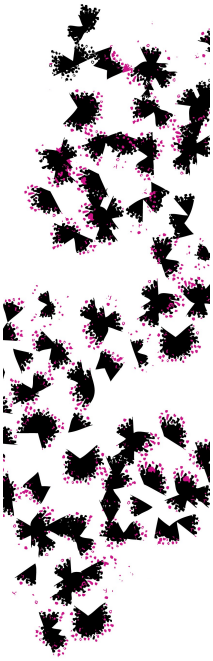

Visual puppeteering using the Vizualeyez 3D motion capture system

M. (Mark) van Holland

BSc Report

Committee:
Dr.ir. E.C. Dertien
Prof.dr.ir. G.J.M. Krijnen
Dr.ir. R.W. van Delden

July 2018

017RAM2018
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

The goal of this report is to find a method which can be employed to puppeteer robots. The main thing that have to be kept in mind is that the movements made by the robot have to mimic the movement of the puppeteer.

To accomplish this goal two methods which should be able to do this are described and compared to eachother. One of the methods involves kinematics and inverse kinematics. This method has the puppeteer hold a target which is measured and from which the necessary variables are determined, which are used to calculate the needed motor positions. The second method involves measuring the angles of the movements of the puppeteer and use these to control the robot.

The puppeteer will be measured using the VZ4000 3D motion capture system. This system will communicate with MATLAB using the software provided. To test the methods, simulations will be done with the help of MATLAB. A robot desk light will be used as puppet and the methods will be used with this robot in mind. The robot will also be controlled real time by the puppeteer.

The method which had the puppeteer hold a target worked for the most part, it had the robot move to the desired point. The problem was however that that the orientation measurement of the target did not work completely as intended. The other method worked as intended for two of the four used angles. Out of the other two, one was influenced by two of the other angles, while the other angle of these two had problems in the definition of the positive and negative values.

The results of both methods show that both methods have potential to achieve the goal of this paper. The method involving the angle measurements is better suited for robot with less degrees of freedom, while the other method is expected to work better for robots with more degrees of freedom.

Contents

1 Introduction

Making robots mimic human movements is a subject that is already explored by some, but making the robot be able to convey emotions is a subject that is less explored. Mimicking human movements by robots is most often achieved by hardcoding, or using rigorous controllers, using this way of programming it is possible to try and give the robot some emotion in the way it moves. The problem with hardcoding, or the often used controllers, is that these are not intuitive to use and it often takes lots of time to make the robot do what you want.

In theatres the art of puppeteering is often used to bring inanimate objects to live and give them emotions an feelings. This is something that is not often used in robotics, but can prove useful to intuitively program a robot with movements that can convey emotion, or messages, with only movements.

There are some studies who looked at puppeteering as a way to program robots. One of these studies is by Guy Hoffman et all. [1] In this study they made a program that allowed the users to move a robot in front of a webcam and record a routine for the robot this way. The main issue with this is that the webcam can only record in 2D, meaning that bigger, more advanced robots can not be programmed this way.

In another study [2] they used the Kinect in combination with the Baxter robot to mimic the human movements. They were able to recreate human movements using one of their methods, while the other method failed to recreate human like movements. The problem with this first method however was that one of the movements was influenced by another movement, which should have been independent.

The goal of this paper is to find a method which can be used to do puppeteering on a robot. To achieve this goal the Visualeyez VZ4000, a 3D motion capture device will be used. In this paper two methods are proposed to achieve this goal. One method will measure the desired end point and orientation and use inverse kinematics to obtained the desired motor angles. The second method will measure the angles of the joints of the puppeteer and map these to the joints of the robot. These methods will also be compared to eachother.

2 Theory

The Visualeyes is used in tetherd mode. Meaning that the target LEDs are operated by a target control module (TCM), which is connected via a wire to the sensor. The combination of the LED connection and TCM ID gives a unique number to all the LEDs. Using this target ID, it is possible to separate the markers from one another, which means that when two markers move behind one another, it is still possible to tell which is which. Since the LEDs can be identified individually it is possible to recognize the positions of the places the markers are placed by using the looking at these marker id's and the locations these are at.

The Visualeyes can be used to record the movement of the markers and safe these recordings for further use. The other manner of operating the sensor allows for real time streaming of the data. This is what the end result should eventually be able to do. This method has MATLAB requesting frames from the sensor, which then sends the last recorded frame to MATLAB, using the MATLAB plugin, which are provided with the software of the system.

The vizualeyes will output it's the coordinate information in millimetres, so also the lengths used in the calculations will be expressed in millimetres. The axis of the Visualeyes can be set to how the user wants it, this is done by following a procedure as explained in the manual [3]. This step is done whenever the Visualeyes is set up, to make the positioning of the Visualeyez not influence the measurements.

The robot that will be used is the desk light which found here [4] (see also figure 2.1). This robot has five degrees of freedom (DOF). The software available on the site [4] will be used to control the robot, small adjustments have to be made however to have a working communication between the lamp and MATLAB. This way of controlling the robot is chosen because the software has a recording mode for the lamp, which can be altered to have MATLAB control the lamp over a serial connection.

The motor limits are: from -0.698 to 0.698 rad for the first motor, from 0 to -$\frac{\pi}{2}$ rad for the second motor, from 0 to 1.570 rad for the third motor, from -1.064 to 1.396 rad for the fourth motor, and from -1.047 to 0.785 rad for the fifth motor.

In the frame worked with in this paper, the x-axis is sideways, the y-axis is the towards the back, and the z-axis is sideways. (This is also shown in figure 2.3)

Figure 2.1: Robot desk light.

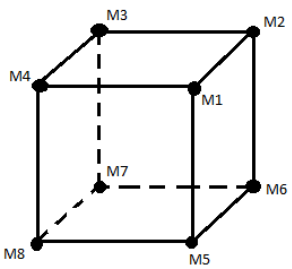

Figure 2.2: Target with marker placements.

2.1 End effector

For this method a target will be designed, which is held by the puppeteer and which is used to puppeteer the robot. Using the Visualeyez the position of the markers on the target can be measured and from the way the markers are placed the orientation and position of the centre of the target can be calculated. From this data the desired position and orientation of the end effector of the robot can be determined. This method will then calculate the motor angles needed to get the end effector to the desired position with the desired orientation. To make this possible the forward, and inverse, kinematics of the robot needs to be determined. One last thing needed is a scaling factor, which scales the things the puppeteer does to the appropriate dimensions of the robot.

2.1.1 Designing the target

The target needs to be designed in such a way that the position and orientation of it can be calculated, in a way that does not require much computing power, to minimize delays induced by these calculations. One other thing to keep in mind, is that some markers can be hidden from the sight of the sensor. Keeping the above mentioned aspects in mind, the target is chosen to be in the form of a cube, with markers placed on each of the corners, see figure 2.2. This way always four markers can be seen from the sensor, if nothing is obstructing the view of the sensor. The benefit of this placement is also that vectors pointing from one marker to a neighbouring marker are always along one of the axis.

From the obtained 3D coordinates of the markers the orientation can be calculated using the known placement of the markers. Figure 2.2 shows that the distance between markers M4&M1, M3&M2, M8&M5, and M7&M6 is only in the direction of the x-axis, markers M2&M1, M3&M4, M6&M5, and M8&M7 in the direction of y, and markers M5&M1, M6&M2, M7&M3, and M8&M4 in the direction of z. From the set of coordinates received from the measurements, one or more of each set mentioned above will be looked for. Because of the way the target is designed, it is possible to find at least one pair out of two of the sets. The distance vector pointing from one to the other marker of each of these detected pairs will be calculated. If only pairs were found out of two of the sets, a vector along the last axis can be obtained by the cross product of the two known vectors. (This step will be done after normalizing the vectors.)

Because all the vector lengths are equal to the length of the ribs of the cube, it is possible to normalize the vectors by dividing them by the rib length. This results in vectors which are of length one and that have their direction along one of the axis of the cubes frame, expressed in the reference frame.

The above calculated vectors, can also be calculated by multiplying the rotation matrix R, the rotation matrix from the cubes frame to the reference frame, with the vectors along the axis of the cube in the cubes frame:
$\begin{bmatrix}1\\0\\0\end{bmatrix}$ for the cubes x-axis, $\begin{bmatrix}0\\1\\0\end{bmatrix}$ for y, and $\begin{bmatrix}0\\0\\1\end{bmatrix}$ for z.
This results in the following vectors expressed in the reference frame:
$\begin{bmatrix}R_{1,1}\\R_{1,2}\\R_{1,3}\end{bmatrix}$ for the vectors along the cubes x-axis, $\begin{bmatrix}R_{2,1}\\R_{2,2}\\R_{2,3}\end{bmatrix}$ for the vectors along y, and $\begin{bmatrix}R_{3,1}\\R_{3,2}\\R_{3,3}\end{bmatrix}$ for the vectors along z.
Using both ways of calculating the vectors along the cube ribs, the rotation matrix can be calculated.

From the rotation matrix the angles ψ, θ, and ϕ can be calculated with equations 2.1, to 2.3. [5]

$$\psi = atan2(R_{2,1}, R_{1,1}) \tag{2.1}$$

$$\theta = atan2(R_{3,1}, \sqrt{R_{2,3}^2 \cdot R_{3,3}^2}) \tag{2.2}$$

$$\phi = atan2(R_{3,2}, R_{3,3}) \tag{2.3}$$

Where atan2 represents the 4 quadrant inverse tangent: [6]

$$atan2(x,y) = \begin{cases} tan^{-1}(\frac{y}{x}), & x > 0 \\ \pi + tan^{-1}(\frac{y}{x}), & y \geq 0, x < 0 \\ -\pi + tan^{-1}(\frac{y}{x}), & y < 0, x < 0 \\ \frac{\pi}{2}, & y > 0, x = 0 \\ \frac{\pi}{2}, & y < 0, x = 0 \\ 0, & y = 0, x = 0 \end{cases} \tag{2.4}$$

Now that the orientation is known, the position of the centre of the cube can be calculated by multiplying the inverse of the rotation matrix with the known marker, positions on the cube, in the cubes reference frame. This value will then be subtracted from the measured marker position to obtain the position of the centre of the cube.

2.1.2 The scaling factor

The scale factor between the robot and the puppeteer can be found by the use of a calibration step. This step will require the puppeteer to straighten the part of the body that will be used for the puppeteering, which in this paper will be the lower arm. When the puppeteer does this, the position of the target can be measured, and using a marker on the stationary part connected to the moving part, in this case the elbow, the distance between these locations can be calculated. This is the maximum distance the puppeteer can reach during the puppeteering. This distance is than divided by the distance the robot reaches when it is also stretched out to gain the scaling factor, which in this case is 414 mm. The maximum distance is set to 400 in the system to have some margin for errors. The marker on the base will also be seen as the origin. This means that the position of the target with respect to this marker is calculated and used in the rest of the paper.

2.1.3 Forward kinematics

The goal in this part is to find a formula, using which the position and orientation of the end effector can be determined.

$$\vec{P} = f(\vec{q}) \tag{2.5}$$

Where $\vec{P}$ is a vector which denotes the end effector position, and orientation, and $\vec{q}$ is the vector of the motor angles. Using joint space a transformation matrix will be calculated, from which the end-effector position and orientation can be determined.

