename to save the movement data. The menu will appear as shown in Figure 4-11.

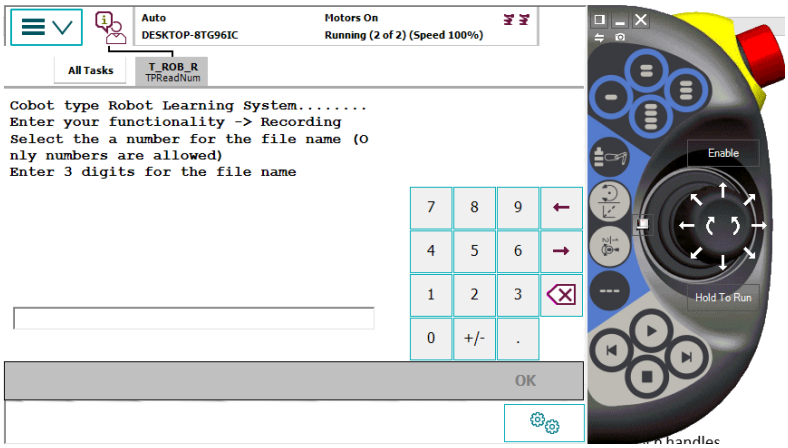

Figure 4-11 Recording Mode in Robot UI

Currently there will be only numbers are allowed to be entered into the filename. For example, if the user entered number 473 the filename will be saved in the following format. “CobotLearn473.csv”. next the program will set to record mode and will move to next step in the main flow chart described in section 4.1.1.

Play mode is used when the operator needs to play the recorded movement data once the learning to completed. In the play mode there will be two tier selection process as shown in Figure 4-9. First selection would be to select a required filename which the movement data was saved. The menu will be same as described in Figure 4-11. Once the filename is selected the next option is to select whether the motion needs to play continuously or single. The User Interface is shown in Figure 4-12 below.

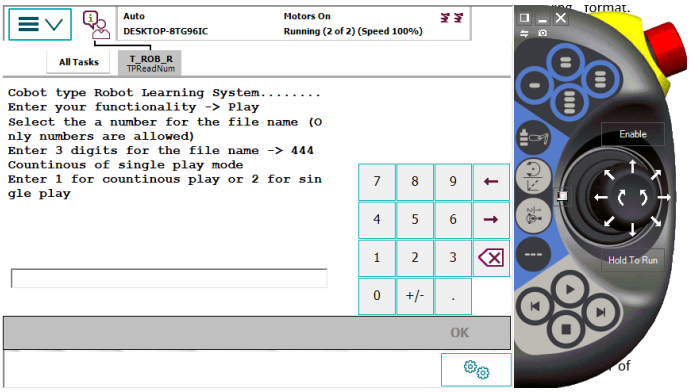

Figure 4-12 Play mode selection in robot UI

If the user selected 1 the learned program will play continuously until the user press the stop button. If the user selected 2 the learned program will only play once.

Delete mode is used if the operator wants to delete any learned program. Once selected there will be a prompt to enter the file name as in Figure 4-11 and then the file will be permanently deleted.

4.5 Programming the robot

This section will describe on how the programming has been done in the RobotStudio which handles all the functions, codes, instructions to control and play in the YuMi robot. A virtual controller of the YuMi robot which is the replica of the real YuMi robot was used for the programming and it is easily possible to port into the real YuMi robot without any issue.

Brief introduction of the RobotStudio software was given in section 3.3.3. , the description of the socket connection was provided in section 4.3.3 and the description of the user interface was given in section 4.4.2.

4.5.1 RobotStudio parameters

The virtual controller uses RobotWare version 6.07.1011. the following modules have been enabled in the system. These parameters can be changed by selecting “Change Options” menu in the “Controller” tab in RobotStudio. The summery of the required modules are explained the Table 4-3 below.

Table 4-3 Summery of control modules in RobotStudio

Control Module Code	**Control Module name**	**Control Module Code**	**Control Module name**
623-1	Multitasking	611-1	Path Recovery
604-1	Multimove Coordinated	612-1	Path Offset
709-1	DeviceNet Master/Slave	616-1	PC Interface
617-1	Flexpendant Interface	689-1	Externally Guided Motion
841-1	Ethernet/IP scanner Adapter	688-1	RobotStudio App Connect
603-1	Absolute Accuracy		

As per the above parameters Multimove coordinated is required in ABB YuMi to work with both arms. It offers the functionality for one hand to hold the work piece while the other hand performs the task. Multitasking refers in RAPID programming the ability to run multiple tasks parallelly, and it should be defined as separate TASK each. Ethernet/IP scanner adaptor is required to enable the TCP/IP communication between the robot and the PC software. FlexPendant Adaptor module interface is required to connect the FlexPendant to the robot. The user interface for the control of functions in the ABB YuMi is programmed in the FlexPendant. Path recovery module is used store the current movement path and restore to it in case of error in movement [30]. Externally Guided Motion (EGM) is used to drive the robot using external motion inputs and it supports error recovery via low level management within the robot. All these modules are important in the project.

The system configuration window of RobotStudio is shown in Figure 4-13 below.

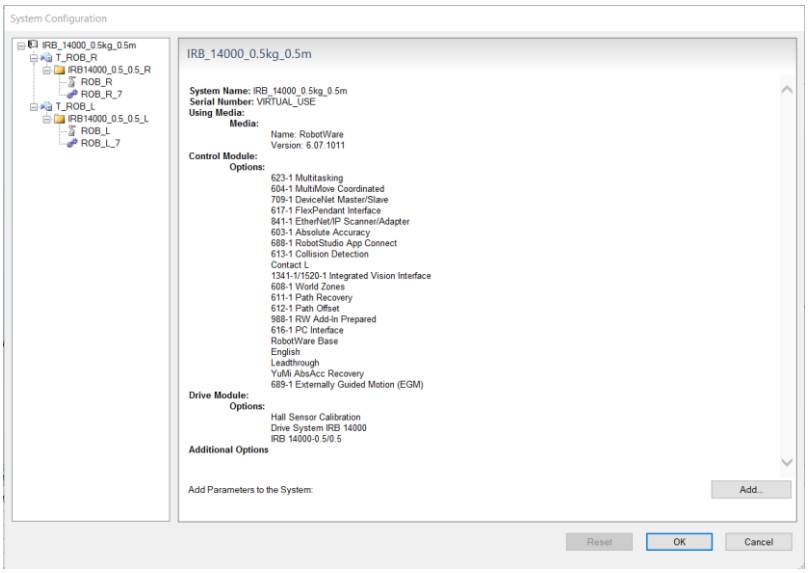

Figure 4-13 System Configuration in RobotStudio

4.5.2 RobotStudio programming

In this section the details on how the robot was programmed in the RobotStudio software was discussed. Since the YuMi robot has two arms they are programmed separately. The main program was designed to be run in the Right arm. The parameters and values that needs to control the left arm are shared through persistent variables called PERS. These types of variables will save the data in the permanent memory of the robot and will not get deleted when stopped or powered off.

