Chalmers University of Technology

Humanoid Robotics
TIF160

Hyoubert

Authors:
Ebaa Asaad **Sara Larsson** **Jonatan Nordström** **Ria Raj**
October 27, 2021

Abstract

Teleoperation of a humanoid robot has many different applications ranging from providing access to hazardous environments to entertainment and edutainment. In this project, a program for controlling a stationary humanoid robot's manipulators, using a Virtual Reality headset, was developed. The scope of the project was limited to controlling a single arm and the torso of the robot. The success-measure was divided into four stages, where the first one was that the robot mimicked simple movements, the second was to place objects in a confined area, the third one was to place object in a confined area accurately and the fourth and final stage is an expansion of stage three with both placing functionality, and picking. The results showed that the robot succeeded with all four stages, however with some remarks. The remark that came with stage one included that although the robot reached the target point, the different anatomies of the operator and robot was visible. The errors that came from stage two, three and four were tightly linked to Hubert not having any visual feedback. So with the parameters given, the project was concluded a success.

Contents

1 Introduction

There are many reasons as to why a person would want to be able to teleoperate a robot. The most important reason is to be able to access and manipulate a dynamic environment where the presence of humans is not desired, i.e. safety. Another reason is to motivate children to pursue technological studies (Edutainment). One more reason would be to use it for entertainment. With many reasons to realize an implementation like this, this project aims to remotely control and operate (teleoperate) a humanoid robot using a Virtual Reality (VR) headset. The robot used in this project is Hubert 2.0 and is stationary and has three joints, torso, shoulder and elbow, as well as a gripper hand.

1.1 Goal Statement

We aim to use teleoperation with the help of a VR headset through mapping the controller's position to Hubert's manipulators. The operator should also be able to make Hubert grasp objects by pressing a button on the controller.

1.2 Scope

This project is limited to controlling one arm of the robot. The performance will be measured in the following four stages.

Stage 1: Simple movements The robot should be able to mimic simple, straight arm movements made by the operator. eg. raising the arm to the side. The performance will be measured by the difference in angle between the operators and robots arms.

Stage 2: Placing object in area A small item will be placed in the robots hand and the robot will need to place the object in a designated area. Performance is measured as success or failure to place the object in the area.

Stage 3: Placing object accurately The robot will need to place the item on a specified point, the performance will be measure by the distance between the center of the object and the designated point.

Stage 4: Picking and placing object The robot will now need to pick up the object before placing it on the point. The performance is measured as in Stage 3.

1.3 Contributions

The project was internally divided so that each member worked on different aspects. The members then briefed each other and combined all subdivided parts. As for the report, each member gave their contributions.

2 Theory

2.1 Virtual environment

In order to get the position of the operators hand, a virtual environment will need to be constructed. The virtual reality headset needs to be able to connect to this virtual environment to get the hand position of the right hand controller, i.e. the operators right hand. The environment has no visual requirements, the only requirement is that it has a world frame that the controllers position can be represented in.

The Unity game engine, also known as Unity Software, will be used to create the virtual environment as some of the authors have some previous experience with this tool. There is also a lot of online documentation and guides surrounding Unity Software and virtual reality.

2.2 Kinematics

When building a robot with several joints, it is crucial to determine an inverse kinematics (IK) model that gives the required motor configurations, i.e. the angles, to reach a specified target position with for example the gripper. This chapter will explain the theory behind the IK-model used for Hubert. Since the methods and models are problem specific, the resulting structure of Hubert in this project (shown in figure 8) is used throughout the theory chapter, where the lengths L_1 to L_7 are assumed to be known. Additionally, the forward kinematics model will be calculated for later testing purposes.

2.2.1 Forward kinematics

The solution for the inverse kinematics model was calculated in the simplest matter since the time frame of the project was short. Therefore, static equations were calculated specified for this setup of links and joints and no kinematics libraries were used. The resulting solution was to simplify the 3-dimensional space problem, by dividing it into two 2-dimensional problems. One problem that calculates the angle of the torso and one problem that calculates a combination of angles in the arm to reach the target position with the gripper.

Note that we are using U_x as a notation for all the unknown variables that are to be calculated.

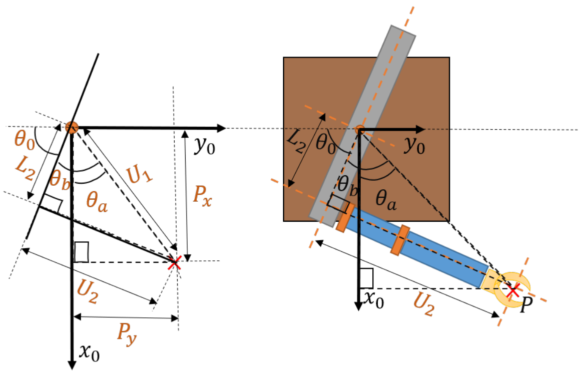
First 2D problem

This problem is calculated using only the x and y axis. To calculate the torso angle θ_0, we make use of the knowledge that the arm is always perpendicular to the shoulders since both joints are rotating in the same direction. This results in only one solution of θ_0 if we assume that it will only be able to reach positions in front of the shoulders. We can then draw perpendicular triangles as shown in figure 1a-1d, and calculate the angles θ_a and θ_b. These angles are then used to calculate θ_0. The equations follows:

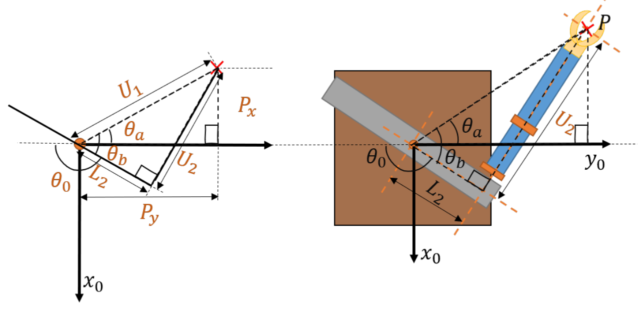
First quadrant, (figure 1a): $\theta_a = arctan(\lvert\frac{P_y}{P_x}\rvert)$ $U_1 = \sqrt{P_x^2 + P_y^2}$ $\theta_b = arccos\frac{L_2}{U_1}$ $\theta_0 = \frac{\pi}{2} + \theta_a - \theta_b$	Second quadrant, (figure 1b): $\theta_a = arctan(\lvert\frac{P_x}{P_y}\rvert)$ $U_1 = \sqrt{P_x^2 + P_y^2}$ $\theta_b = arccos\frac{L_2}{U_1}$ $\theta_0 = \pi + \theta_a - \theta_b$
Third quadrant, (figure 1c): $\theta_a = arctan(\lvert\frac{P_x}{P_y}\rvert)$ $U_1 = \sqrt{P_x^2 + P_y^2}$ $\theta_b = arccos\frac{L_2}{U_1}$ $\theta_0 = -\theta_a - \theta_b$	Fourth quadrant, (figure 1d): $\theta_a = arctan(\lvert\frac{P_y}{P_x}\rvert)$ $U_1 = \sqrt{P_x^2 + P_y^2}$ $\theta_b = arccos\frac{L_2}{U_1}$ $\theta_0 = \frac{\pi}{2} - (\theta_a + \theta_b)$

Here it is easy to further calculate the arm length seen from above U_2 that is used in the next 2D problem in 2.2.1. The equation follows:

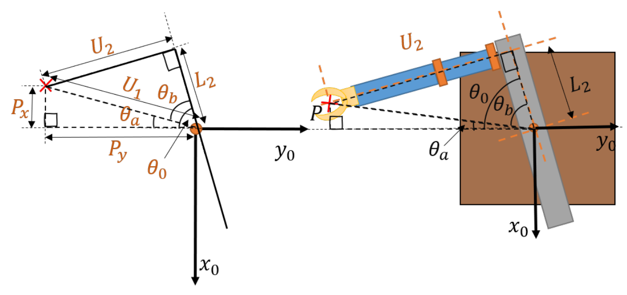
$$U_2 = \sqrt{U_1^2 - L_2^2} \tag{1}$$



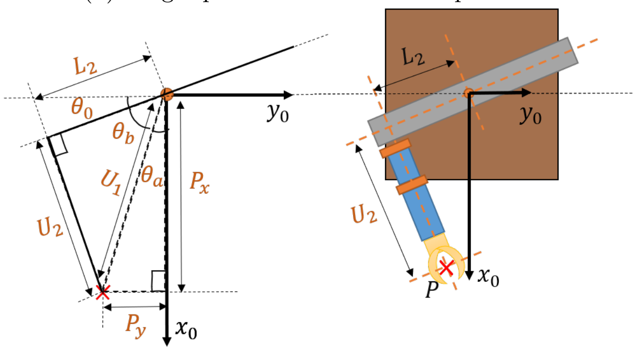
(a) Target position at the first quadrant.



(b) Target position at the second quadrant.



(c) Target position at the third quadrant.