The transformation matrix from the end effector frame to the base frame can not be determined in a single step if the robot consists of multiple bodies. To get the transformation the chain rule is used, which states that the transformation from frame n to frame 0 equals the transformation matrixes between the separate bodies multiplied with each other: [7]

$$H_n^0 = H_1^0 H_2^1 ... H_n^{(n-1)} \tag{2.6}$$

Where H_n^0 describes the transformation matrix from frame n to frame 0.

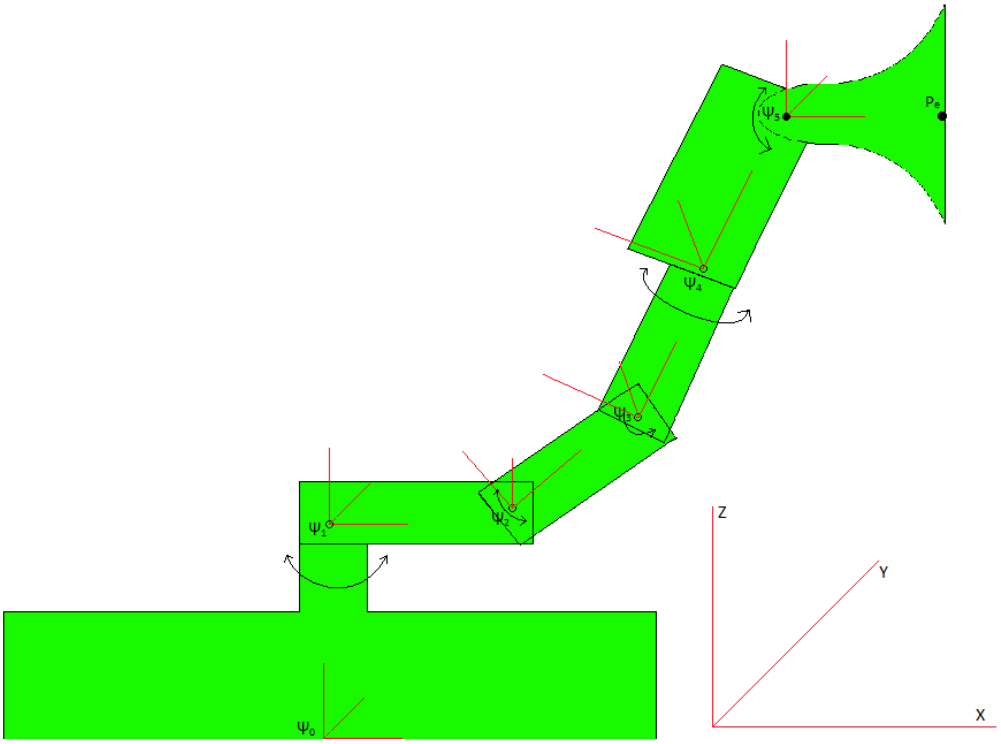

Figure 2.3: Drawing of the lamp with the different frames.

Before the transformations between the bodies of the robot are determined, it is necessary to find out what type of joints are between the bodies. In this case all the joints are rotational joints which only rotate around one axis. To get the transformation from one body to the other it is assumed that the origin of the first body's axis is located in the joint between the first and zeroth body. The second body's axis in the joint between the first and the second body etc. The zeroth body is the stationary part of the robot and acts as the base. See figure 2.3. Using this it is possible to determine the transformation matrixes, which only depend on the angle of the joints and the distance between the frames. For the robot used in this paper this is done as follows.

The zeroth body is connected via a rotational joint, which rotates around the first body's z-axis, to the first body. The distance from the bottom of the zeroth body to the first joint, expressed in the zeroth frame is 0 mm in x, 0 mm in y, and 96 mm in the z direction, this results in the matrix: [8]

$$H_1^0 = \begin{bmatrix} cos(q_1) & -sin(q_1) & 0 & 0 \\ sin(q_1) & cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 96 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The fist body is the connected to the second body via a rotational joint, which rotates in around the second body's y-axis. The distance from the first joint to the second expressed in the first frame is (-15; 0; 30). The second body is connected to the third body via a rotational joint, which rotates around the third body's y-axis. The distance from the second and third joint is (78; 0; 1).

$$H_2^1 = \begin{bmatrix} cos(q_2) & 0 & sin(q_2) & -15 \\ 0 & 1 & 0 & 0 \\ -sin(q_2) & 0 & cos(q_2) & 30 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad H_3^2 = \begin{bmatrix} cos(q_3) & 0 & sin(q_3) & 78 \\ 0 & 1 & 0 & 0 \\ -sin(q_3) & 0 & cos(q_3) & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The third body is connected to the fourth body via a rotational joint which rotates around the fourth body's x-axis. The distance from the third to the fourth joint is (68; 0; 0).

$$H_4^3 = \begin{bmatrix} 1 & 0 & 0 & 68 \\ 0 & cos(q_4) & -sin(q_4) & 0 \\ 0 & sin(q_4) & cos(q_4) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Lastly the fourth body is connected to the fifth body via a rotational joint which rotates around the fifth bodys y-axis and the distance is (48.5; 0; 0).

$$H_5^4 = \begin{bmatrix} cos(q_5) & 0 & sin(q_5) & 48.5 \\ 0 & 1 & 0 & 0 \\ -sin(q_5) & 0 & cos(q_5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Using these transformation matrices the transformation matrix from the last body to the base can be calculated (see appendix A). The end effector is located on the last body (point p_e in figure 2.3), which means that the end effector is a fixed point in the frame of the last body. Using the end effector position in the last body's frame, the position in the reference frame can be calculated using the transformation matrix from the last frame to the reference frame: [10]

${}^0P_e = H_5^0 \cdot {}^5P_e$, where ${}^5P_e$ is the homogenous coordinate of P_e in the fifth fame. ${}^5P_e = \begin{bmatrix} 93.5 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

The end effector in this case also has the same orientation as the last body, meaning that the orientation of the end effector can be calculated from H_5^0. To get the orientation from the transformation matrix, equations 2.1 to 2.3 can be used.

Now that the position and the orientation can be calculated from the motor positions, the inverse kinematics has to be done.

2.1.4 Inverse kinematics

The goal here is to find the set of angles $\vec{q}$ which result in $\vec{P} = \vec{T}$, where $\vec{T}$ is the target location and orientation of the end effector, which was calculated in section 2.1.1. Here the damped least squares method is used to obtain the desired motor angles.

Finding the set of angles that results in $\vec{P} = \vec{T}$ is often not possible by simply inverting f, since this function is often highly non-linear. [9] Because of this another method is needed to find the desired angle values. To achieve this a numerical solution using the Jacobian matrix can be used. The Jacobian is a function of changing joint angles around the current joint angles and how these relate to a change in the end effector:

$$J(\vec{q}) = \begin{bmatrix} \frac{\partial P_1}{\partial q_1} & \frac{\partial P_1}{\partial q_2} & \cdots & \frac{\partial P_1}{\partial q_n} \\ \frac{\partial P_2}{\partial q_1} & \frac{\partial P_2}{\partial q_2} & \cdots & \frac{\partial P_2}{\partial q_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial P_6}{\partial q_1} & \frac{\partial P_6}{\partial q_2} & \cdots & \frac{\partial P_6}{\partial q_n} \end{bmatrix} \tag{2.7}$$

where n is the amount of DOF of the mechanism.

This matrix can be obtained by taking the partial derivatives, with respect to the different joint angles, from the transformation matrix calculated earlier. During each iteration step only the current motor angles need to be filled in this way. If it is not possible to take the partial derivatives of the transformation matrix, it is also possible to approximate the individual values of the Jacobian by numerical approximations by taking $\frac{\partial P_1}{\partial q_1} = \lim_{\Delta \to 0} \frac{\Delta P_1}{\Delta q_1}$. This approximation how-

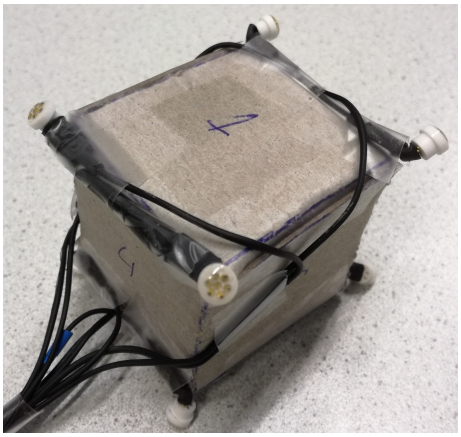

Figure 2.4: The resulted targed.

ever, requires to be calculatd again for every iteration step. The Jacobian still relates a change in angles to a change in end effector position $\vec{P'} = J(\vec{q}) \cdot \vec{q'}$. The Jacobian can be computed for the current joint angles J=J($\vec{q}$) and the next set of joint angles, is equal to the current set plus an update value: $\vec{q}(t^+) = \vec{q}(t) + \Delta\vec{q}$. Let $\vec{e}$ denote the error between the desired position of the end effector, and the current position of the end effector $\vec{e} = \vec{T} - \vec{P}$. Using this, the following equation can be obtained:

$$\vec{e} = J\Delta\vec{q} \tag{2.8}$$

This shows that if the inverse of the current Jacobian can be obtained, the necessary change in the joint angles can be calculated by using the error $\vec{e}$. [9]

Since the Jacobian is not always a square matrix it can not be directly inversed. There have been proposed several methods to calculate an inverse of the Jacobian, all with their pro's and con's.

One of these methods is the pseudo inverse Jacobian. This method has the problem however that when the matrix is near singularity, that the method will not converge. One other method simply transposes the Jacobian. This method is computationally fast, but this method can not reach the desired position with the same accuracy the pseudo inverse method can. The method that is used here is the damped least squares method (DLS). This method avoids the problem the pseudo-inverse method has with singularities. One thing DLS does different from the pseudo inverse method is the fact that it wants to give a best solution to (2.8), instead of a minimum vector $\Delta\vec{q}$ [9].
The DLS uses the equation:

$$\Delta\vec{q} = (J^T J + \lambda^2 I)^{-1} J^T \vec{e} = J^T (J^T J + \lambda^2 I)^{-1} \vec{e} \tag{2.9}$$

Here λ is the damping factor, which needs to be large enough as for the equation to be well behaved near singularities, but when it becomes too large the convergence rate will be slow. [9]

2.1.5 Implementation

The target is made from carboard, and the LED markers are taped to the corners of it. (see figure 2.4) The ribs of the cube are all 40 mm in length, this length is chosen to have some distance between the markers, so inaccuracies in the measurements have less influence on the calculations, but the target is also still not too big to hold.

Each face of the cube is numbered from 1 to 6. The LED-IDs of the LEDs on the corner of each Face are denoted, to know what marker on what corner. Face number 2 is seen as the front of the cube, with face 1 as the top.

The calibration step will be the puppeteer holding the cube upright, with face 1 facing to the sensor, and face 2 facing down. This way the orientation of the target is $-\frac{\pi}{2}$, which also the orientation of the end effector when the lamp is up right.