The Main function has two loops each for “Recording” and “Play” functions. Corresponding loop was activated depending on the selection from the user interface.

In the Record loop the file will be cleared at first. And will wait for the TCP/IP connection to be activated. Then the received data will be unpacked as per section 4.3.3 and the left- and right-hand coordinates which are stored in arrays are converted to number. The coordinates for the left hand are stored as Persistent variable to pass for the left-hand movements. Next the function is called to follow procedure as per the control message received and followed by writing the movement data to the selected file.

In the Play loop the selected file will be opened and read the file. Similar to the above the data will be converted to numbers and use the function as per the control message. Next if the single mode is selected the robot will stop movement at the end of movement data in the file. The in continuous mode the movement data will repeat once the end is reached.

The movement of robot arm is a function in the program. In this function the coordinate values received from the PC software is matched with the coordinate system in the robot as explained in sections 3.3.2 and 3.3.3.

i. X coordinate of the robot to -Y of the PC software (depth camera)
ii. Y coordinate of the robot to -X of the PC software (depth camera)
iii. Z coordinate of the robot to Z of the PC software (depth camera)

Next the program will check whether the coordinates are within the limits of the lengths of the robot. Then the program will check whether the difference between the current and previous coordinate is bigger than 2mm. This is to avoid the “Shortmovement” error. Which results in the robot takes significant amount of time to interpolate the movements points when the difference between two points are smaller.

The control message function is programmed to act according to the control message received from the PC software of the saved movement file. According to the message id as explained in Table 4-2 there is a separate function programmed.

In order to make sure the left arm is operated according to the function a Boolean flag named “leftHandOn” passed between the function and the left arm. The code of the function is shown below. A similar flag named “leftHome” was used to distinguish the command to move the left arm and the command to move the arm to home position.

```
PROC SelectFunc(string function,num xcord,num ycord,num zcord)
    IF function="01" THEN
        !Left hand movements only
        leftHandOn:=TRUE;
    ELSEIF function="02" THEN
        !Right hand movements
        MoveRightArm xcord,ycord,zcord;
    ELSEIF function="03" THEN
        !Move both hands
        leftHandOn:=TRUE;
        leftHome:=FALSE;
        MoveRightArm xcord,ycord,zcord;
    ELSEIF function="04" THEN
        !Home position
        MoveAbsJ rotJ1_X,\NoEOffs,v1000\T:=3,z100,tool0\WObj:=wobj0;
        leftHandOn:=false;
        leftHome:=TRUE;
```

Home position of each arm of the robot is obtained by using the command “MoveAbsJ rotJ1_X,\NoEOffs,v1000\T:=3,z100,tool0\WObj:=wobj0;” The jointtarget values as explained in Table 3-2 of the home positions represented by “rotJ1_X” variable.

4.6 Saving and replay

This section describes the functions of saving the movement data and replaying them after. Both the operations are conducted in the robot software so that the robot can work as a standalone unit once the training is done.

4.6.1 Saving of the movement data

As described in the flowchart in section 4.1 the robot movement is saved to a file in the robot as a part of the learning in process. At the start of the process the user needs to enter the filename where the data is saved.

Table 4-4 Movement data save file byte format

A 2	B 1	C 1	D 1	E 7	F 1	G 7	H 1	J 7	K 1	L 1	M 1	N 7	O 1	P 7	Q 1	R 7
A		The Control Message														
B,D,F,H,K,M,O and Q		Comma “,”														
C		“L” letter to identify left hand parameters														
L		“R” letter to identify left hand parameters														
E, G and J		X, Y and Z coordinates of left wrist														
N, P and R		X, Y and Z coordinates of right wrist														
Sample Packet		01,L,-152.89,+307.35,-229.63,R,+040.27,+024.14,+005.92														

The data will be saved in a comma separated value (csv) file and the data format is given in the above Table 4-4.

The above data format is encoded and saved into the file in RobotStudio using the following code. It is required to open the file using “Append” command and close the file at each iteration.

```
PROC WriteData()
    Open FileLocation,File\Append;
    Write File,controlMsg+","\NoNewLine;
    Write File,"L,"+NumToStr(leftHandNum{1},2)+","\NoNewLine;
    Write File,NumToStr(leftHandNum{2},2)+","\NoNewLine;
    Write File,NumToStr(leftHandNum{3},2)+","\NoNewLine;
    Write File,"R,"+NumToStr(rightHandNum{1},2)+","\NoNewLine;
    Write File,NumToStr(rightHandNum{2},2)+","\NoNewLine;
    Write File,NumToStr(rightHandNum{3},2);
    Close File;
ENDPROC
```

4.6.2 Replaying the data

As described in section 4.1.2 the program has the functionality to use the saved movement data described in the above section to replay the learned movements when required by the operator. The saved file is decoded in RobotStudio using the following code.

```
PROC ReadData()
    controlMsgRead:=ReadStr(File\Delim:=",");
    leftSide:=ReadStr(File\Delim:=",");
    leftHandRead{1}:=ReadStr(File\Delim:=",");
    leftHandRead{2}:=ReadStr(File\Delim:=",");
    leftHandRead{3}:=ReadStr(File\Delim:=",");
    rightSide:=ReadStr(File\Delim:=",");
    rightHandRead{1}:=ReadStr(File\Delim:=",");
    rightHandRead{2}:=ReadStr(File\Delim:=",");
    rightHandRead{3}:=ReadStr(File\Delim:=","\RemoveCR);
```

As per the above code the coordinates were decoded and saved to variables for necessary motion commands as described in section 4.5.2. It is important to open the file before the loop in the main function and close the file after the loop in the main function.

4.7 Accuracy testing

When the system is developed it is important to check whether the response is adequate to get a real output. This section will discuss the accuracy of the system.

The testing was done to resemble a pick and place scenario which collaborative robots done frequently. The idea was to check the correctness of the measurements taken by the depth camera. The protocol is described in the following **Error! Reference source not found.**.