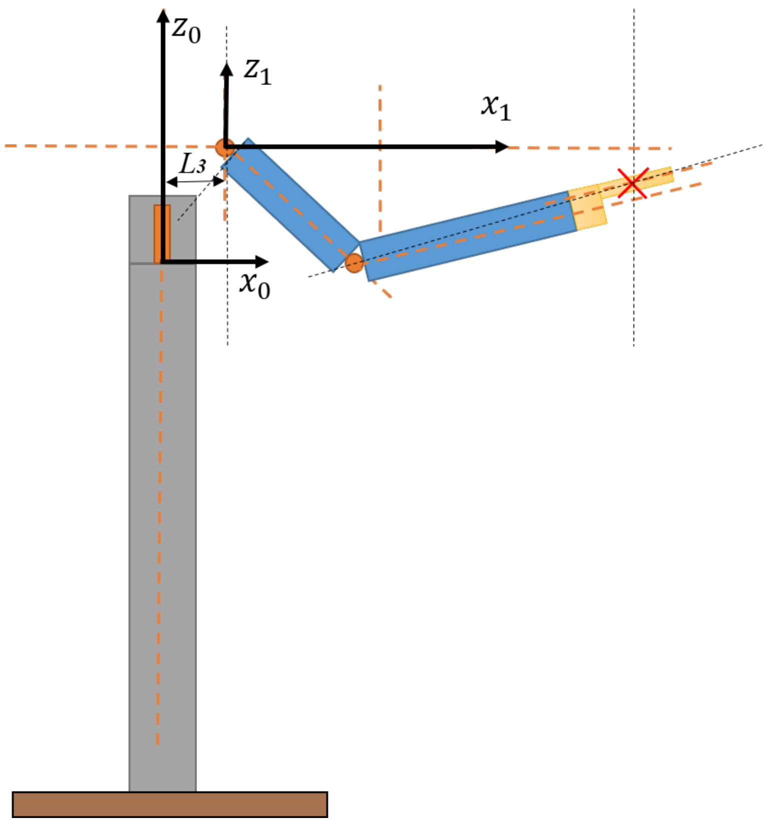


(d) Target position at the fourth quadrant.

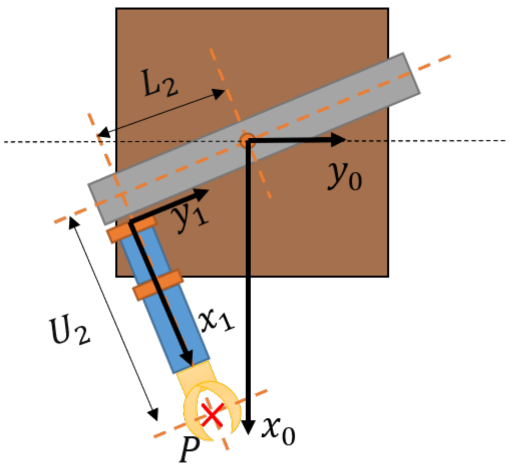
Figure 1: Sketches of Hubert reaching the target point in different quadrants seen from above. To the left in all 4 sub figures, a sketch of the different lengths are given, and to the right a corresponding sketch of the robot.

Second 2D problem

The second 2D problem is solved using a second coordinate system that is located at the upper arm joint shown in figure 2. This coordinate system follows the arm and therefore, the y-value of the position will always be 0, thus it becomes a 2D problem. The target position at the red cross in coordinate system 1 has now the position $P_1[x, z] = [U_2 - L_3, P_{0,z}]$, where U_2 is given in equation 1. The lower arm is also simplified according to figure 3 for easier calculations. The angles θ_1 and θ_2 are calculated using the variables used are defined in figure 4.



(a) Seen from the side.



(b) Seen from above.

Figure 2: Showing the position of coordinate system 1 compared to the base coordinate system 0.

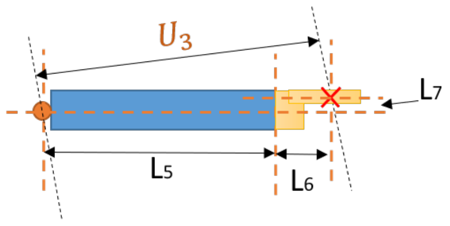


Figure 3: Simplification of lower arm using Pythagoras theorem: $U_3 = \sqrt{(L_5 + L_6)^2 + L_7^2}$

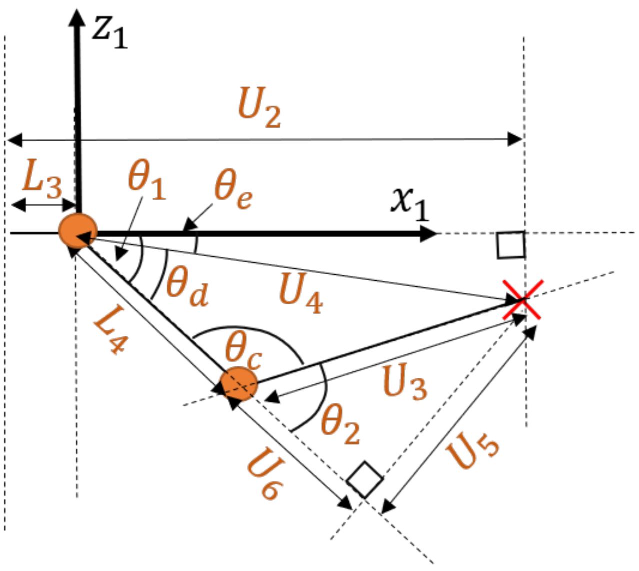


Figure 4: Sketch with all the lengths and variables required to calculate θ_1 and θ_2.