For the calculations a MATLAB script is made, which uses the marker coordinates and the way they are arranged and determines first the target's orientation and using this the position, like explained in section 2.1.1. The calculated transformation matrix is used to calculate the end effector position and orientation of the robot, using measured motor data, and determines the error $\vec{e}$. The partial derivatives needed to calculate the Jacobian are implemented as functions, so only the current motor angles are needed to get the derivatives for the current iteration.

Using these values for $\vec{e}$, and J the new motor angles are calculated according to formula 2.9, these angles are then communicated using the serial connection to the robot. After this the program starts again from the point of calculating the error.

The value of λ is determined by running a series of simulation, where the only change between them is the vale of λ. This resulted in the value for λ being: 100.

Since the error of the orientation is a value between $-\pi$ and π and the error of the position can be between -400 and 400, a weighing factor is implemented so both types of errors contribute equally in the calculation of the angles:

$$\vec{e} = \begin{bmatrix} T_x - P_x \\ T_y - P_y \\ T_z - P_z \\ 100 \cdot (T_\psi - P_\psi) \\ 100 \cdot (T_\theta - P_\theta) \\ 100 \cdot (T_\phi - P_\phi) \end{bmatrix} \tag{2.10}$$

2.2 Angle mapping

This method will try to see what the angels the joints of the puppeteer make are and map these to the DOF of the robot. Meaning the amount of DOF, and the type of joints need to be taken into account when placing the markers on the puppeteer. Placing the markers at specific location will make it possible to calculate the angles the joints make from the coordinates off the markers. This information can then be used to map these angles one on one to the joints of the robot.

2.2.1 Placement the markers

Markers need to be places at appropriate places, so that all the desired angles can be calculated and the markers are always be visible for the sensor. For bending angles, markers need to be placed at the joint itself and both body parts connected to it. The further these are placed from the joint, the greater the accuracy of the measurement can be, since small errors in the measurements will then have less of an influence on the lengths between the markers. The marker on the not moving body part can be left out when the movements are of the joint that is stationary in the reference frame, in this case a virtual point can be calculated on one of the axis of the frame. For twisting joints multiple markers will be placed at the moving joint, at equal distance from the centre of rotation.

When placing the markers the placement of the sensor will also be taken into account, because every place mentioned above must have at least one marker visible for the sensor.

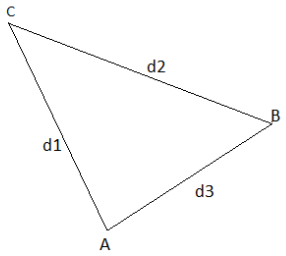

Figure 2.5: Trangle used in the law of cosines.

2.2.2 Calculation of the angles

To calculate the angles the joints make the law of cosines is going to be used: [2]

$$\angle ACB = cos^{-1}(\frac{dq^2 + d2^2 - d3^2}{2d1d2}) \tag{2.11}$$

Where A,B,C,d1,d2 and d3 are depicted in figure 2.5 the distances d1, d2, d3 can be calculated using:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.12}$$

When calculating the angles one must keep in mind what the rest position is, or what the axis is from which the joint makes an angle. The distances between the tree points of interest for one angle can be calculated, and the law of cosines can be used to calculate the angle the joint makes. The angle that is calculated is not always the angle that is of interest. Think of the elevation of the lower arm, the markers would then be placed at the shoulder, elbow, and the wrist, so the angle is calculated with respect to the shoulder and not with the position which is considered the rest position, which is when the arm is resting on the table. In this case the angle calculated need to be subtracted from the value of the angle when the arm is in rest position.

Calculating the twist of a joint, the fact that there are multiple markers used is going to be used. The transformation matrix from the main body, to the body part of interest is calculated, in this case this is used to transform the markers to a frame which is 'looking' from the arm at the wrist. From this frame the angle between one of the markers, and the centre can be calculated. The centre will be calculated by using the two outermost markers that can be seen.

2.2.3 Mapping the angles to the robot joint

When mapping the DOF the human can have, to the DOF the robot has, also see if the robot is over defined (DOF human < DOF robot), properly defined (DOF human = DOF robot), or under defined (DOF human > DOF robot). One other thing to look at are similarities between the human movements, and the robot movements. Think about the bending of the elbow, and a (rotating) joint that moves the robot body up and down.

When the robot is over defined it is possible to map some movement to multiple robot joints. Depending on the users preference this could be done, or some robot joints can otherwise not be used and kept in the same position.

With proper, and underdefined robots it is more likely to encounter a movement of the human, which can not be mapped on the robot. In this case the puppeteer must not use these movements when puppeteering just to be safe, so that it can not influence the other calculated angles.

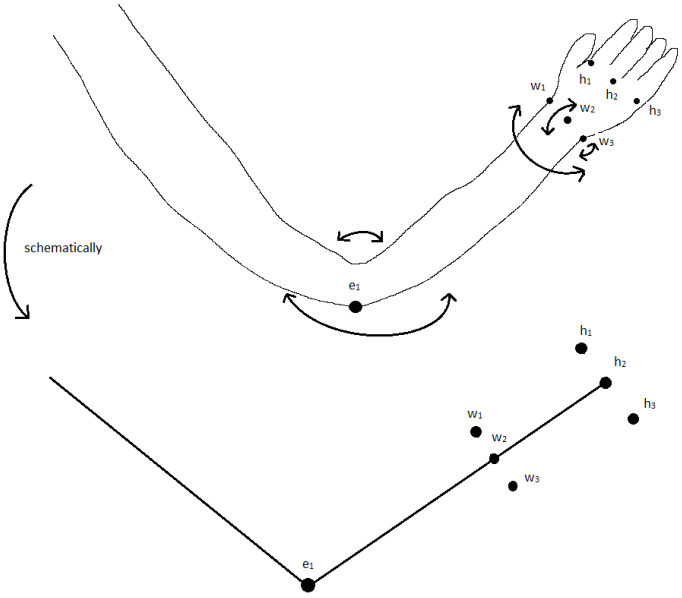

Figure 2.6: Movement options of the lower arm and the placement of the markers.

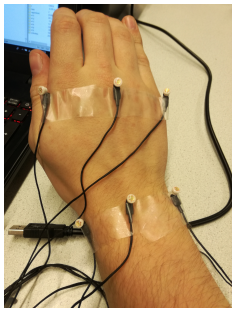

Figure 2.7: Placements of the markers on the wrist and the hand.

2.2.4 Implementation

The puppeteer uses his/her lower arm, and hand to puppeteer the robot. The types of movements the puppeteer can make are the bending of the elbow, the rotating of the elbow around the point that rests on the table, the bending of the wrist in two directions, and the twisting of the wrist. (see figure 2.6) The robot can rotate at the base, bend at 2 points after each other, rotate at the end, and bend the lamp. (see figure 2.3)

The bending of the elbow can be directly mapped to the second motor of the robot, this could also be divided over the second and third motor but here it is chosen not to do so, so that the movements made by the robor looks more like the movements of ans arm. The rotating of the elbow can be directly mapped to the first motor of the robot. The bending of the wrist in the direction of the hand palm can be mapped to the fifth motor of the robot. The twisting of the wrist to the fourth motor. Lastly the bending of the wrist in the direction normal to the hand palm can not be mapped directly to any one (or two) motor of the robot.

To measure the angles, markers are placed at all the joints of the lower arm, meaning at the elbow and the wrist. Because the wrist also has a twisting motion, also two more markers are placed at the end of the wrist. Markers are also placed at the end of the moving bodies, meaning again at the wrist, which are already there, and on the hand. Since the hand can turn away from the sensor 2 more markers are places here so the end of the hand can always be seen. The placements of the LED markers is also shown in figure 2.6 and also shown in figure 2.7 with the exception of the marker at the wrist.

The transformation matrix used to calculate the twist of the elbow is:

$$\begin{bmatrix} cos(q_2) & sin(q_2) & 0 & e_{1x} \\ -cos(q_1)sin(q_2) & cos(q_1)cos(q_2) & sin(q_1) & e_{1y} \\ -sin(q_1)sin(q_2) & -sin(q_1)cos(q_2) & cos(q_1) & e_{1z} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For all the angles It needs to be determined when they are negative. For the bending of the elbow, the angle would be negative if the marker on the wrist would be lower than the marker of the elbow. The turning angle of the elbow is considered negative when the wrist crosses the x-axis. The twisting of the elbow is seen as negative when the end points of the wrist cross the y-axis of the frame calculated using the transformation matrix mentioned above. The bending of the elbow is seen as negative when the markers on the hand pass the line that goes through the marker of the elbow and the marker on the wrist.

2.3 Communication with the robot

The communication with the robot will be done using the serial object in MATLAB. MATLAB will connect to the COM-port the robot is connected to. Using this port the functions scanf and fwirte can be used to read from the serial port and write to it respectfully. The robot is controlled using an Arduino ADK, the code of which is provided on the site. [4] This code is altered to communicate to MATLAP. The record of the program is used to make the connection with MATLAB. In this function a loop is made that has the code wait till there is serial data available. When it is available this data is read and used to set the motor angles. When this is done it sends back the angles the motors should have.

The data is sent starting with the angle of motor one, then two, and so on till motor five. After this three more values are sent which are not motor angles. The speed at which the Arduino code runs is set to 5 Hz, this is because when it ran faster there was a mismatch in the communication of the motor angles. (Motor angle two became motor angle one among other errors.)

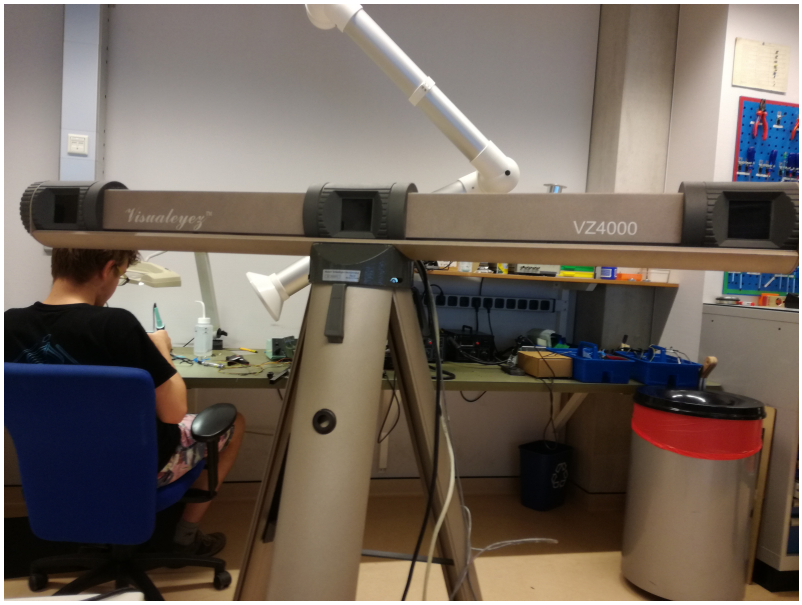

Figure 2.8: The Visualeyez VZ4000.

3 Experiments

3.1 Set-up

The visualeyes is used in the streaming mode explained in the theory, this is done because the whole system should work in the streaming mode and to get the most appropriate impression of the performance this mode should then also be used for the measurements. First when the Visualeyes is set up the reference frame is defined using method described in the manual. [3] The signal flow of the whole system is seen in figure 3.1. The sensor itself is show in figure 2.8.