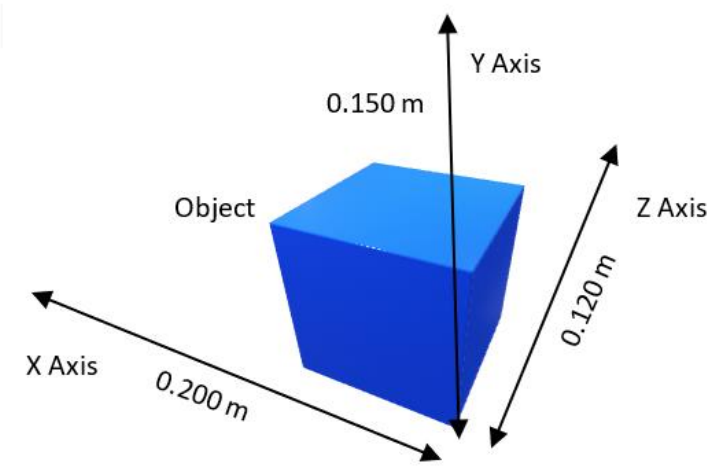

Figure 4-14 Accuracy test setup

The operator will stay in stay in front of the camera with an object placed in front of him. As shown in the figure the object is moved back and forth 0.20m along X axis, 0.15m along Y axis and 0.12m along Z axis and measured the coordinate value change from the depth camera. The collected data is then analyzed and the graphical illustration of the X, Y and Z axis movements in shown in appendix 3.

Table 4-5 Summery of the accuracy analysis

Axis	Actual movement / m	Measured forward movement / m	Absolute Error	Measured backward movement / m	Absolute Error
X	0,200	0,205	3%	0,189	5%
Y	0,150	0,138	8%	0,162	9%
Z	0,120	0,115	4%	0,128	7%

As per the above Table 4-5 the absolute measurement error for the backward and forward movement for X, Y and Z axis are less than 10%. The error significantly depends on the lighting and placement conditions of the camera and the confidence level of the skeleton tracking at that instance.

5 Discussion

In this section of the report, limitations of the project that the author encountered, justification of the completed project and suggestions for possible future improvements to make the end product more user friendly and efficient will be discussed.

5.1 Limitations

There were few limitations the encountered during the implementation of the project. They are briefed below.

- The accuracy of the movement of the robot has a very significant dependence on the accuracy of the data obtained from the depth camera. Therefore, it is very important to have proper lighting conditions and position of the operator as described in the section 3.3.1.

- Delay during the testing it was noticed that there is a slight delay when controlling the robot in real-time using the depth camera. There are two types of delays. The delay between the high-level controller (PC) and the control box will be referred to as the transmission delay [31]. The delay between the control box and the robot is called the actuation delay. The transmission delay of the YuMi robot is a result of the time the depth camera takes to process the feed, convert to skeleton tracking, process and send via Ethernet to the YuMi robot.

- The filename to save the movement data is only limited to numbers at the robot side as the user interface of FlexPendant only allows to read input as numbers. This issue will be needed to sort out in case of future development. Author believes that the FlexPendant SDK will have a workaround, but he was uable to do it during the time of the thesis.

- At the current setting although it is possible to make both arms of the YuMi robot to move simultaneously according to the movement of the human arm, it is only possible to teach and record one arm at a time. The reason is that the other arm is used to send gesture signals to start and stop the movement. However, this issue will not arise for single arm collaborative robots.

- The middleware software NuiTrack is not free. The free version only allows 3 minutes of use. It is required to purchase the full version in order to use for unlimited time [26].

5.2 Justification of the completed project

The Cobot type Robot learning system was a Masters thesis project was implemented using the resources with the affordability in mind. The depth camera used was Intel RealSense which is affordable, compact size yet offers significant capabilities. The operator can easily perform the movement as the system will auto detect the person in front of the camera. There is a PC software with the user-friendly interface where the operator can control critical functions and view their movement real-time on the screen in an animated humanoid character. There is another user interface on the robot to control critical functions. Once the teaching is done the user can play those movements on the robot itself. There is no requirement for the operator to know robot programming languages.

The accuracy of the results which discussed in section 4.7 is within the acceptable range. However further accuracy can be obtained by having proper lighting and placements of the sensor.

However, the measurement accuracy of the depth sensor relies heavily on the placement and the lighting conditions of the sensor, colour of the dress of the operator, movement speed of the operator. Therefore, it is vital to focus on these factors prior to operation of the program.

5.3 Future improvements

The idea of the project is to increase the efficiency of robots as well as human workers who collaboratively work together. This section will describe further improvements that the author can suggest improving the system.

- Introducing new tools such as Augmented Reality (AR) or Mixed Reality (MR) into the product. By incorporating AR or MR the operator can visualize the exact setting at the collaborative robot virtually and the operator can see from the viewpoint of the robot. The advantage of AR is that the workobjects can be augmented in 3D space and the operator can exactly work with them as the robot should. Then the operator can teach the robot from a remote location as well. AR glasses such as Hololens™ can be used for this purpose.

- From the experiments during the thesis project the author realized that the ABB YuMi has many safety protocols in place to stop the motion when sudden jitter in the motion is detected. The whole reason is because it is developed as a collaborative robot. However, in instances such as the teaching process in the project, these safety protocols can become an issue as it will always stop the robot movement. In order to overcome this ABB introduced a feature called Externally Guided Motion (EGM). it is a fast-low level interface to the robot controller's path planning. It can be used to change the robot path with high responsiveness based on input from external devices. The function is designed for advanced users [32] . Therefore, EGM can be used in this project to have more fluid and error free movements. It is also important to do more research on how to playback those EGM movements from the robot itself.

- The project can be easily converted to work with any type of collaborative robot not only ABB YuMi. this can be done by modifying the user interface to set some initializing parameters for each robot such as no of joints, end effector coordinates, robot workable area etc.

- With complying with Industry 4.0 it will be beneficial to consider the programming of the robot to newer and faster communications platforms such as 5G from typical ethernet protocol. This will provide high speed data communication and allowing delay free AR and MR solutions in future developments.

SUMMARY

The Cobot type robot learning system was a Masters thesis topic that was aimed at enabling operator or any human user to effortlessly teach the movements for tasks possible with the robot by moving his or her arm. These movements can be later played at the robot to conduct the designated task.

In order to implement the idea, the initial step was to investigate on the existing technologies available. It was found out from the analysis that there are existing systems that does robot program by demonstration but there were no such viable projects that the author could find where an ABB YuMi robot can be easily programmed using a depth camera skeleton data and later use that data to replicate the program to move the robot in their own.