First θ_2 is calculated. We begin with determining U_4 using the Pythagorean theorem of the x and z coordinates, which together with the lengths L_4 and U_3 creates a triangle. By using the cosine rule on the triangle, θ_c can be calculated, which then can be used to calculate θ_2. The equations are summarized below. Since the inverse of cosine is symmetric to the origin, it will give two possible solutions. However, only the positive solution will be considered since the physical arm cannot move as much in the negative direction of θ_2 as in the positive.

$$U_4 = \sqrt{x_1^2 + z_1^2}$$
$$cos\theta_b = \frac{L_4^2 + U_3^2 - U_4^2}{2L_4U_3}$$
$$\theta_2 = \pi - \theta_b \implies cos\theta_2 = -cos\theta_b$$
$$\theta_2 = arccos\left(\frac{U_4^2 - L_4^2 - U_3^2}{2L_4U_3}\right)$$

The angle θ_1 is then calculated by first getting U_5 and U_6 by using trigonometric functions, θ_2 and U_3. Now it is possible to calculate both θ_d and θ_e using the two perpendicular triangles with a corner at the origin, shown in figure 4, which will give the resulting angle θ_1. Note that θ_e will have both negative and positive values, while θ_d will always be positive due to the solely positive span of θ_2.

$$U_5 = U_3 sin\theta_2$$
$$U_6 = U_3 cos\theta_2$$
$$\theta_d = arctan\left(\frac{U_5}{L_4 + U_6}\right)$$
$$\theta_e = arctan\left(\frac{z_1}{x_1}\right)$$
$$\theta_1 = \theta_e - \theta_d$$

2.2.2 Inverse Kinematics

The forward kinematics is required to test the correctness and range of the inverse kinematics derived in 2.2.2. By using transformation matrices between frames with origin at the body ends as shown in figure 5, then the location of the body parts would be known. This method origins from the project course content. The matrices are shown below.

Frame $a \rightarrow 0$: Rotation around z_0

$$\begin{pmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta_0 & -sin\theta_0 & 0 & 0 \\ sin\theta_0 & cos\theta_0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_a \\ y_a \\ z_a \\ 1 \end{pmatrix}$$

Frame $b \rightarrow a$: Translation along y_a

$$\begin{pmatrix} x_a \\ y_a \\ z_a \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -L_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_b \\ y_b \\ z_b \\ 1 \end{pmatrix}$$

Frame $1 \rightarrow b$: Translation along x_b

$$\begin{pmatrix} x_b \\ y_b \\ z_b \\ 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & L_3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix}$$

Frame $c \rightarrow 1$: Rotation around y_1, rotation of coordinate system around x_1 and translation along x_1

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta_1 & -sin\theta_1 & 0 & L_4cos\theta_1 \\ 0 & 1 & 0 & 0 \\ sin\theta_1 & cos\theta_1 & 0 & L_4sin\theta_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}$$

Frame $d \rightarrow c$: Rotation around z_c and translation along x_c

$$\begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix} = \begin{pmatrix} cos\theta_2 & -sin\theta_2 & 0 & U_3cos\theta_2 \\ sin\theta_2 & cos\theta_2 & 0 & U_3sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_d \\ y_d \\ z_d \\ 1 \end{pmatrix}$$

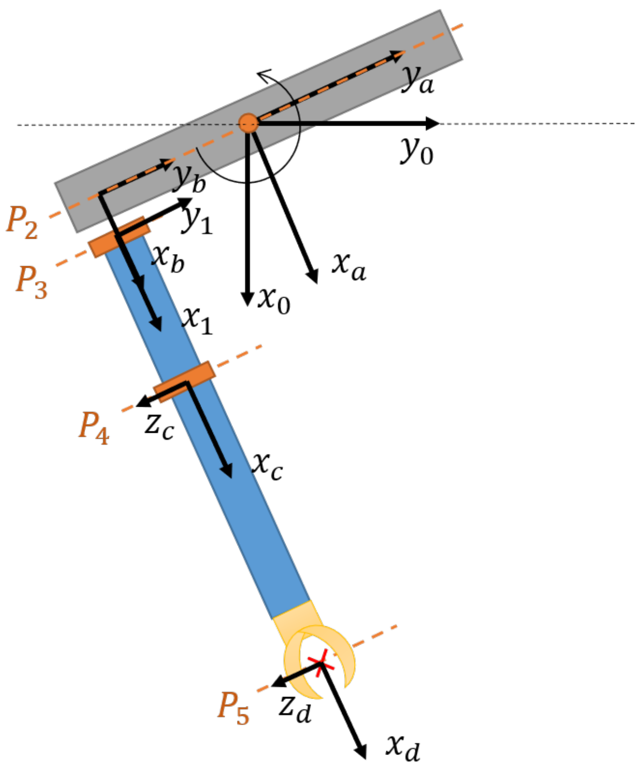


Figure 5: Sketch seen from above, with all frames used for the forward kinematics and points P_x that is the position of the ends of body parts in frame 0.