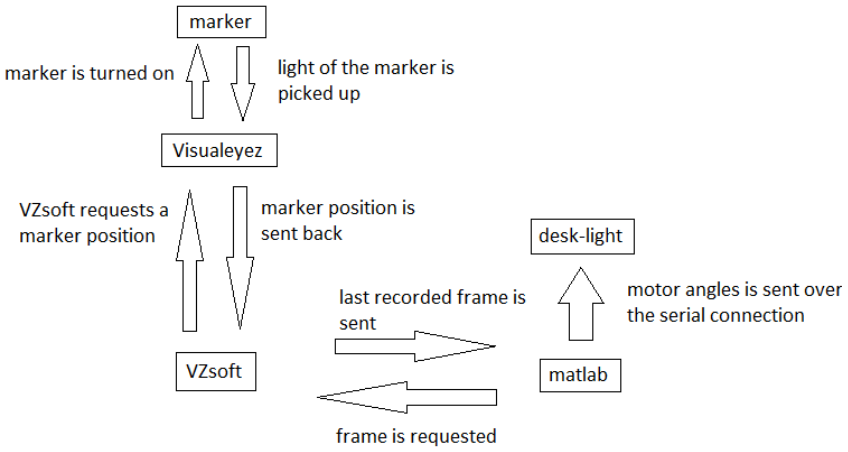

Figure 3.1: Signal flow diagram of the system

The first tests are used to see if the orientation, and location of the target can be correctly calculated by the system. The second test will involve the end effector method and have the puppeteer preform pre-defined routines, and one improvised routine. During this test the sensor will be oposite of the puppeteer. The third test will involve the angle method and has the puppeteer preform some pre-defined routines, and an improvised routine. During this last test the visualeyez will be positioned next to the puppeteer.

3.2 Test one

3.2.1 Results

To see if the orientation detection worked as intended, the target was hold above the marker that signified the origin in different orientations. The results of these measurements are shown in figures 3.2 to 3.5. In figure 3.2 the orientation of the target was (0, 0, 0), in figure 3.3 it was (0; $\frac{\pi}{2}$; 0), in figure 3.4 ($-\frac{\pi}{2}$; 0; 0), and lastly in figure 3.5 the orientation was (0; 0; $\frac{\pi}{2}$).

Figures 3.6 to 3.8 show how the position, and orientation, of the target that is determined by the system. In figure 3.6 the target was placed at (-100; 0; 20), in figure 3.7 at (0; 100; 20), and in figure 3.8. at (0; 0; 100) again the distances are given in millimetres. The first two positions also have a non-zero z element, because the cube was put on top of the table and the distance from the surface of the cube to its centre is 20 mm. The position is scaled by the scaling factor that is calculated at the beginning of the program, which means that the expected values are: (-392; 0; 78), (0; 392; 78), and (0; 0; 400). The scaling factors were calculated to be 4.06, 4.06, and 4.01. For the first measurement, the values that coordinates that are obtained are (395; 16; 100). Scaling this back to the original scale using the scaling factor gives (97; 4, 25). For the second measurement (-59; 400; 99), which leads to (10; 99; 24), and for the third the values ranged between (0; 12; 400) to (15; 30; 432), so the scaled version would be between (0; 3; 100) and (4; 7; 108).

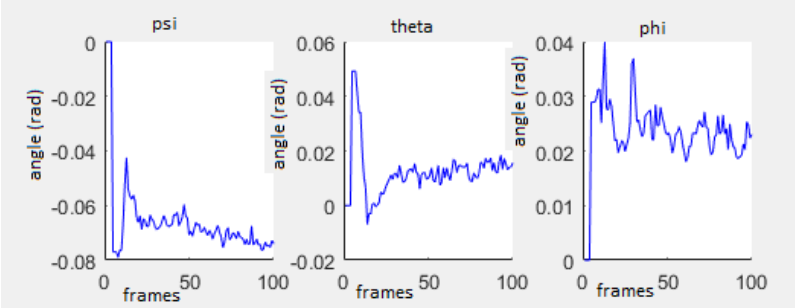

Figure 3.2: Measured orientation when the real orientation was (0;0;0).

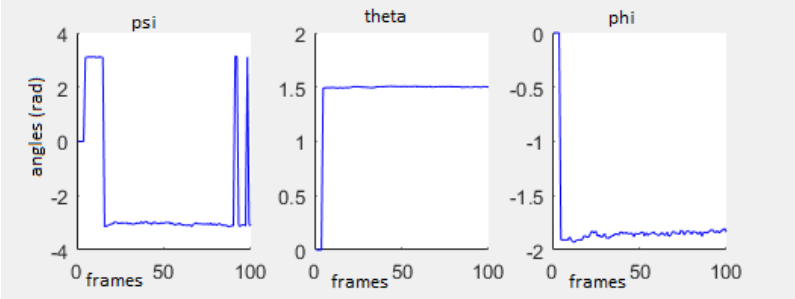

Figure 3.3: Measured orientation when the real orientation was $(0;\frac{\pi}{2};0)$.

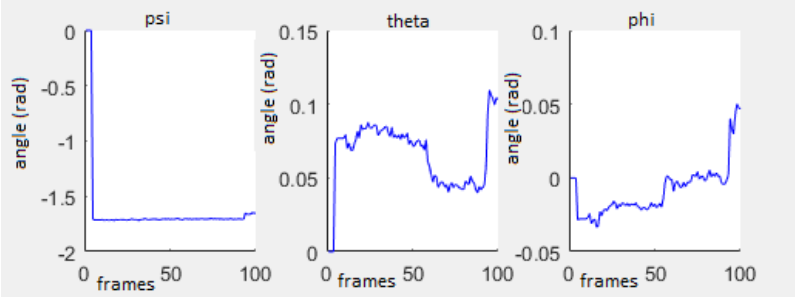

Figure 3.4: Measured orientation when the real orientation was $(-\frac{\pi}{2};0;0)$.

Figure 3.5: Measured orientation when the real orientation was $(0;0;\frac{\pi}{2})$.

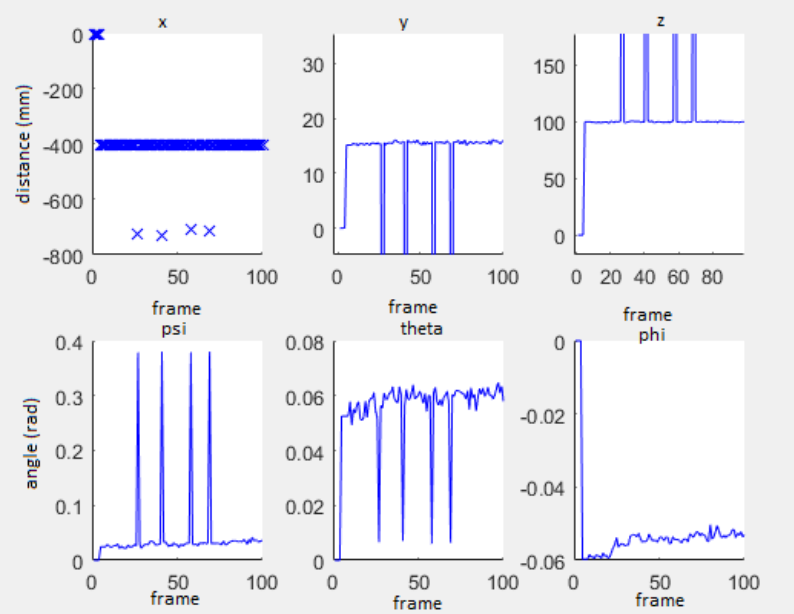

Figure 3.6: Measured position and orientation when the scaled position was (-400;0;20) and the real orientation (0;0;0).

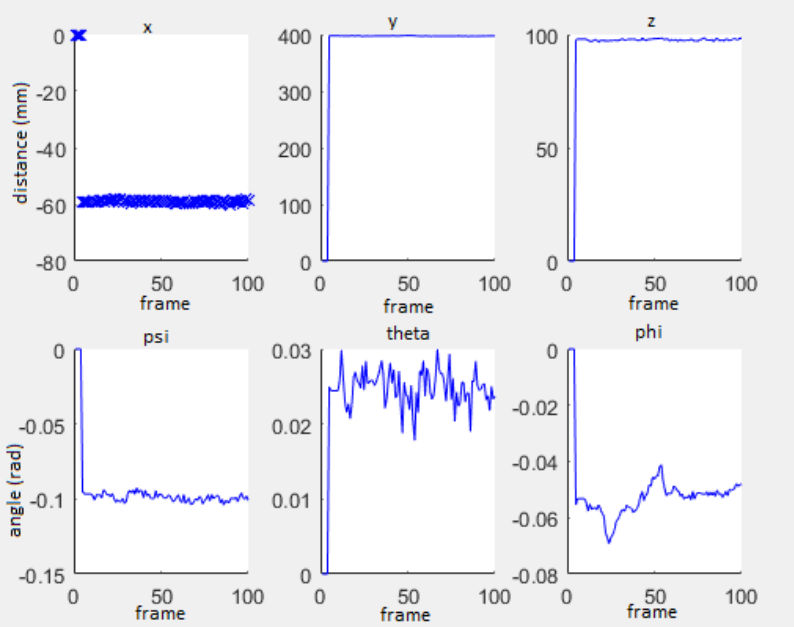

Figure 3.7: Measured position and orientation when the scaled position was (0;400;20) and the real orientation (0;0;0).

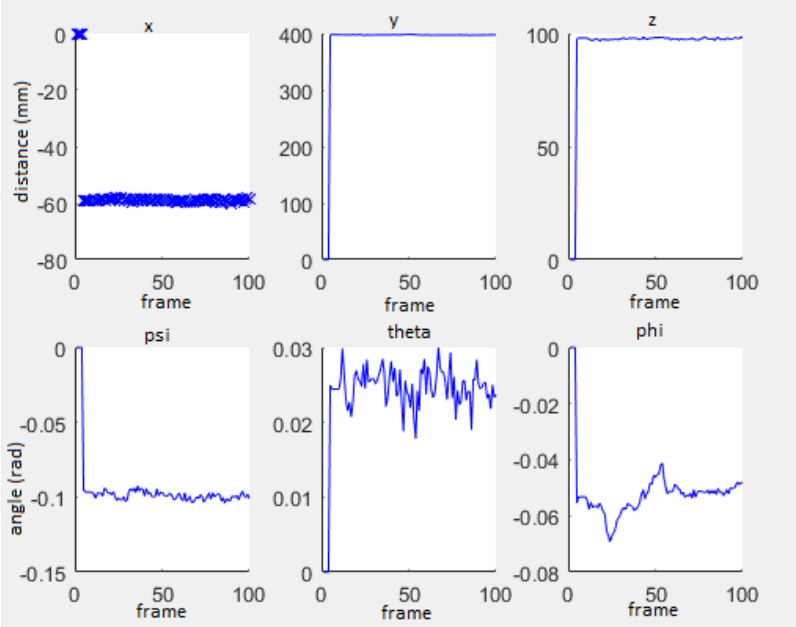

Figure 3.8: Measured position and orientation when the scaled position was (0;0;400) and the real orientation (0;0;0).