ABB YuMi robot and Intel RealSense D435 camera used as hardware for the project and Unity was used for program the PC interface and RobotStudio was used to program the robot. The skeleton data and gesture controls were used from the depth camera module. The coordinates of the wrist are matched with the coordinates of the end effector of the robot arm.

The working of the program was tested, and it was found out that it is very important for the operator to comply with lighting and placement of the depth sensor in order to get accurate data from the depth sensor. The system developed can be used to program one arm at a time as the other hand is used for gesture controls.

Two user interfaces were developed one for PC which helps the operator to teach the robot. the other interface is in the robot which helps the operator to select recoding mode and to play the recorded movement for the designated task without the use of the depth camera or PC interface. Communication between the PC and the robot was done using TCP/IP protocol via the ethernet connection. Communication packet protocol was designed in order to minimize data loss during transmission.

Eventually the system helps the human operators to program a collaborative robot quickly and effortlessly with minimum technical knowledge.

LIST OF REFERENCES

[1] J. Mainprice and D. Berenson, "Human-Robot Collaborative Manipulation Planning Using Early Prediction of Human Motion," in *International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, 2013.

[2] M. Rüßmann, M. Lorenz, P. Gerbert, M. Waldner, J. Justus, . P. Engel and M. Harnisch, "Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries," 2015.

[3] G. Ferrer, A. Garrell, M. Villamizar, I. Huerta and A. Sanfeliu, "Robot Interactive learning through Human Assistance," *Multimodal Interaction in Image and Video Applications. Intelligent Systems Reference Library,* vol. 48, pp. 185-203, 2013.

[4] T. Fong, C. Thorpe and C. Baur, "Collaborative Control: A Robot-Centric Model for Vehicle Teleoperation," AAAI Technical Report SS-99-06, 1999.

[5] T. Fong, C. Thorpe and C. Baur, "Multi-Robot Remote Driving With Collaborative Control," *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS,* vol. 50, no. 4, pp. 699-704, 2003.

[6] S. Kock, T. Vittor, B. Matthias, H. Jerregard and M. Källman, "Robot concept for scalable, flexible assembly automation: A technology study on a harmless dual-armed robot," in *IEEE International Symposium on Assembly and Manufacturing (ISAM)*, Tampere, 2011.

[7] B. Matthias, S. Kock, H. Jerregard, M. Kallman, I. Lundberg and R. Mellander, "Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept," in *IEEE International Symposium on Assembly and Manufacturing (ISAM)*, Tampere, 2011.

[8] A. Ude, C. G. Atkeson and M. Riley, "Programming full-body movements for humanoid robots by observation," *Robotics and Autonomous Systems,* vol. 47, pp. 93-108, 2004.

[9] L. Cheng, Q. Sun, H. Su, Y. Cong and S. Zhao, "Design and implementation of human-robot interactive demonstration system based on Kinect," in *24th Chinese Control and Decision Conference (CCDC)*, Taiyuan, 2012.

[10] K. Lai, J. Konrad and P. Ishwar, "A gesture-driven computer interface using Kinect," IEEE Southwest Symposium on Image Analysis and Interpretation, Santa Fe, 2012.

[11] H. Liu and L. Wang, "An AR-based Worker Support System for Human-Robot Collaboration," in *27th International Conference on Flexible Automation and Intelligent Manufacturing*, Modena, 2017.

[12] F. J. Abu-Dakka, L. Rozo and D. G. Caldwell, "Force-based variable impedance learning for robotic manipulation," *Robotics and Autonomous Systems,* vol. 109., pp. 156-167, 2018.

[13] M. U. Suleman and M. M. Awais, "Learning from demonstration in robots: Experimental comparison of neural architectures," *Robotics and Computer-Integrated Manufacturing,* vol. 27, no. 4, pp. 794-801, 2010.

[14] "Automated assembly skill acquisition and implementation through human demonstration," *Robotics and Autonomous Systems,* vol. 99, pp. 1-16, 2018.

[15] L. Zhao, X. Li, P. Liang, C. Yang and R. Li, "Intuitive Robot Teaching by Hand Guided Demonstration," in *International Conference on Mechatronics and Automation*, Harbin, 2016.

[16] T. Fong, C. Thorpe and C. Baur, "Multi-robot remote driving with collaborative control," *IEEE Transactions on Industrial Electronics ,* vol. 50, no. 4, pp. 699-704, 2003.

[17] B. Matthias, S. Kock, H. Jerregard, M. Kallman, I. Lundberg and R. Mellander, "Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept," in *2011 IEEE International Symposium on Assembly and Manufacturing* , Tampere, 2011.

[18] ABB Robotics, *Product specification IRB14000,* Västerås: ABB, 2018.

[19] ABB Robotics, "YuMi® - Creating an automated future together.," [Online]. Available: https://new.abb.com/products/robotics/industrial-robots/irb-14000-yumi. [Accessed 26 04 2019].

[20] ABB Robotics, "ABB FlexPendant plastic software," ABB, [Online]. Available: https://new.abb.com/products/robotics/application-software/plastic. [Accessed 03 05 2019].

[21] J. F. Nunes, M. M. Pedro and M. João , "Human Motion Analysis and Simulation Tools: A Survey," in *Handbook of Research on Computational Simulation and Modeling in Engineering.* , IGI Global, 2016, pp. 359-388.

[22] S. Giancola, M. Valenti and R. Sala, "State-of-the-Art Devices Comparison," in *A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies*, Milan, Springer, 2018, pp. 29-39.

[23] Intel Corporation, "Intel® RealSense D400 Series Product Family Datasheet," Intel, 2019.

[24] Intel Corporation, "Intel RealSense Technology," Intel, [Online]. Available: https://software.intel.com/en-us/realsense/d400. [Accessed 26 04 2019].

[25] NuiTrack, "3DiVi Nuitrack 3D Skeleton Tracking Middleware," Nuitrack, [Online]. Available: http://download.3divi.com/Nuitrack/doc/index.html. [Accessed 02 05 2019].

[26] NuiTack, "Nuitack SDK," [Online]. Available: https://nuitrack.com. [Accessed 29 04 2019].

[27] ABB Robotics, "RobotStudio," ABB, [Online]. Available: https://new.abb.com/products/robotics/robotstudio. [Accessed 29 04 2018].

[28] Intel, "USB2 Support for Intel® RealSense™ Technology," Intel Realsense, [Online]. Available: https://www.intelrealsense.com/usb2-support-for-intel-realsense-technology/. [Accessed 29 04 2018].