2.3 Angle conversions

Physical motors have other values for commanding angle-values and it is often specific for each motor. Therefore a conversion formula will solve this problem. Assuming that the max and min value for the motor is known and their corresponding angles, the relation of two angles θ (angle) and α(motor position value) can be determined. First step is to move the minimum value of θ and α to 0 by subtracting with the min values. Then normalize both by dividing with the difference of max and min. The relation is shown below as well as the resulting conversion from θ to α.

$$\frac{(\alpha - \alpha_{min})}{(\alpha_{max} - \alpha_{min})} = \frac{(\theta - \theta_{min})}{(\theta_{max} - \theta_{min})}$$
$$\Longleftrightarrow$$
$$\alpha = \frac{(\theta - \theta_{min}) * (\alpha_{max} - \alpha_{min})}{(\theta_{max} - \theta_{min})} + \alpha_{min} \tag{2}$$

3 Method

3.1 Hardware

The following hardware was used in the project.

Object	**Provider**	**Cost**
Hubert robot	Examiner	-
Power supply	Examiner	-
Arduino mega	Examiner	399:-
Computers	Team members	-
Oculus Quest VR-headset	Team members	-
A wifi router	Team members	-
Total		399:-

Table 1: Hardware used in the project and their cost.

3.2 Virtual environment

The virtual environment was created using the Unity Software ”Universal Render Pipeline” template. The setup of the Unity Software Project was done by following the guide by Justin P Barnet [1]. To allow the Unity Software to connect and interact with the Oculus Quest headset the OpenXR plugin 1.10.0 and the XR Interaction Toolkit 1.0.0-pre.5 was installed. The connection between the computer/Unity Software and the Oculus Quest headset is handled by the Oculus drivers and Oculus Air Link. This allows the Unity Software to stream the virtual environment to the headset via a shared WiFi network.

A GameObject with the name ”World Frame” was created to serve as the world frame for this project. Every object in the Unity Software is a GameObject and can hold many different components to serve different functions. These components determine the properties of the GameObject. All GameObjects have a Transform component, which gives it a location and orientation. The World Frames Transform will be used as the origin of the world frame that the right hand position will be represented in.

A script called PrimaryButtonWatcher was added to the GameObject World Frame, and a reference to the GameObject RightHand Controller was made. The RightHand Controller is a GameObject created by the XR Interaction Toolkit and is located at the position of the right hand controller. This script is a modified version of the ”Example for primaryButton” from the online Unity Manual [2].

By placing the script on the World Frame GameObject and referencing to the RightHand Controller in that script, the built in function GameObject.transform.position gets the position of the referenced GameObject in the frame of the current GameObject. When this function is called on the RightHand Controller from the World Frame the RightHand Controllers position in the World Frame coordinate system is returned. This position is then scaled to Hubert scale and sent to the inverse kinematics. The inverse kinematics outputs the three angles which is then packaged into a 12 long string and sent to the arduino using the SerialPort class. Where every three values represent one angle in degrees, i.e "010020030040" represent 10°, 20°, 30° and 40°, respectively.

The function calculating the inverse kinematics and sending the data to the Arduino is run every 100ms. This is to avoid over saturating the serial buffer. If the buffer gets full, the script has to stop and wait for the Arduino to clear the buffer. This leads to instability and eventually crashes the Unity Software. The updating frequency also affects how smooth the robot operates, which is why a low value is desired. Note that a frequency of 100ms is a little to high to guarantee long term stability, but performs well in a shorter time frame.

3.3 Arduino

The Arduino code is based on zoomkat's code from the Arduino forum [3]. While the program is not updating the servo values it is continually looking for a new string on the Serial port. This is to avoid missing when a string is received in the serial buffer and avoid the over saturation issue mentioned in the previous chapter. If a string is found in the serial buffer the string is read and slit into four separate integers, each representing the angle of a different servomotor.

Every 5ms the program updates the values of the servomotors using an Exponentially Weighted Moving Average (EWMA). The EWMA is a simple smoothing filter which implements the following function.

$$s_t = \alpha x_t + (1 - \alpha)s_{t-1} \tag{3}$$

Where the x_t is the current goal for the angle, the s_t is the smoothed value and s_{t-1} is the previous smoothed value. The α is the smoothing factor and was experimentally determined to be 0.05.

3.4 Inverse kinematics

In order to determine the joint angles, an IK- model was created according to chapter 2.2.2. To easier debug, evaluate and to avoid giving Hubert any invalid motor positions, a python test program was created, which is described in 3.4.1.

The same code for the IK-model was then translated to C# in order to be applicable in Unity. In addition it also converted the angles to the physical motor command angles using the conversion formula given in equation 2. Similarly as the python code it first restricted the θ angles to $\pm\pi 3/4$, but it also restricts the motor angles to the real physical max and min values. If the angles were outside the restrictions or had nan values, then theta would take the previous value instead.