3.2.2 Discussion of the results

Figure 3.2 shows that the orientation within 0.1 rad from the value it should be at.

Figure 3.3 shows that when the target is turned around the y-axis, the orientation is not accurately determined. The value of θ is within 0.1 of the of $\frac{\pi}{2}$,which is where it should be. The value of ψ is jumping form π to $-\pi$, which would mean the target is completely turned the other way around around the z-axis, also the value for ψ is wrong at -1.91. This means that when the orientation of the target changes in θ the measured results of the total orientation will be effected.

Figure 3.4 shows that the value of ψ is 1.70 rad, which is an error of 0.13 rad from the desired $\frac{\pi}{2}$. The values for θ, and ϕ within 0.1 rad from the desired position. This means that when the target turns around the z-axis, the orientation can be measured with an accuracy of $\pm$0.13 rad.

Figure 3.5 shows that the orientation can be measured within the accuracy considered human error.

Figure 3.6 shows that the measurement data obtained sometimes contains a faulty data point resulting in the orientation, and position detection giving a often drastically different value. The results of the position detection show that the position can be determined within 10 mm of the actual value.

Considering all the results mentioned above, the system works for detecting the position of the target, but the orientation is faulty when the orientation contains a rotation around the y-axis of the target. This effects the total system for it looks at both the position, and the orientation to determine the required motor angles.

3.3 Test two

3.3.1 Results

Figures 3.9 to 3.14 show the results of test two, where the second method was used. During all these measurements the puppeteer holds his elbow still on the table. In figures 3.9 to 3.14 the blue line is the target location and orientation, the red line is the end effector location and orientation of the robot, and the green line is the error between these two. Table 3.1 gives the RMS values and the scaling factors of the different measurements.

For the first measurements, shown in figure 3.9, the puppeteer was asked to hold the target diagonally towards the sensor. The end effector of the robot can be seen to move to the value of the set point, until it converges to some value. When the robot can not reach the desired end point value it still mimics the changes that occur in the target position to some point. This can be best seen in the y position in figure 3.9. The RMS values for this measurement can be seen in table 3.1.

For the second measurement, shown in figure 3.10, the puppeteer was asked to first hold the target in the calibration position, and then move to the same position as the first measurement. In figure 3.9 it can be seen that the target changes position, and orientation. The orientation of θ and ϕ seem to change yet another time when the target was already in the second position. Just like in the first measurement the end effector tends to the target, till it converges to some value. The RMS values for this measurement are shown in table 3.1.

For the third measurement, shown in figure 3.11, the puppeteer was asked to hold the calibration position. Figure 3.9 shows that the value for the y position of the robot converges to the target y position, while the x, and z positions have a, marginally, constant error with the target position. The RMS values, and scaling factor, for this measurement can be seen in table 3.1.

For the fourth measurement, shown in figure 3.12, the puppeteer was asked to after the calibration step, move the target forward, and down, and from this position rotate his arm left and right. The left and right movement can be seen to be happening from frame 30, till the end of the measurement. The same observations as in the previous measurements can be made about the position of the end effector. The orientation of the end effector seem to converge to a value closer to the target orientation in the case of θ and ϕ, but not for ψ. The values for the RMS, and scaling factor can be seen in table 3.1.

For measurement five, seen in figure 3.13, the puppeteer was asked to do the same as in the second measurements, with the only difference that the position he moved the target to was closer to the origin. As can be seen in figure 3.13, this made it so that the position of the end effector converged to the position of the target. The orientation of the end effector also converged, with the exception of the value for ψ to the target orientation, but with a slower rate than the position. The RMS values, and the scaling factor, are shown in table 3.1.

For the sixth measurement, shown in figure 3.14, the length of the program was increased and the puppeteer was improvising the movements. The value of ϕ of the end effector follows that value of the target. The position of the end effector converges to the position of the target, but when it can not reach this the end effector converges to the value closest to it, while still able to mimic the changes in the target position.

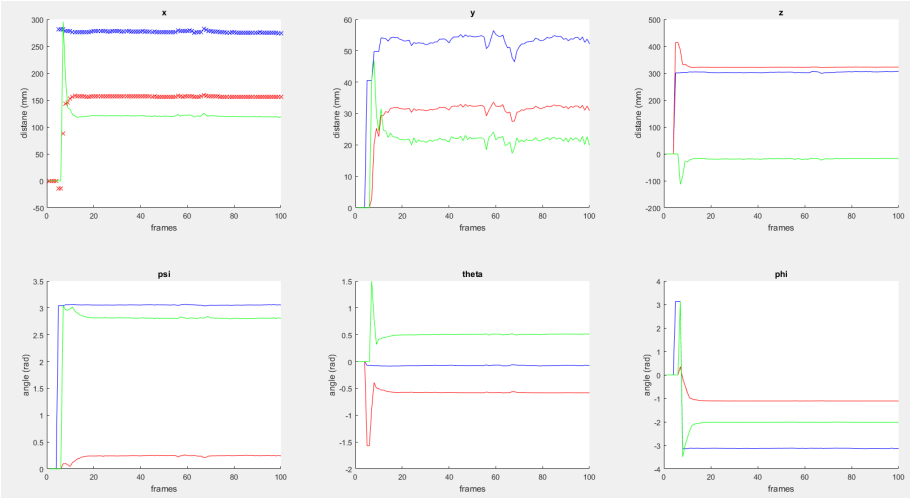

Figure 3.9: Measured position and orientation of the first measurements. Here the blue line is the targed, the red line is the end effector, and the green line is the error between the two.

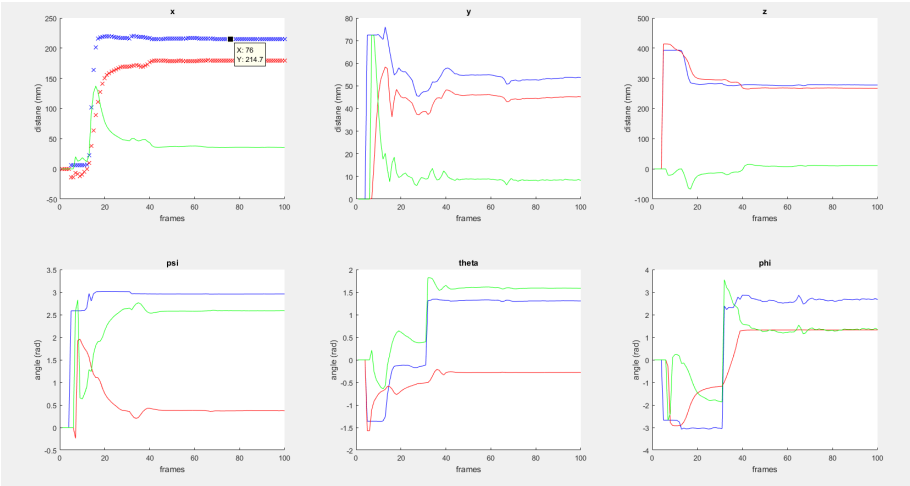

Figure 3.10: Measured position and orientation of the second measurements. Here the blue line is the targed, the red line is the end effector, and the green line is the error between the two.

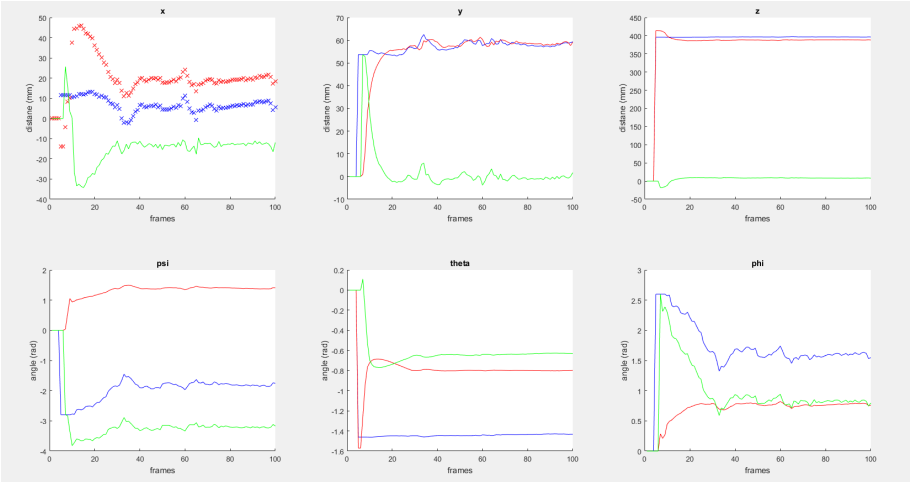

Figure 3.11: Measured position and orientation of the third measurements. Here the blue line is the targed, the red line is the end effector, and the green line is the error between the two.

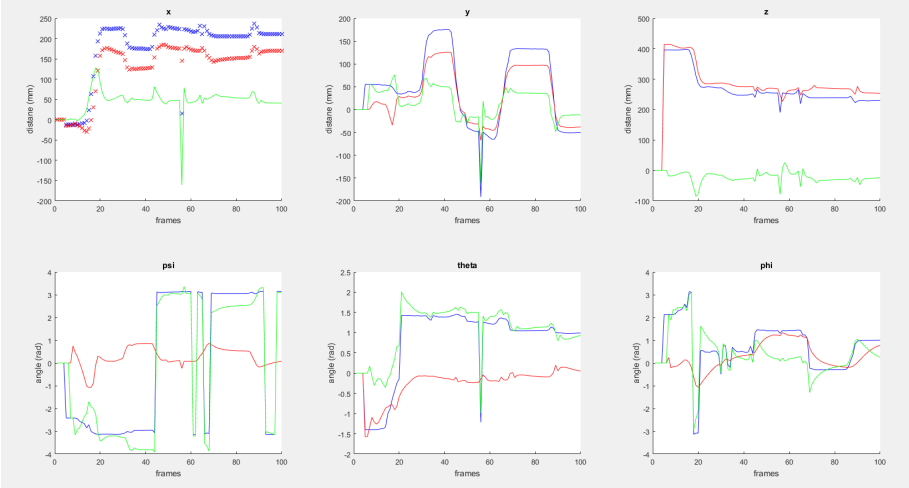

Figure 3.12: Measured position and orientation of the fouth measurements. Here the blue line is the targed, the red line is the end effector, and the green line is the error between the two.