[29] Military Disability, "The Shoulder and Upper Arm Muscles," 2003. [Online]. Available: http://www.militarydisabilitymadeeasy.com/. [Accessed 09 04 2019].

[30] ABB Robotics, Application manual Motion functions and events, ABB, 2011.

[31] T. T. Andersen, H. B. Amor, N. A. Andersen and O. Ravn, "Measuring and Modelling Delays in Robot Manipulators for Temporally Precise Control using Machine Learning.," in *Proceedings of IEEE ICMLA'15*, 2015.

[32] ABB Robotics, "Product Range For the flexible and efficient Factory of the Future," p. 66.

[33] L. S. Sterling, The Art of Agent-Oriented Modeling, London: The MIT Press, 2009.

APPENDICES

Appendix 1 – Programming code for Unity

Depth camera acquisition

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class RiggedAvatar : MonoBehaviour
{
    [Header("Rigged model")]
    [SerializeField]
    ModelJoint[] modelJoints;
    //public Text calibrationText = null;
    bool calibration = false;
    private FileTcp FileTcp;
    string userFound = "";
    string calibText = "";

    /// <summary> Model bones </summary>
    Dictionary<nuitrack.JointType, ModelJoint> jointsRigged = new
Dictionary<nuitrack.JointType, ModelJoint>();

    void Start()
    {

        //calibrationText.enabled = false;

        for (int i = 0; i < modelJoints.Length; i++)
        {
            modelJoints[i].baseRotOffset = modelJoints[i].bone.rotation;
            jointsRigged.Add(modelJoints[i].jointType, modelJoints[i]);
        }

        FileTcp = FindObjectOfType<FileTcp>();
    }

    void Update()
    {

        if (CurrentUserTracker.CurrentSkeleton != null)
        {
            ProcessSkeleton(CurrentUserTracker.CurrentSkeleton);
            userFound = "User found";
        }
        else
        {
            userFound = "User Not Found";
        }
```

```
    }

    void ProcessSkeleton(nuitrack.Skeleton skeleton)
    {
        //Calculate the model position: take the Torso position and invert movement
along the Z axis
        Vector3 torsoPos = Quaternion.Euler(0f, 180f, 0f) * (0.001f *
skeleton.GetJoint(nuitrack.JointType.Torso).ToVector3());
        transform.position = torsoPos;

        foreach (var riggedJoint in jointsRigged)
        {
            //Get joint from the Nuitrack
            nuitrack.Joint joint = skeleton.GetJoint(riggedJoint.Key);

            ModelJoint modelJoint = riggedJoint.Value;

            //Calculate the model bone rotation: take the mirrored joint
orientation, add a basic rotation of the model bone, invert movement along the Z
axis
            Quaternion jointOrient =
Quaternion.Inverse(CalibrationInfo.SensorOrientation) *
(joint.ToQuaternionMirrored()) * modelJoint.baseRotOffset;
            modelJoint.bone.rotation = jointOrient;

        }

        if (calibration == false)
        {
            calibration =
CalibratingHand(CurrentUserTracker.CurrentSkeleton.GetJoint(nuitrack.JointType.LeftH
and));
        }
        else
        {
            //Left Hand
            int[] leftHomePosition = { -150, -130, 110 };
            string leftFinalString = ArmMovement(nuitrack.JointType.LeftHand ,
leftHomePosition);

            //RightHand
            int[] rightHomePosition = { 150, -130, 110 };
            string rightFinalString = ArmMovement(nuitrack.JointType.RightHand,
rightHomePosition);

            if(ButtonContorl.dataActive==true)
            {
                FileTcp.SendTcpMessage("S"+ButtonContorl.signalCode+"," +
leftFinalString+","+rightFinalString+"E");
                WriteToFile("S" + "," + ButtonContorl.signalCode + "," +
leftFinalString + "," + rightFinalString + ","+ "E");
            }

        }


    }

    private string ArmMovement(nuitrack.JointType joint, int[] homePos)
    {
        StringBuilder jointsString = new StringBuilder(200);
```

```
        nuitrack.Joint handL = CurrentUserTracker.CurrentSkeleton.GetJoint(joint);
        nuitrack.Joint waist =
CurrentUserTracker.CurrentSkeleton.GetJoint(nuitrack.JointType.Waist);

        //  Debug.Log(handL.ToQuaternionMirrored().eulerAngles);

        //double homePositionX = -150; // -182.61;
        //double homePositionY = -130; // -9.58;
        //double homePositionZ = 110; // 198.63;

        double initialPositionX = (waist.Real.X - handL.Real.X) - homePos[0];
        double initialPositionY = (waist.Real.Y - handL.Real.Y) - homePos[1];
        double initialPositionZ = (waist.Real.Z - handL.Real.Z) - homePos[2];

        jointsString.Append(initialPositionX.ToString("+000.00;-000.00;+0") + "," +
initialPositionY.ToString("+000.00;-000.00;+0") + "," +
initialPositionZ.ToString("+000.00;-000.00;+0"));


       // Debug.Log(initialPositionX.ToString("+000.00;-000.00;+0") + "," +
initialPositionY.ToString("+000.00;-000.00;+0") + "," +
initialPositionZ.ToString("+000.00;-000.00;+0"));

        string finalString = jointsString.ToString();

        return finalString;

    }

    private void WriteToFile(string fileString)
    {
        StreamWriter sw = new StreamWriter("D:\\Test.txt", true);
        sw.WriteLine(fileString);
        sw.Close();

    }

    private bool CalibratingHand(nuitrack.Joint joint)
    {
        bool calibrationDone = false;
        //calibrationText.enabled = true;
        //calibrationText.text = "Move your hands for T position for Calibration";
        calibText = "Move your hands for T position for Calibration";
        if (joint.ToQuaternionMirrored().eulerAngles.z > 0 &&
joint.ToQuaternionMirrored().eulerAngles.z < 25 ||
joint.ToQuaternionMirrored().eulerAngles.z > 340 &&
joint.ToQuaternionMirrored().eulerAngles.z < 360)
        {
            var startTime = DateTime.UtcNow;

            while (DateTime.UtcNow - startTime < TimeSpan.FromSeconds(3))
            {
                //calibrationText.text = "Hold the position";
                calibrationDone = true;
                Debug.Log("Calibration done");
                calibText = "Calibration done";
            }
        }
        else
        {

        }
```

```
        //calibrationText.enabled = false;
        return calibrationDone;
    }

    float ChangeAngle(float angle)
    {
        float angleChange;

        if (angle > 180)
        {
            angleChange = angle - 360;
        }
        else
        {
            angleChange = angle;
        }
        return angleChange;
    }

    private void OnGUI()
    {
        //GUI.color = Color.red;
        //GUI.skin.label.fontSize = 35;

        //GUILayout.Label(userFound);

        GUIStyle user = new GUIStyle();
        user.fontSize = 35;
        GUI.color = Color.red;
        GUI.Label(new Rect(30, 100, 100, 20), userFound, user);


        GUIStyle caliStyle = new GUIStyle();
        caliStyle.fontSize = 25;
        GUI.Label(new Rect(30, 140, 140, 20), calibText, caliStyle);
    }

}
```

TCP client protocol.