3.4.1 Testing IK-model

To test the correctness and range of the calculated IK-model a python script was created to draw the robot in a 3D-plot for a given position, which is illustrated in figure 6. The positions are given by the slides shown to the left, and the resulting angles are shown to the right. The python program "model.py" is found in the Github repository for this project. The repository also includes a demo of the user interface and a short explanation of the code. Note that only the sliders for the position is changeable by the user, which is explained more in the repository. Following steps are carried out every time a new position is given:

- A combination of angles are calculated using the IK-model from chapter 2.2.2

- The forward kinematics derived in chapter 2.2.2 is then used to get the position of the different body parts (the points used to plot the body in python are illustrated by figure 5)
- The body parts and all values are updated in the plot
- If any of the angles are out of range or has invalid values, it will be written "Out of range!" on the plot

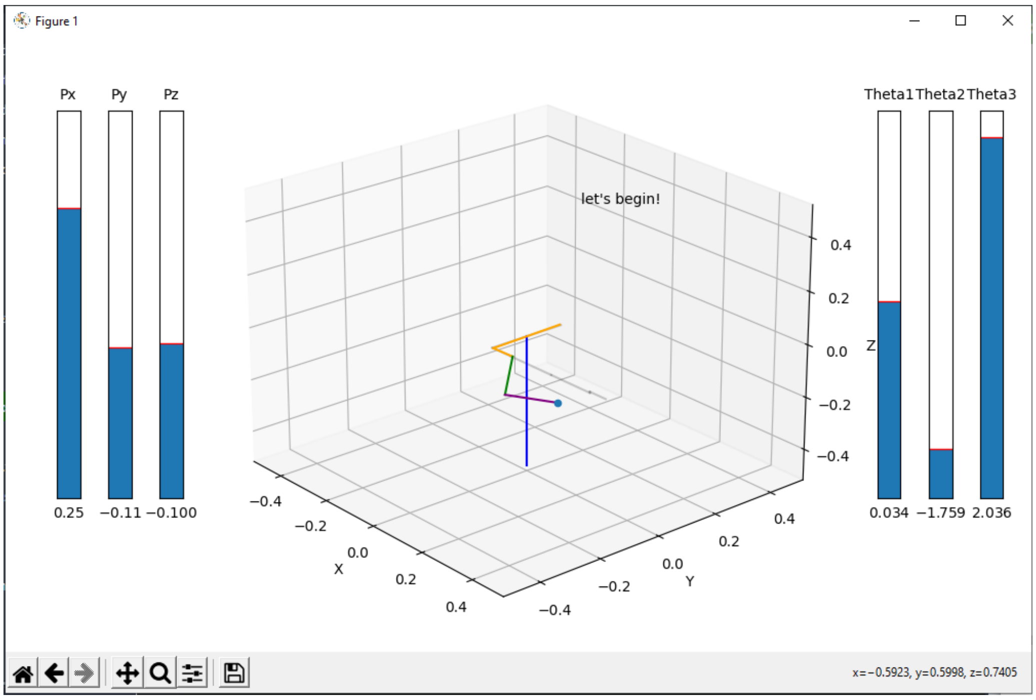


Figure 6: 3D-plot from the python test program for the IK-model. The blue line represent the torso, orange one the shoulders, green the upper arm and purple the lower arm. The blue dot is the target point (Px,Py and Pz) and the black shadow is the position of Hubert when all angles are 0.

4 Results

In figure 7 the final Hubert robot can be seen during the assessment phase of the project. It can be seen that the robot has approximately the same pose as the operator in the background. In the image most of the hardware can also be seen. To the bottom left is the computer and power supply. To the right is the Hubert, with the Arduino mega attached to the back (not visible in the image). In the background the router can be seen and the headset can be seen on the operator.