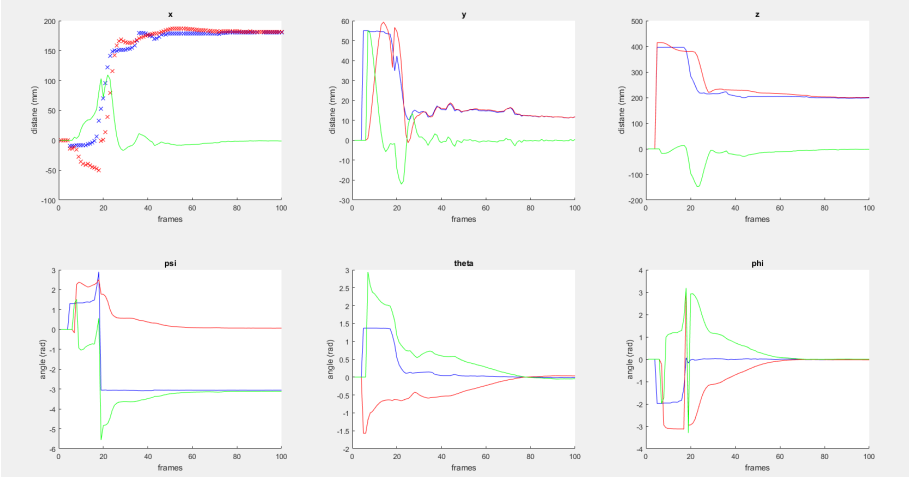

Figure 3.13: Measured position and orientation of the fifth measurements. Here the blue line is the targed, the red line is the end effector, and the green line is the error between the two.

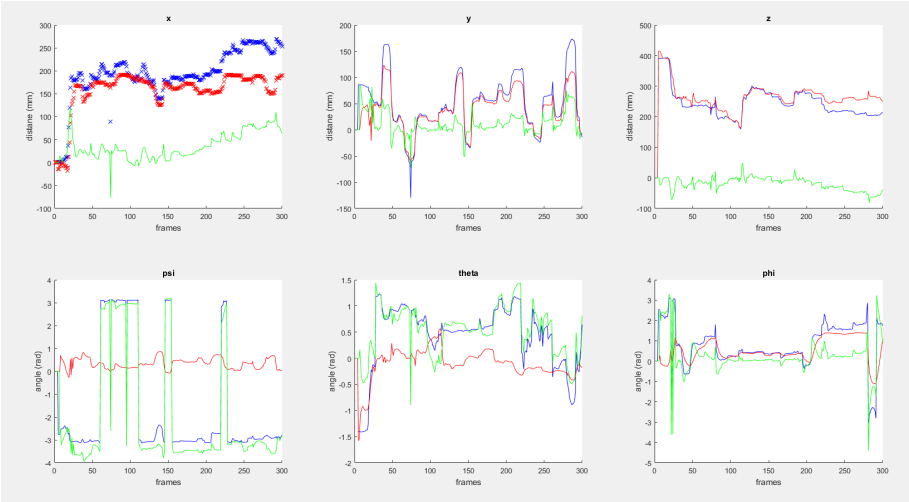

Figure 3.14: Measured position and orientation of the sixth measurements. Here the blue line is the targed, the red line is the end effector, and the green line is the error between the two.

Table 3.1: RMS values, and factor value for the measurements of test 2.

Measurement	x	y	z	ψ	θ	ϕ	factor
1 (figure 3.9)	$1.208 \cdot 10^3$	220.1	228.7	27.363	5.0753	20.199	1.0468
2 (figure 3.10)	475.98	172.66	105.44	27.980	3.8143	19.785	0.9216
3 (figure 3.11)	161.04	95.967	86.891	31.653	6.3265	10.730	0.9077
4 (figure 3.12)	549.72	374.67	277.45	29.423	12.163	10.705	0.9680
5 (figure 3.13)	269.79	169.66	347.04	13.018	8.850	10.310	0.8911
6 (figure 3.14)	787.64	426.23	484.44	55.81	12.105	16.787	0.9062

3.3.2 Discussing the results

The value of the orientation in the first measurement is measured to be (3.085; -0.06929; 2.659) while the actual orientation should have been (0; $(-\frac{3 \cdot \pi}{4})$; 0). This means that the orientation of the robot tends to a different value than the puppeteer desired. This error can be contributed to the what was determined during test one, which was that when the value for θ changes, so does the values for ψ, and ϕ in the measurements. The positioning part of the system seems to work as intended.

The RMS values, for the position, are lower in the second measurement than in the first measurement. This is because the factor in the second measurement is lower than it was during the first measurement, see table 3.1. Because of this the target point is of the same point in space, is placed closer to the origin in the scaled space, while the robot is unchanged in the scaled space. This means the robot can reach closer to the target point in measurement two, than it could in measurement one.

In the third measurement the z position of the robot is constantly lower than the target z position, this is caused by the fact that the robot also has to meet the target x and y position, which have a length more than the 16 mm that the robot can reach because of the placement of the second motor, so the robot has to bend a little, making its height less, this also effects its orientation.

The RMS value for the x position is better in the fifth experiment when compared to the second experiment, while the RMS value for y stays roughly the same, and the value for z increases. The reduction in the RMS value for the comes from the fact that the target is this time placed in a position that the end effector can actually reach. The increase in the RMS value for the z position seem to be caused by the fact that in the second experiment the end value for the z position was around 300 mm and in the fifth experiment this was around 200 mm, making the convergence to the value to take longer and the error at points larger. The rate at which the angles converges changes when the position of the end effector closes on the target position. Meaning that the position error weighs more than the orientation error.

The sixth measurement gives the best picture of how the system reacts to someone using it, since it has the puppeteer do whatever he likes. From this measurement it can be seen that the positioning of the end effector works as is intended. For the orientation no clear conclusion can be drawn as to the performance of it. This is caused by the error in the measured orientation measured in test one. What can be said about the orientation is that the value for ϕ for the end effector follows the same value for the target. The end effector value for ψ seem to be affected by the target orientation, since similar changes occur in both the end effector and target, look at frames 130 till 140.

3.4 Test three

3.4.1 Results

Figures 3.15 to 3.21 show the measurements, where 3.21 is the improvised routine. In these figures the red line represents the elevation angle of the arm, this is mapped to the second motor, the blue line is the turning angle of the arm, which is mapped to the first motor, the green line is the angle of the twist of the wrist, which is mapped to the fourth motor, and lastly the black line represents the bending angle of the wrist, which is mapped to the fifth motor.

The first measurement had the puppeteer start with his arm straight up, and move it flat on the table and back up again, while keeping everything else still. The angles that are calculated are shown in figure 3.15. This shows that the angle measured when the arm was in the up right position was: 1.092 rad and when the arm was completely flat on the table it was: 0.099 rad. The actual angles the arm had during this routine was 0 rad for the rotation of the arm, $-\frac{\pi}{2}$ rad for the twist in the wrist, and 0 rad for the bending of the wrist, while the bending of the arm was between 0.146 rad, and $\frac{\pi}{2}$ rad. The angle for the rotation is measured to be 0.420 rad, the angle with which the wrist is bend is measured to be 0. The angle that is the twist of the elbow changes during this routine in the same direction of the bending angle, except when its motor limit is reached, and when the elevation angle goes over 0.691 rad, when the twist angle changes in the opposite direction of the elevation angle.

The second measurement, shown in figure 3.16, had the puppeteer move his arm from left to right only changing the rotation angle of the arm. The rotation angle changes between the values of 0.698 rad (which is the motor limit of motor one) and -0.374 rad. The actual angles were 0.352 rad for the elevation angle, $-\frac{\pi}{2}$ rad for the twist, and 0 rad for the bending of the wrist. The measured angles for these were on average 0.305 rad for the elevation angles, and -0.065 rad for the bending of the wrist. The rotation angle made by the puppeteer was from $\frac{\pi}{4}$ rad to -0.3906 rad. Also here the angle that is the twist of the wrist changes, this time it changes in the same direction as the turning angles, but stops when it reaches its motor limit. The angle that is the bending of the wrist is mostly constant, except at two peaks. The elevation angle also changes a during this routine, although with less amplitude that the twist angle.

The third measurement, shown in figure 3.17, had the puppeteer rotate its wrist while he was holding his arm up right. The angle of the twist variated between $-\pi$ and 0 rad. The calculated twist angle changes in value between -0.328 rad and -1.396 rad. (Which is the motor limit of motor four.)The actual elevation angle was $\frac{\pi}{2}$ rad, the rotation angle 0 rad, and the wrist bending was 0 rad. The actual measured angles were 0.248 rad for the rotation angle, 1.162 rad on average for the elevation angle, and 0.049 rad on average for the bending of the wrist. The angles for the elevation, and the rotation of the arm variate little during this measurement. The angle for the elevation is however slowly decreasing, and it is higher than during the first routine, when the arm was also uptight at points. The angle of the bending of the wrist changes more than the other two angles during this measurement.

The fourth measurement, shown in figure 3.18, had the puppeteer bend his wrist, while keeping everything else still. The angle changed between 0.962 rad and -0.950 rad. The other angles were 0 rad for the turning angle, $\frac{\pi}{2}$ rad for the elevation angle, and $-\frac{\pi}{2}$ rad for the twist of the wrist. The measured angles were 0.214 rad for the rotation angle, 1.277 rad for the elevation angle, and the value for the twist of the wrist clipped against its motor limit at -1.396 rad. The angle for the bending of the wrist variated between 0.808 rad, and -0.799 rad. The results show that the change between the positive and negative angle is not a smooth transition in this case.

measurements five and six were the same as three and four, but with the difference that now the elevation angle is supposed to be $\frac{\pi}{4}$ rad. The results of this are shown in figures 3.19 and 3.20 respectfully. The measured values for the elevation angles were 0.700 rad and 0.863 rad respectfully.

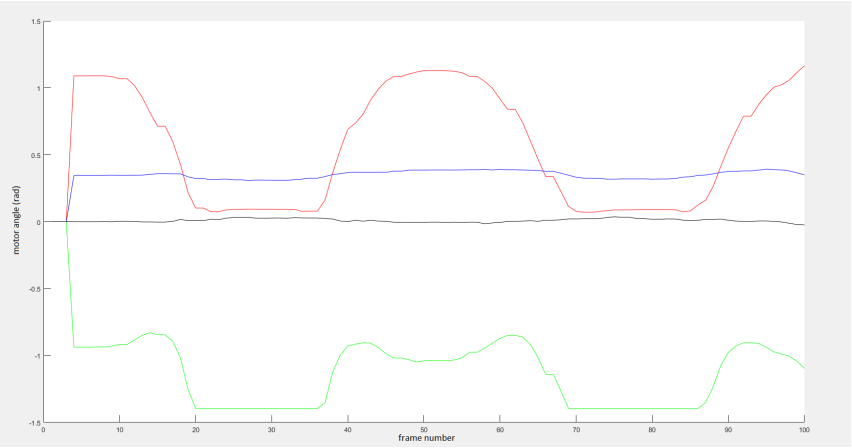

Figure 3.15: Measured angles of measurement one. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

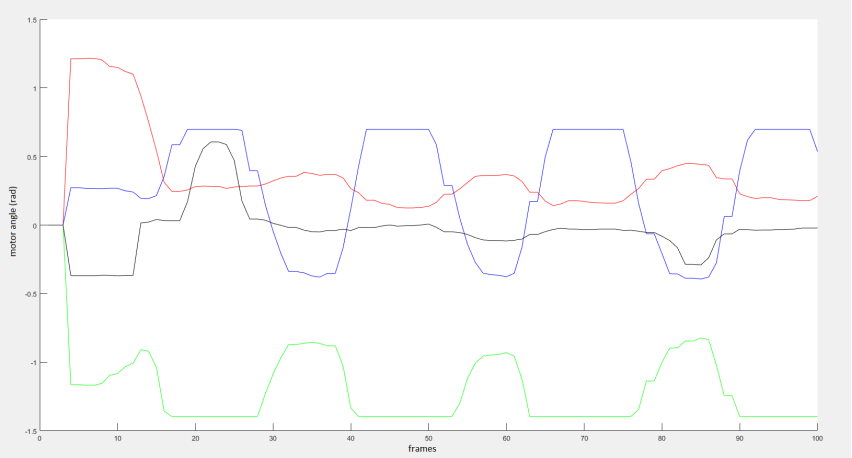

Figure 3.16: Measured angles of measurement two. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