```
using UnityEngine;
using System.Collections.Generic;
using System.IO;
using System.Threading;
using System;
using System.Net;
using System.Net.Sockets;
using System.Text;

public class FileTcp : MonoBehaviour
{
    #region private members
    private TcpClient socketConnection;
private Thread clientReceiveThread;
    #endregion

    public static string receivedMsg = null;

    // Start is called before the first frame update
    void Start()
    {
    ConnectToTcpServer();
    }

    // Update is called once per frame
    void Update()
    {

    }

    //TCP Connection
    /// <summary>
    /// Setup socket connection.
    /// </summary>
    private void ConnectToTcpServer()
    {
        try
        {
            clientReceiveThread = new Thread(new ThreadStart(ListenForData));
            clientReceiveThread.IsBackground = true;
            clientReceiveThread.Start();
        }
        catch (Exception e)
        {
            Debug.Log("On client connect exception " + e);
        }
    }
    /// <summary>
    /// Runs in background clientReceiveThread; Listens for incomming data.
    /// </summary>
    private void ListenForData()
    {
        try
        {
            socketConnection = new TcpClient("localhost", 55555);
            Byte[] bytes = new Byte[1024];
            while (true)
            {
                // Get a stream object for reading
```

```
                using (NetworkStream stream = socketConnection.GetStream())
                {
                    int length;
                    // Read incomming stream into byte arrary.

                    while ((length = stream.Read(bytes, 0, bytes.Length)) != 0)
                    {
                        var incommingData = new byte[length];
                        Array.Copy(bytes, 0, incommingData, 0, length);
                        // Convert byte array to string message.

                        string serverMessage =
Encoding.ASCII.GetString(incommingData);
                        Debug.Log("server message received as: " + serverMessage);
                        receivedMsg = serverMessage;
                    }
                }
            }
        }
        catch (SocketException socketException)
        {
            Debug.Log("Socket exception: " + socketException);

        }
    }
    /// <summary>
    /// Send message to server using socket connection.
    /// </summary>
    public void SendTcpMessage(string clientMessage)
    {
        if (socketConnection == null)
        {
            return;
        }
        try
        {
            // Get a stream object for writing.
            NetworkStream stream = socketConnection.GetStream();
            if (stream.CanWrite)
            {
                //string clientMessage = "This is a message from one of your
clients.";
                // Convert string message to byte array.
                byte[] clientMessageAsByteArray =
Encoding.ASCII.GetBytes(clientMessage);
                // Write byte array to socketConnection stream.
                stream.Write(clientMessageAsByteArray, 0,
clientMessageAsByteArray.Length);

                Debug.Log("Client sent his message - should be received by server");
            }
        }
        catch (SocketException socketException)
        {
            Debug.Log("Socket exception: " + socketException);
        }
    }
}
```

User interface design

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class ButtonContorl : MonoBehaviour
{
    public static string signalCode;
    public static bool dataActive;
    public Button startButton;
    public Button stopButton;
    public Button HomePositionButton;
    public Toggle leftArmToggle;
    public Toggle rightArmToggle;

    private FileTcp FileTcp;


    // Start is called before the first frame update
    void Start()
    {
        signalCode = null;
        leftArmToggle.isOn = false;
        rightArmToggle.isOn = false;
        dataActive = false;
        FileTcp = FindObjectOfType<FileTcp>();
    }

    // Update is called once per frame
    void Update()
    {
        startButton.onClick.AddListener(() => BtnPrs("05"));
        stopButton.onClick.AddListener(() => BtnPrs("06"));
        HomePositionButton.onClick.AddListener(() => BtnPrs("04"));

        if (leftArmToggle.isOn==true && rightArmToggle.isOn == false)
        {
            BtnPrs("01");
        }
        else if (rightArmToggle.isOn == true && leftArmToggle.isOn == false)
        {
            BtnPrs("02");
        }
        else if (leftArmToggle.isOn == true && rightArmToggle.isOn == true)
        {
            BtnPrs("03");
        }
        else
        {       leftArmToggle.isOn = false;
                rightArmToggle.isOn = false;
        }    }
    private void BtnPrs(string value)
    {
        signalCode = value;
       // FileTcp.SendTcpMessage(("S" + signalCode +"E").ToString());
        Debug.Log(value + " Button Pressed");
        dataActive = true;
    }
}
```