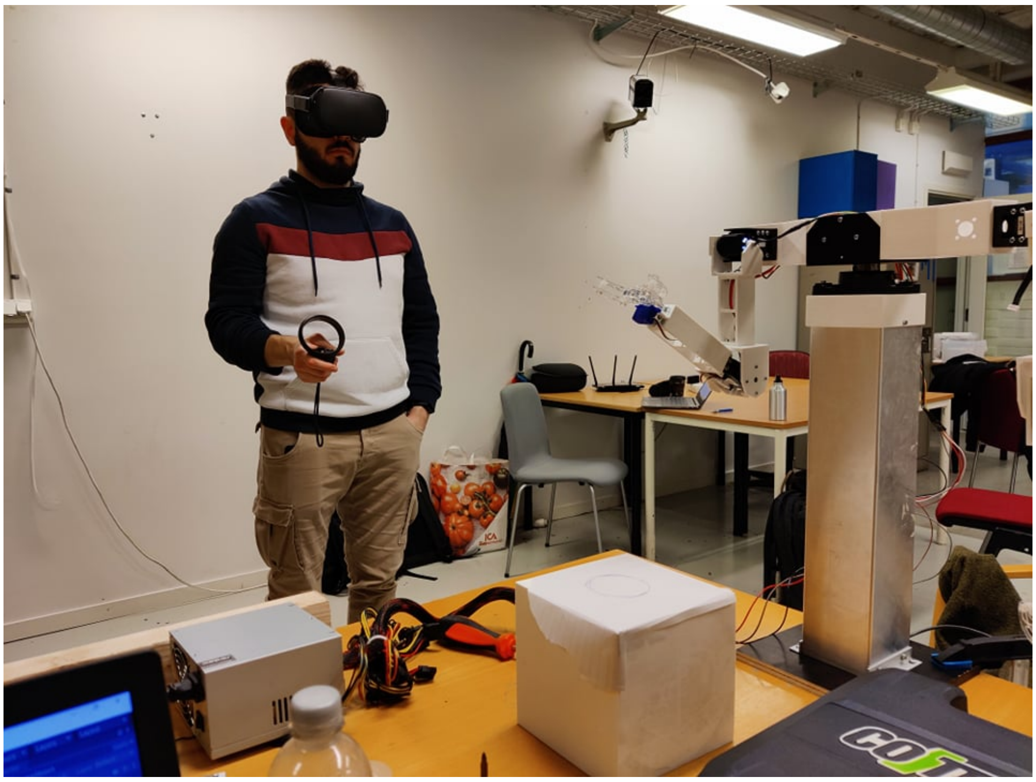
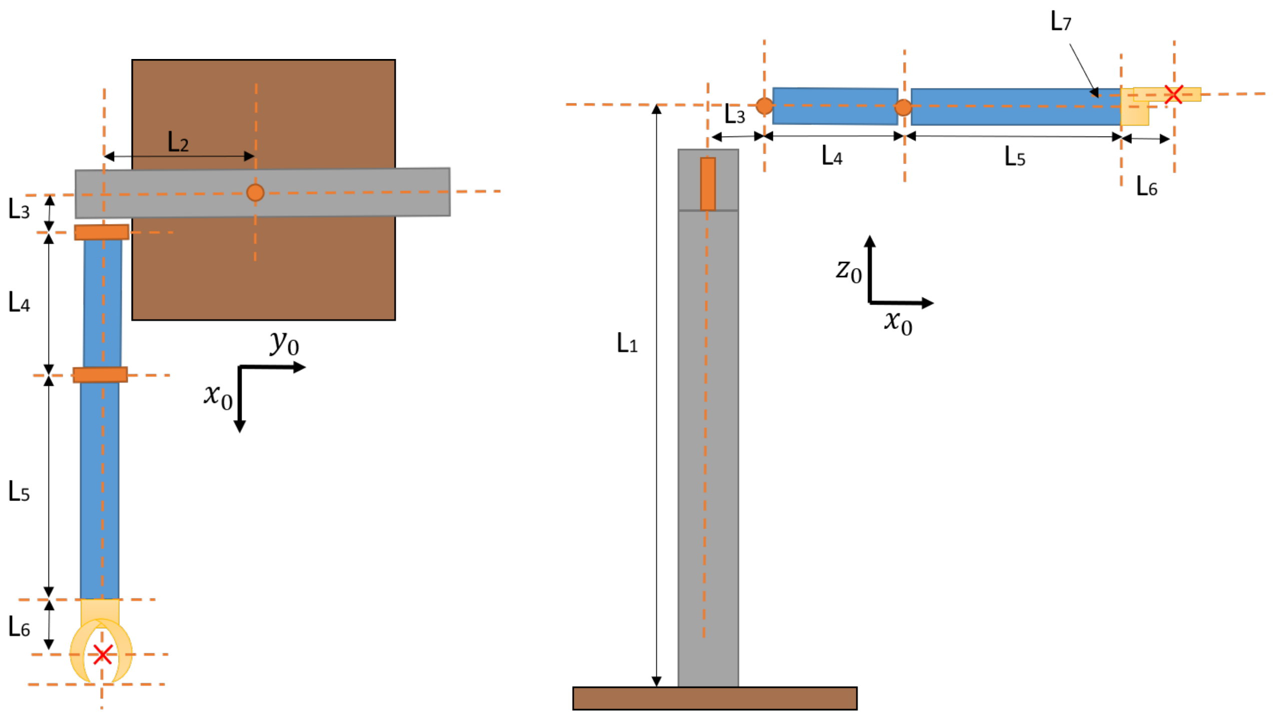


Figure 7: Final robot and operator during assessment

4.1 Assessment

The resulting model of the robot is shown in image 8.



(a) Seen from above. (b) Seen from the side.

Figure 8: Sketches of Hubert

To assess the resulting robot, the tests given in chapter 1.2 were made using a empty water bottle as an object. Associated pictures and videos on the testing of stages are shown in the Github repository for the project.