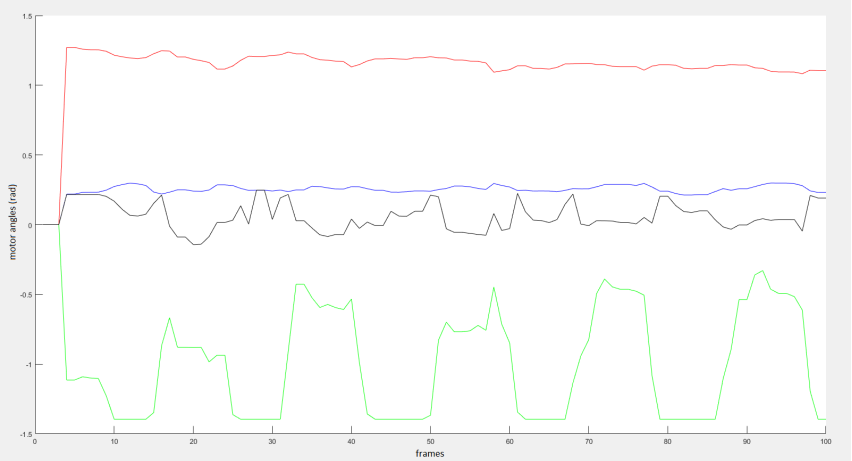

Figure 3.17: Measured angles of measurement three. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

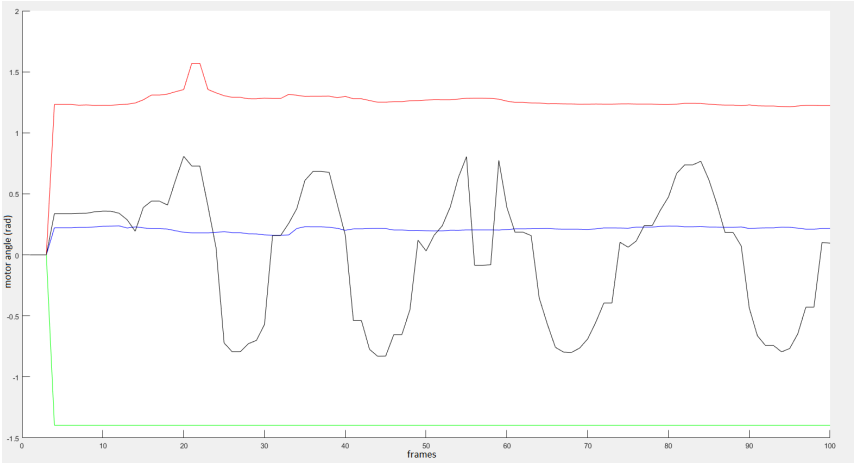

Figure 3.18: Measured angles of measurement four. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

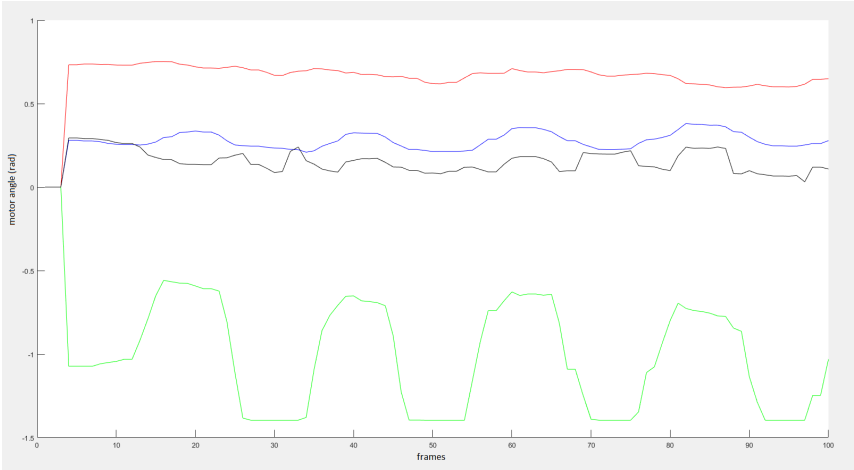

Figure 3.19: Measured angles of measurement five. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

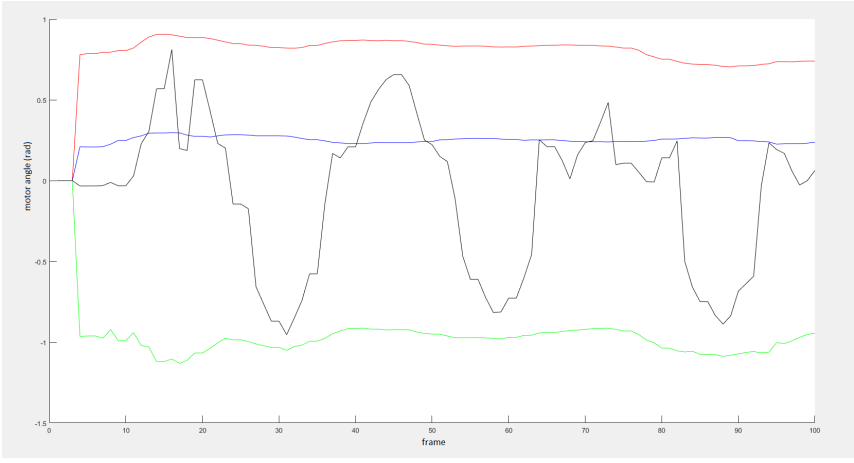

Figure 3.20: Measured angles of measurement six. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

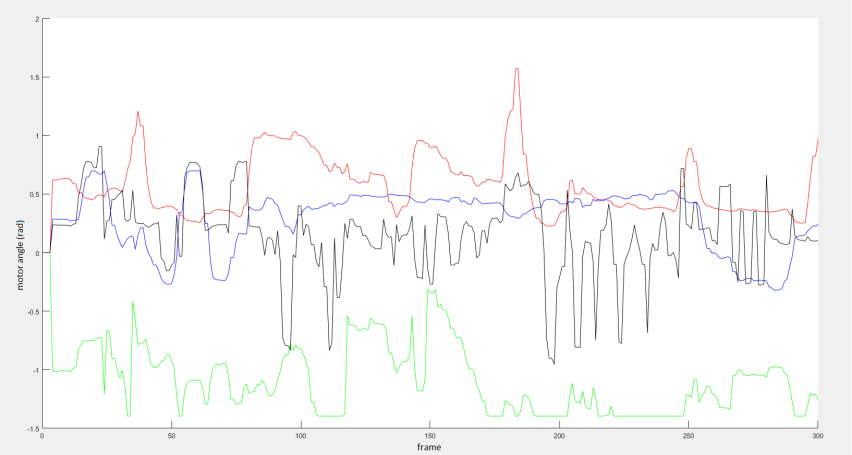

Figure 3.21: Measured angles of measurement seven. Here the red line is the elevation angle, the blue line is the rotation angle, the green line is the angle of the wrist twist, and the black line is the wrist bending angle.

3.4.2 Discussing the results

The results of the third test show that the elevation angle can be measured as the value is in the range of 0.099 rad to 1.277 rad with a deviation of ± 0.1 rad. The elevation angle changed when the rotation angle changed. This is explained by the fact that the markers are on the surface of the puppeteers arm and elbow. When the puppeteer moved his arm from left to right, the marker on the elbow tended to moved up and down with this motion. The marker moved up when the rotation angle was positive and down when it was negative, causing this change.

The rotation angle had a static deviation of 0.2 rad, double that during the first measurement. The transitions between positive to negative had no stutter, and the movement of this angle behaved as it should, al be it with a standard error. This error is explained by the fact that the markers are placed on the surface of the arm. The elbow is wider than the wrist is, so the marker of the wrist already has an angle with the marker on the elbow. Also during the rotational action the marker of the elbow moves, which lead to errors in the angle calculations.

The twist of the wrist is dependant on the elevation angle, and inversely on the rotation angle. This could be explained, by the fact that the transformation matrix uses the elevation, and rotation angles. These measured angles have some errors with the actual angles, so that the transformation is effected by this. The angle of the wrist does change when the wrist is changes, in the same direction as the twist.

The bending of the wrist does change how it is expected, but the change between what is positive and what is negative with varying efficiency. It works the best when the arm was held in an up right position, and when the arm was lowered the border between positive and negative becomes less efficient. This is best seen in figure 3.20, where the third time the angle becomes positive.

3.5 Comparing the methods

Looking at the results of both methods, both methods can be used to puppeteer the robot. In both methods the robot was being moved close to the places/angles the program calculated. Comparing the methods in terms of scalability can be done by looking at different aspects.

One aspect would the amount of markers that are needed. Looking from this angle it would mean that the end effector method would be more scalable to more complicated robots, since the same amount of markers are needed independent of the complexity of the robot. For the angle method more markers are needed for every joint in the human body that is going to be used and even more if the view of the marker can be obstructed.

An other way of looking at how scalable the methods are would be looking at the amount of calculations that need to be done. This would mean that the angle method would be the more scalable one. This is because every angle can be calculated with just one calculation for each angle and maybe one or two more calculations, depending on how the negatives are defined and if the calculated angle need to be changed in some way. The end effector method requires the Jacobian, inverse of the Jacobian, the error, and the motor angle for every single motor. Only the amount of calculations of the error, does not depend on the amount of motors here.

Looking at the speed the program can reach the end effector method would be more favourable for more complicated robots. This is because the angle method would need more markers when the robot becomes more complicated, this means the sensor can not output the same frame rate. For less complicated robots the angle method is faster since the calculations done by the angle method are faster than the calculations of the end effector method.

Looking at these aspects it can be seen that the angle method is better suited for the less complicated robots, while the end effector method is better suited for more complex ones.

3.6 Comunication to the robot

Using the MATLAB performance analyser the performance of both programs was investigated. The program of method one ran for a total time of 23.800 s. The majority of this time was caused by the serial communication between MATLAB and the robot, this serial communication took 22.514 s. The same was analysis was done for the second program. This program had a total run time of 23.650 s. Here again the majority of the time was spend in the serial communication, which took 21.794 s.

There was a noticeable delay between the moment the puppeteer moved to the moment the robot moved to match this movement. Also the motor angles of the robot and the motor angles MATLAB calculated were not the same.

The noticeable lag between the puppeteer and the robot, is caused by the speed of the serial communication between MATLAB and the robot. It is also possible for the error in the motor angles to come from the serial communication between the two, but this can not be concluded completely with the data available.

3.7 Sensor performance

The sensor ran at a frame rate of 120 frames per second. This was faster than the amount of frames MATLAB requested per second. This means that when MATLAB requests a new frame, a new frame was always available.