Right hand code

```
MODULE Module1

    !***********************************************************
    !Module:  Module1    !
    ! Description:
    !   <Insert description here>
    !Author: damara    !
    ! Version: 1.0    !
    !***********************************************************
    !***********************************************************
    ! Procedure main    !
    !   This is the entry point of your program    !
    !***********************************************************

    !Variables for Socket
    VAR socketdev server;
    VAR socketdev client;
    CONST string ServerIp:="127.0.0.1";
    CONST num ServerPort:=55555;
    LOCAL VAR string clientIp;
    VAR rawbytes data;
    !VAR String data;
    VAR bool ok;
    VAR num noOfBytes;

    !Variables for  and unpack
    VAR num found{9}:=[0,0,0,0,0,0,0,0,0];
    VAR string startHeader:="";
    VAR string endHeader:="";
    VAR string controlMsg:="";
    VAR string waist;
    VAR string leftHand{3};
    VAR string rightHand{3};

    VAR num leftHandNum{3};
    PERS num rightHandNum{3};

    PERS bool leftHandOn;
    PERS bool leftHome;

    VAR num userProm;
    VAR num fileRepeat;
    VAR num fileName;
    VAR string fileNameS;
    VAR num xAxisLimit:=600;
```

```
VAR num yAxisLimit:=300;
VAR num zAxisLimit:=0;

!Variables for Read
VAR string controlMsgRead;
VAR string leftHandRead{3};
VAR string rightHandRead{3};

VAR num leftHandNumRead{3};
PERS num rightHandNumRead{3};
VAR string leftSide;
VAR string rightSide;

PERS string modeSelect:="PLAY";

VAR jointtarget rotJ1_X:=[[0,-130,30,0,40,0],[-135,9E9,9E9,9E9,9E9,9E9]];
!home position
VAR robtarget incrementTarget;
VAR num absoluteposition{7}:=[6,6,6,6,6,6,6];
VAR num minTrans:=5;
VAR pos initialPosition;
VAR pos incrementPosition;
VAR robtarget initialTarget;
VAR robtarget variableTarget;
VAR tooldata initialTool;
VAR tooldata variableTool;

!Variables for file
VAR string FileLocation:="HOME:test.csv";
VAR iodev File;

VAR num numberOfBytes;

PROC main()

    !Initial Home Position
    TPErase;
    MoveAbsJ rotJ1_X,\NoEOffs,v1000\T:=3,z100,tool0\WObj:=wobj0;
    initialPosition:=CPos();
    initialTarget:=CRobT(\Tool:=tool0);
    initialTool:=ctool();

    UserPrompt;

    IF modeSelect="RECORD" THEN

        Open FileLocation,File\Write;
        !Clear the file
        Close File;
        ServerCreate;
```

```
        WHILE (true) DO

            UnpackBytes;
            FOR i FROM 1 TO 3 DO
                ok:=StrToVal(leftHand{i},leftHandNum{i});
                ok:=StrToVal(rightHand{i},rightHandNum{i});
            ENDFOR
            SelectFunc controlMsg,leftHandNum{1},leftHandNum{2},leftHandNum{3};
            WriteData;
        ENDWHILE
        ServerClose;

    ELSEIF modeSelect="PLAY" THEN
        Open FileLocation,File\Read;
        WHILE (TRUE) DO
            ReadData;

            FOR i FROM 1 TO 3 DO
                ok:=StrToVal(leftHandRead{i},leftHandNumRead{i});
                ok:=StrToVal(rightHandRead{i},rightHandNumRead{i});
            ENDFOR
            SelectFunc
controlMsgRead,leftHandNumRead{1},leftHandNumRead{2},leftHandNumRead{3};

            IF fileRepeat=2 AND controlMsgRead="EOF" THEN
                TPWrite "End of File Reached";
                Break;
            ELSEIF fileRepeat=1 AND controlMsgRead="EOF" THEN
                SelectFunc
"04",leftHandNumRead{1},leftHandNumRead{2},leftHandNumRead{3};
                !Home Position
                Rewind File;
            ENDIF

        ENDWHILE
        Close File;
    ELSE
    ENDIF
ENDPROC

PROC convertAngles(num angle,num angle1)


    IF angle>180 THEN
        angle1:=angle-360;
        RETURN ;
    ELSE
        angle1:=angle;
        RETURN ;
```

```
    ENDIF

  ENDPROC

  PROC MoveRightArm(num xcord,num ycord,num zcord)

    !Match Home position to cordinates
    !X axis of the robot it -Y of the sensor
    variableTarget.trans.x:=-ycord;

    !Y axis of the robot equal to X to the sensor
    variableTarget.trans.y:=-xcord;

    !Z axis of the robot is equal to Z of the sensor
    variableTarget.trans.z:=zcord;

    IF xcord>-xAxisLimit AND ycord>-yAxisLimit THEN
      IF Abs(absoluteposition{1})>minTrans AND Abs(absoluteposition{2})>minTrans
AND Abs(absoluteposition{3})>minTrans THEN

        MoveJ
RelTool(initialTarget,variableTarget.trans.x,variableTarget.trans.y,variableTarget.trans.z),
v1000,z30,tool0;

      ENDIF
    ENDIF

    incrementTarget:=CRobT(\Tool:=tool0\WObj:=wobj0);
    variableTool:=CTool();
    absoluteposition{1}:=incrementTarget.trans.x-variableTarget.trans.x;
    absoluteposition{2}:=incrementTarget.trans.y-variableTarget.trans.y;
    absoluteposition{3}:=incrementTarget.trans.z-variableTarget.trans.z;

  ENDPROC

  PROC WriteData()
    Open FileLocation,File\Append;
    Write File,controlMsg+","\NoNewLine;
    Write File,"L,"+NumToStr(leftHandNum{1},2)+","\NoNewLine;
    Write File,NumToStr(leftHandNum{2},2)+","\NoNewLine;
    Write File,NumToStr(leftHandNum{3},2)+","\NoNewLine;
    Write File,"R,"+NumToStr(rightHandNum{1},2)+","\NoNewLine;
    Write File,NumToStr(rightHandNum{2},2)+","\NoNewLine;
    Write File,NumToStr(rightHandNum{3},2);
    Close File;
  ENDPROC

  PROC ReadData()
    !   Open FileLocation, File\Read;
    controlMsgRead:=ReadStr(File\Delim:=",");
```

```
        leftSide:=ReadStr(File\Delim:=",");
        leftHandRead{1}:=ReadStr(File\Delim:=",");
        leftHandRead{2}:=ReadStr(File\Delim:=",");
        leftHandRead{3}:=ReadStr(File\Delim:=",");
        rightSide:=ReadStr(File\Delim:=",");
        rightHandRead{1}:=ReadStr(File\Delim:=",");
        rightHandRead{2}:=ReadStr(File\Delim:=",");
        rightHandRead{3}:=ReadStr(File\Delim:=","\RemoveCR);
        !  Close File;

    ENDPROC

    PROC UserPrompt()
        TPReadFK userProm,"Cobot type Robot Learning System........"+"Enter your
functionality",stEmpty,stEmpty,"Delete File","Play","Recording";
        IF userProm=5 THEN
            TPWrite "Select the a number for the file name (Only numbers are allowed) ";
            TPReadNum fileName,"Enter 3 digits for the file name";
            FileLocation:="HOME:CobotLearn"+NumToStr(fileName,0)+".csv";
            modeSelect:="RECORD";

        ELSEIF userProm=4 THEN
            TPWrite "Select the a number for the file name (Only numbers are allowed) ";
            TPReadNum fileName,"Enter 3 digits for the file name";
            FileLocation:="HOME:CobotLearn"+NumToStr(fileName,0)+".csv";
            TPWrite "Countinous of single play mode";
            TPReadNum fileRepeat,"Enter 1 for countinous play or 2 for single play";
            modeSelect:="PLAY";

        ELSEIF userProm=3 THEN
            TPWrite "Select the a number for the file name (Only numbers are allowed) ";
            TPReadNum fileName,"Enter 3 digits for the file name";
            RemoveFile "HOME:CobotLearn"+NumToStr(fileName,0)+".csv";

        ELSE
    ENDIF

ENDPROC

PROC UnpackBytes()
    SocketReceive client,\RawData:=data,\NoRecBytes:=noOfBytes;

    IF noOfBytes>100 THEN


    ELSEIF noOfBytes=52 THEN

        UnpackRawBytes data,1,startHeader,\ASCII:=1;
        UnpackRawBytes data,2,controlMsg,\ASCII:=2;
        UnpackRawBytes data,5,leftHand{1},\ASCII:=7;
```

```
        UnpackRawBytes data,13,leftHand{2},\ASCII:=7;
        UnpackRawBytes data,21,leftHand{3},\ASCII:=7;

        UnpackRawBytes data,29,rightHand{1},\ASCII:=7;
        UnpackRawBytes data,37,rightHand{2},\ASCII:=7;
        UnpackRawBytes data,45,rightHand{3},\ASCII:=7;

        UnpackRawBytes data,52,endHeader,\ASCII:=1;

        ClearRawBytes data;

    ENDIF

ERROR
    IF ERRNO=ERR_SOCK_CLOSED THEN
        ServerCreate;
        RETRY;
    ELSEIF ERRNO=ERR_SOCK_TIMEOUT THEN
        TPWrite "ReadStream() timeout. Retrying.";
        RETRY;
    ENDIF

ENDPROC

PROC SocketInitialize()
    <SMT>
ENDPROC

PROC ServerCreate()
    SocketClose server;
    SocketClose client;

    SocketCreate server;
    SocketBind server,ServerIp,ServerPort;
    SocketListen server;
    TPWrite "Waiting for incoming connection from
"+ServerIp+":"+NumToStr(ServerPort,0);

    SocketAccept server,client\ClientAddress:=clientIp\Time:=WAIT_MAX;

    TPWrite "Connected to client"+clientIp;

    SocketSend client\Str:="Connect to"+ServerIp;

ERROR
    IF ERRNO=ERR_SOCK_TIMEOUT THEN
        TPWrite "ERROR: ServerRecover timeout. Retrying...";
        RETRY;
    ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
        RETURN ;
```

```
    ELSE
        ! No error recovery handling
    ENDIF

ENDPROC

PROC ServerClose()
    SocketClose server;
    SocketClose client;
    WaitTime 2;
ENDPROC

PROC SelectFunc(string function,num xcord,num ycord,num zcord)
    IF function="01" THEN
        !Left hand movements only
        leftHandOn:=TRUE;

    ELSEIF function="02" THEN
        !Right hand movements
        MoveRightArm xcord,ycord,zcord;

    ELSEIF function="03" THEN
        !Move both hands
        leftHandOn:=TRUE;
        leftHome:=FALSE;
        MoveRightArm xcord,ycord,zcord;

    ELSEIF function="04" THEN
        !Home position
        MoveAbsJ rotJ1_X,\NoEOffs,v1000\T:=3,z100,tool0\WObj:=wobj0;
        leftHandOn:=false;
        leftHome:=TRUE;

    ELSEIF function="05" THEN
        !Start

    ELSEIF function="06" THEN
        !Stop

    ELSEIF function="07" THEN
        !Reset

    ELSE
    ENDIF
ENDPROC
ENDMODULE
```