Stage 1: Simple movements

Several simple poses where tested by the operator, where the calculated angle was written down in table 2. The poses where the torso angle seem to have small errors, but the ones to the side did not give any valid values at all. Visually, seen in the pictures shown in the Github repository, it manage to match the target pose. However, it is clear from the images that there are visible errors as well, especially in the second pose.

Pose	θ_0	θ_1	θ_2	**target** θ_0	**target**θ_1	**target**θ_2
Arm straight forward	5,8	3,3	6,4	0	0	0
Arm bent	10,1	-100	104	0	-90	90
Arm up	6,7	64,9	10,8	0	90	0
Arm straight to the left	-	-	-	90	0	0
Arm straight to the right	-	-	-	-90	0	0

Table 2: The theta angles calculated live by the robot IK-model and the angles of the target pose that the operator is having in reality. Right and left from Hubert's perspective.

Stage 2 & 3: Placing object in area or accurately

Both stages are assessed at the same time, illustrated in the first video in Github. For stage 2, it succeeded in placing the water bottle in a designated area that was the white box. Even for stage 3 it succeeded to place the object accurately with an error of 0 to 1 cm measured from several test iterations. The errors however, where mostly only in the direction that the operator could not see since the operator looks at the scene from the side.

Stage 4: Picking and placing object

The same result as in stage 3 was achieved for stage 4 since the only extra moment was to pick up the object, which did not seem to be a problem. The test is illustrated in the second video in Github.

5 Discussion

At the time of the midway-milestone, the hopes for implementing a computer vision system, allowing Hubert to see, were high. However, as there was an accident at the lab, which broke some of Hubert's 3D printed bones, the time span of the project was reduced. With the already limited time-span, this incident removed the possibility of integrating the computer vision system.

Furthermore, as a result of the accident, we had to 3D-print the parts that would encapsulate the servomotors among other times. The time-constraints only allowed for a limited 3D-printing which implied reduction of the number of joint in the arm. From there previously being three, the number of joints were reduced to two. This did reduce the range of Hubert's movement. Subsequently however, this joint-reduction implied a simplified calculation of the inverse kinematics. However, the developed inverse kinematics is not dynamic which implies that if a part of Hubert is changed, the whole model has to be changed.

Measure of success was evaluated in accordance with the four stages mentioned in section 1.2. As mentioned in the results, stage one carried extremely varying results. It seemed to be considerably dependent on the pose that was tested by the operator. Taking into account that the anatomy of Hubert differs widely from that of the operator, it may be argued that this is the reason for the varying results. Furthermore, given that the only knowledge given to Hubert is where the target position is, and nothing else, it is reasonable that the interpretation of how to reach that position is open.

As for stage two, three and four, however, the test was a success. Considering case two, Hubert was able to place the water bottle in a designated area. On the other hand, this was only possible if the operator learned how Hubert moves and adapted the movements after Hubert. One may argue that this was only possible when the operator followed Hubert, and not the other way around. The same results acquired for stage three, with the error being mostly due to the limitations of sight for the operator since the operator is seeing the scene from the side.

The system-chain consists of many components, the primary ones being Unity, Arduino, motors, VR-headset and router. With increased number of components, there is a larger probability of errors in the chain. On the other hand, we would probably need more components that collects more information, such as a visual complement. This would decrease the error that the operator perceived due to limited sight.

6 Conclusion and Summary

The conclusion of this project is that it is considered a success. Regardless of the circumstances, the work was dynamic and managed to fulfill all stages. The movement being accurate is a big difficulty, but perhaps it doesn't matter. One could argue that the quality of the performance is a learning curve of the operator.

6.1 Future developments

For the future, there are many aspect that are subject to possible improvements, one of the first ones being fixing the anatonmy problem. This would imply modification of the hardware while taking into account this information in software.

Furthermore, the inverse kinematics needs to be expanded. Both to be formulated as a more dynamical approach so that Hubert could be subject to change without having to change the whole model. This would include using libraries and possibly ROS. There is also a possibility of the robot being controlled using ROS-Unity integration. It is also of interest to allow for Hubert to reach a further range in his movement.

Furthermore, one could add visual complements through receiving a video feed from Hubert into the VR headset, expand to two arms and use more joints. The issue of many parts in the system-chain is a contributing factor in added latency, so working on the communication between Unity and Arduino is also a point for future development.

References

J. P. Barnet, “How to make oculus quest 2 games // introduction to unity vr development.” [Online]. Available: https://www.youtube.com/watch?v=wnn-dzHz-tA

U. . Technologies. Unity - manual: Unity xr input. [Online]. Available: https://docs.unity3d.com/Manual/xr_input.html

zoomkat. Response to: Sending multiple values to arduino through serial [arduino online forum]. [Online]. Available: https://forum.arduino.cc/t/sending-multiple-values-to-arduino-through-serial/40454/5

A Code & Demo

The code and demo video can be found on our GitHub repository here.