One obstacle encountered during the tests was that it was possible for the puppeteer to hide markers from the sensor during operation, by moving in such a way that a body part came between the sensor and the markers of one place. This meant that the sensor could not find these markers and this position could then not be used in the calculation of the angle it was involved in.

4 Conclusion

From the measurements of the target was able to give the position of the target within a range of 10 mm. The orientation was being calculated correctly if the target only rotated around its x and z axis, but when the target rotates around its y axis the calculation of the orientation was incorrect. This also means that whenever there is a rotation around the y axis of the target, the orientation can be calculated wrongly.

The end effector method works as intended in the position aspect of the end effector. The orientation aspect seem to work as intended, the ϕ angle of the end effector followed that of the target. All the angles converged towards the orientation of the target if they were able to. It can not be concluded if the orientation was doing what it was supposed to, because the orientation of the target can be calculated wrongly.

The angle method did not completely work the way it was intended. The elbow rotation and elevation angles did what they were supposed to do in the measurements. The rotation of the wrist did react to the wrist rotating, but it also reacted to the elbow rotation and elevation angles. This makes the control of the fourth motor difficult for the puppeteer. The wrist bending angle does calculate like it should, but the way the negative of the angles is defined makes the motor angle behave differently than the puppeteer does. This all means that the method can work correctly if the twist angles are calculated in a better way and if the negative of the wrist bending is implemented in a way that works better.

5 Recommendations

One of the main issues of the first method was that the twist angle of the wrist was dependent on the rotation and elevation angles. For this issue another method could be sought for. Another thing that could be investigated would be to see if a glove was made for the puppeteer, so that all the markers are always at the same position, and using the known position of the elbow marker with respect to the actual rotation point of the elbow, the transformation matrix can be improved do that these dependencies could be improved, or even fixed. Using such a glove the fact that the elbow marker moves as the puppeteer moves can also be improved, by calculating the actual turning point of the elbow and using this point in the calculations.

The target had problems when the orientation involved a change in θ. One thing to investigate would be to see if instead of extracting the values of ψ, θ, and ϕ from the rotation matrix, using the rotation matrix itself could solve this problem, or at least improve the performance.

In the second method, only one end effector is used in the inverse kinematics. It can be preferable, especially when the robot becomes more complex, to use multiple end effectors and place these at joint positions of the puppeteer. This could help the robot to more closely mimic the puppeteer. Using multiple end effectors it would be possible to only use the positions of these end effectors, meaning the error in the orientation detection of the target would be less of an issue.

The communication between MATLAB and the robot need improving, since this slows the whole system down and introduces lag. One way of doing this would be to use an input buffer for the robot and increase the speed at which the robot runs its program, which was set to 5Hz in this paper.

A Transformation matrix

$$H^0_{5_{1,1}} = cos(q_1)\cdot cos(q_2)\cdot cos(q_3)\cdot cos(q_5) - cos(q_1)\cdot sin(q_2)\cdot sin(q_3)\cdot cos(q_5) - sin(q_1)\cdot sin(q_4)\cdot sin(q_5) - cos(q_1)\cdot cos(q_2)\cdot sin(q_3)\cdot cos(q_4)\cdot sin(q_5) - cos(q_1)\cdot sin(q_2)\cdot cos(q_3)\cdot cos(q_4)\cdot sin(q_5);$$

$$H^0_{5_{1,2}} = -sin(q_1)\cdot cos(q_4) + cos(q_1)\cdot cos(q_2)\cdot sin(q_3)\cdot sin(q_4) + cos(q_1)\cdot sin(q_2)\cdot cos(q_3)\cdot sin(q_4);$$

$$H^0_{5_{1,3}} = cos(q_1)\cdot cos(q_2)\cdot cos(q_3)\cdot sin(q_5) - cos(q_1)\cdot sin(q_2)\cdot sin(q_3)\cdot sin(q_5) + sin(q_1)\cdot sin(q_4)\cdot cos(q_5) + cos(q_1)\cdot cos(q_2)\cdot sin(q_3)\cdot cos(q_4)\cdot cos(q_5) + cos(q_1)\cdot sin(q_2)\cdot cos(q_3)\cdot cos(q_4)\cdot cos(q_5);$$

$$H^0_{5_{1,4}} = (cos(q_1)\cdot cos(q_2)\cdot cos(q_3) - cos(q_1)\cdot sin(q_2)\cdot sin(q_3))\cdot(r5(1)+r4(1)) - sin(q_1)\cdot(cos(q_4)\cdot r5(2) - sin(q_4)\cdot r5(3)+r4(2)) + (cos(q_1)\cdot cos(q_2)\cdot sin(q_3) + cos(q_1)\cdot sin(q_2)\cdot cos(q_3))\cdot(sin(q_4)\cdot r5(2) + cos(q_4)\cdot r5(3)+r4(3)) + cos(q_1)\cdot cos(q_2)\cdot r3(1) - sin(q_1)\cdot r3(2) + cos(q_1)\cdot sin(q_2)\cdot r3(3) + cos(q_1)\cdot r2(1) - sin(q_1)\cdot r2(2) + r1(1);$$

$$H^0_{5_{2,1}} = sin(q_1)\cdot cos(q_2)\cdot cos(q_3)\cdot cos(q_5) - sin(q_1)\cdot sin(q_2)\cdot sin(q_3)\cdot cos(q_5) + cos(q_1)\cdot sin(q_4)\cdot sin(q_5) - sin(q_1)\cdot cos(q_2)\cdot sin(q_3)\cdot cos(q_4)\cdot sin(q_5) - sin(q_1)\cdot sin(q_2)\cdot cos(q_3)\cdot cos(q_4)\cdot sin(q_5);$$

$$H^0_{5_{2,2}} = cos(q_1)\cdot cos(q_4) + sin(q_1)\cdot cos(q_2)\cdot sin(q_3)\cdot sin(q_4) + sin(q_1)\cdot sin(q_2)\cdot cos(q_3)\cdot sin(q_4);$$

$$H^0_{5_{2,3}} = sin(q_1)\cdot cos(q_2)\cdot cos(q_3)\cdot sin(q_5) - sin(q_1)\cdot sin(q_2)\cdot sin(q_3)\cdot sin(q_5) - cos(q_1)\cdot sin(q_4)\cdot cos(q_5) + sin(q_1)\cdot cos(q_2)\cdot sin(q_3)\cdot cos(q_4)\cdot cos(q_5) + sin(q_1)\cdot sin(q_2)\cdot cos(q_3)\cdot cos(q_4)\cdot cos(q_5);$$

$$H^0_{5_{2,4}} = (sin(q_1)\cdot cos(q_2)\cdot cos(q_3) - sin(q_1)\cdot sin(q_2)\cdot sin(q_3))\cdot(r5(1)+r4(1)) + cos(q_1)\cdot(cos(q_4)\cdot r5(2) - sin(q_4)\cdot r5(3)+r4(2)) + (sin(q_1)\cdot cos(q_2)\cdot sin(q_3) + sin(q_1)\cdot sin(q_2)\cdot cos(q_3))\cdot(sin(q_4)\cdot r5(2) + cos(q_4)\cdot r5(3)+r4(3)) + sin(q_1)\cdot cos(q_2)\cdot r3(1) + cos(q_1)\cdot r3(2) + sin(q_1)\cdot sin(q_2)\cdot r3(3) + sin(q_1)\cdot r2(1) + cos(q_1)\cdot r2(2) + r1(2);$$

$$H^0_{5_{3,1}} = -sin(q_2)\cdot cos(q_3)\cdot cos(q_5) - cos(q_2)\cdot sin(q_3)\cdot cos(q_5) + sin(q_2)\cdot sin(q_3)\cdot cos(q_4)\cdot sin(q_5) - cos(q_2)\cdot cos(q_3)\cdot cos(q_4)\cdot sin(q_5);$$

$$H^0_{5_{3,2}} = -sin(q_2)\cdot sin(q_3)\cdot sin(q_4) + cos(q_2)\cdot cos(q_3)\cdot sin(q_4);$$

$$H^0_{5_{3,3}} = -sin(q_2)\cdot cos(q_3)\cdot sin(q_5) - cos(q_2)\cdot sin(q_3)\cdot sin(q_5) - sin(q_2)\cdot sin(q_3)\cdot cos(q_4)\cdot cos(q_5) + cos(q_2)\cdot cos(q_3)\cdot cos(q_4)\cdot cos(q_5);$$

$$H^0_{5_{3,4}} = (-sin(q_2)\cdot cos(q_3) - cos(q_2)\cdot sin(q_3))\cdot(r5(1)+r4(1)) + (-sin(q_2)\cdot sin(q_3) + cos(q_2)\cdot cos(q_3))\cdot(sin(q_4)\cdot r5(2) + cos(q_4)\cdot r5(3)+r4(3)) - sin(q_2)\cdot r3(1) + cos(q_2)\cdot r3(3) + r1(3) + r2(3);$$

$$H^0_{5_{4,1}} = 0$$
$$H^0_{5_{4,2}} = 0$$
$$H^0_{5_{4,3}} = 0$$
$$H^0_{5_{4,4}} = 1$$

Bibliography

[1] R. Slyper, G. Hoffman, A. Shamir,, "Mirror puppeteering: Animating toy robots in front of a webcam," 2015. [Online]. Available: http://guyhoffman.com/publications/SlyperTEI15.pdf

[2] H. Reddivari, C. Yang, Z. Ju, P.Liang, Z.Li and B.Xu. (2014) Teleoperation control of baxter robot using body motion tracking. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6997722

[3] Phoenix Thechnologies Incorporated, "Visualeyez user manual," pp. 58–59, 2004.

[4] E.Dertien, "Desklight," last visited 26-06-2018. [Online]. Available: http://wiki.edwindertien.nl/doku.php?id=projects:desklight

[5] B.Siciliano, "Industrial robotics," page 16. [Online]. Available: http://www.master-ris.unina.it/custom-php/materiale_didattico/bruno_siciliano_1297463044.pdf

[6] MathWorks, "atan2," last visited 26-6-2018. [Online]. Available: https://nl.mathworks.com/help/fixedpoint/ref/atan2.html

[7] S.Stramigioli, "Multibody dynamics & control." [Online]. Available: https://blackboard.utwente.nl/bbcswebdav/pid-981319-dt-content-rid-2219104_2/courses/2016-201500061-1A/MBDC-HC3%281%29.pdf

[8] MathWorks. Matrix rotations and transformations. Last visited 26-06-2018. [Online]. Available: https://nl.mathworks.com/help/symbolic/examples/rotation-matrix-and-transformation-matrix.html

[9] A. Aristidou and J. Lasenby, "Inverse kinematics: a review of existing techniques and introduction of a new fast iterative solver," pp. 11–16. [Online]. Available: http://www.andreasaristidou.com/publications/papers/CUEDF-INFENG,%20TR-632.pdf

[10] S.Stramigioli. Multibody dynamics & control. [Online]. Available: https://blackboard.utwente.nl/bbcswebdav/pid-976690-dt-content-rid-2181424_2/courses/2016-201500061-1A/MBDC-HC1.pdf