Left Hand Code

```
MODULE Module1

    PERS num rightHandNum{3};
    PERS num rightHandNumRead{3};

    VAR jointtarget rotJ1_X:=[[0,-130,30,0,40,0],[135,9E9,9E9,9E9,9E9,9E9]];

    VAR robtarget incrementTarget;
    VAR num absoluteposition{7}:=[6,6,6,6,6,6,6];
    VAR num minTrans:=5;
    VAR pos initialPosition;
    VAR robtarget initialTarget;
    VAR tooldata initialTool;

    PERS bool leftHandOn;
    PERS bool leftHome;

    VAR num xAxisLimit:=300;
    VAR num yAxisLimit:=75;
    VAR num zAxisLimit:=0;

    PERS string modeSelect:="PLAY";

    VAR robtarget variableTarget;

    PROC main()
        !Set Home position data
        leftHandOn:=FALSE;
        MoveAbsJ rotJ1_X,\NoEOffs,v1000\T:=3,z100,tool0\WObj:=wobj0;
        initialPosition:=CPos();
        initialTarget:=CRobT(\Tool:=tool0);
        initialTool:=ctool();

        IF modeSelect="RECORD" THEN

            WHILE TRUE DO
                MoveLeftArm rightHandNum{1},rightHandNum{2},rightHandNum{3};
            ENDWHILE

        ELSEIF modeSelect="PLAY" THEN
            WHILE TRUE DO
                MoveLeftArm
rightHandNumRead{1},rightHandNumRead{2},rightHandNumRead{3};
            ENDWHILE
        ELSE
        ENDIF
    ENDPROC
```

```
  PROC MoveLeftArm(num xcord,num ycord,num zcord)
    IF leftHandOn=TRUE THEN

      MoveLeftHand xcord,ycord,zcord;

      leftHandOn:=FALSE;

    ELSEIF leftHome=TRUE THEN
      MoveAbsJ rotJ1_X,\NoEOffs,v1000\T:=3,z100,tool0\WObj:=wobj0;
      leftHome:=FALSE;

    ELSE
    ENDIF
  ENDPROC

  PROC MoveLeftHand(num xcord,num ycord,num zcord)
    !X axis of the robot it -Y of the sensor
    variableTarget.trans.x:=-ycord;

    !Y axis of the robot equal to X to the sensor
    variableTarget.trans.y:=-xcord;

    !Z axis of the robot is equal to Z of the sensor
    variableTarget.trans.z:=zcord;

    IF xcord<xAxisLimit AND ycord>-yAxisLimit THEN
      IF Abs(absoluteposition{1})>minTrans AND Abs(absoluteposition{2})>minTrans
AND Abs(absoluteposition{3})>minTrans THEN

        MoveJ
RelTool(initialTarget,variableTarget.trans.x,variableTarget.trans.y,variableTarget.trans.z),
v1000,z30,tool0;

      ENDIF
    ENDIF

    incrementTarget:=CRobT(\Tool:=tool0\WObj:=wobj0);

    absoluteposition{1}:=incrementTarget.trans.x-variableTarget.trans.x;
    absoluteposition{2}:=incrementTarget.trans.y-variableTarget.trans.y;
    absoluteposition{3}:=incrementTarget.trans.z-variableTarget.trans.z;
  ENDPROC
ENDMODULE
```

Appendix 3 Accuracy test

The distances moved along the X,Y and Z axis during the accuracy test. The explanation is available in section 4.7 of the report.