Department of Engineering
Aarhus Universitet

Robotics

Course Project - Group 1

Robotic automation of pressure check of cooler tubes

14. January 2022

Course Instructor: Xuping Zhang
Authors:

202102198 - Antonio Karlović
202102200 - Petar Gavran
201701953 - Emil Behrens Hansen
201711539 - Jesper Mørch Pedersen
202100761 - Sara Passerini

AARHUS UNIVERSITET

Contents

1 Introduction

In the course of this project, a process was automized that until now has been conducted manually. The task involves testing whether the tubes of a heat exchanger are installed and sealed properly or if they have a leakage. At the moment, this process is managed by an operator who inserts a tube tester in each tube of the heat exchanger, pressurizes it, and holds for a few seconds to check if the pressure remains constant inside the tubes. These heat exchangers have a great number of tubes, so the task of doing it manually is very time consuming and uninteresting to the worker. Therefore our objective was to implement this process on a robot in order to make it faster. The company which presented the project is Vestas Aircoil. Vestas Aircoil was founded in the Danish town of Lem in 1956 and it produces heat exchangers mainly for Diesel engines.

1.1 Introduction to UR5e

The project will use the UR5e robot. The UR5e by Universal Robot is a light collaborative robot thought for medium load applications (up to 5 kg). This robot is adapted for many uses thanks to its great versatility and adaptability. The UR5e collaborative robot by Universal Robots is a newer version of the original UR5. With similar specifications as the UR5, this robot has been improved by adding a Force Torque Sensor in Axis 6 of the robot. The other benefit of this newer UR5e robot is an improvement in programming, navigating, accuracy. Set up time is minimized with its easy to install design and drag-and-drop programming style. End of arm tooling options gives this robot flexibility to handle a large variety of manufacturing tasks.

1.2 Introduction to the cooler

The cooling system is a tube-fin heat exchanger mainly for Diesel engines. The tubes are attached with fins and are placed within a hot gas to cool them. A coolant passes through the tubes, which is responsible for extracting the heat from the fins and the tubes and transporting it outside of the system. To obtain a good performance the coolant is pressurized inside the tubes, therefore, if the tubes are not sealed correctly, the coolant would leak from the system.

1.3 Description of test procedure

Every tube needs to be tested whether or not it's installed correctly. In order to do so, a tube tester is used to pressure each tube and check if there are any leakages. In brief, the procedure consists in inserting the tube tester in each tube, applying a pneumatic pressure, and holding for a few seconds. If the pressure remains constant and does not drop, the tube is deemed tight. If not, there is a leak that can be identified and fixed. The tube tester is a piece of equipment provided by Vestas Aircoil and is under patent-pending, it will be used during this project, however cannot be described in full detail.

2 | Problem formulation

This project aims to automate the process of pressure testing each tube in the cooler. The project will be utilizing the UR5e robot because of its good accuracy and the build-in force sensor.
Multiple challenges and aspects of the task present themselves when defining the problem. The provided tube tester needs a tight fit with each tube, to be able to operate properly. This sets big demands for the precision of the robot, the placement of the cooler, and tolerances when manufacturing the cooler, if the process was to rely on hard-coded coordinates alone. The coolers tested have the same size, and the process does not need to involve automatic change between different size tube testers.
When automation of the process is developed, it's desired to automatically inform the worker of which tubes are problematic. These demands can in order to succeed, be summarized in the points below, divided into different priority groups.

Necessary

- Design of a simple gripper, whose purpose is to permanently fasten the tube tester to the end effector of the UR5e, with an accompanying camera to intelligently align the tube tester with a cooler tube.
- Describe the basic robotics concepts, theories, and techniques relevant in an automated process application.
- Develop a system to automatically do the pressure test procedure, and record the results.
- Develop a program controlling the basic movement of the robot and collecting other aspects of the task, showcasing the automated process on a downscaled cooler provided by Vestas Aircoil.

Desirable

- Implementation of visual recognition to achieve precise alignment between the tube tester and each individual tube in the cooler.
- Automated localization of the placement of the cooler.
- Simulation of the automated process in a computer-controlled environment using Robo DK.
- User-friendly interface to see the results of the automated process
- Collection of the different parts: robot movement, cooler localization, visual recognition, pressure testing, and result presentation, into one program.

3 | Gripper

In this section, the design of the gripper design will be described. The gripper used in this project is atypical in the sense that it doesn't have any physically moving parts controlled, as the task doesn't require the gripper to grip anything. The only moving part is in the tube tester and moved through friction contact. Furthermore, the tube tester doesn't need to be switched and can be permanently mounted.
The gripper however still needs a pneumatic system, as it needs to automatically apply pressure to do the task.
The tool is also equipped with a camera to account for small miss placement of the cooler and general production tolerances. This is needed as the tube tester requires a tight fit with the cooler hole to properly function.

3.1 Tube Tester and adaptor

Since the Tube Tester is a patent of Vestas Aircoil, who provided it to us, an accurate description and method of operation were not provided, so a rough description of the Tube Tester and how we assumed it works will be described.

The assumed mode of operation of the Tube Tester is that when inserted into the test tube, by applying air the pressure in the tube increases, the component in the body separates during friction and releases air into the tube, after closing the air the component returns to place and closes the Tube Tester, thus allowing the pressure in the pipe to be tested and to see if there are any leaks. The CAD drawing of the Tube Tester made by reverse engineering is shown in the fig. 3.1 below.

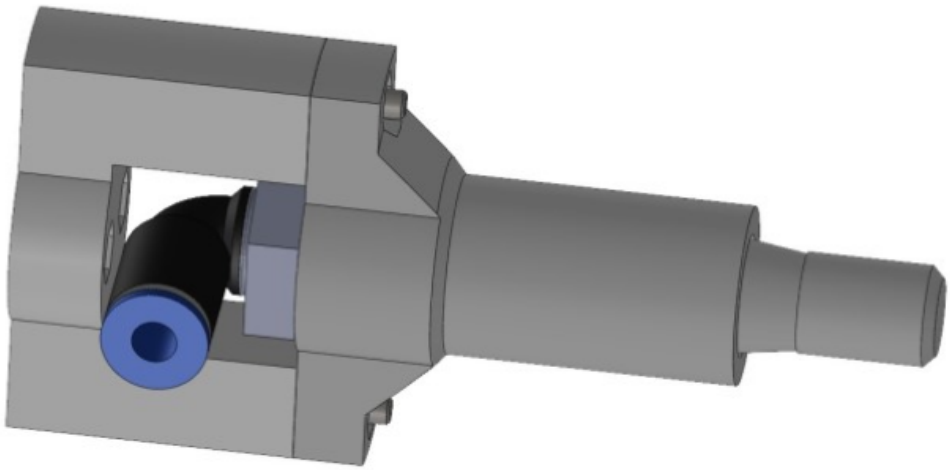


Figure 3.1. Tube Tester

A discussion with Vestas Aircoil clarified that there was no need for switching between different tube testers during the automated process. Therefore the gripper simply needs to permanently fix the tube tester to the robot end-effector, through a simple stiff adaptor pictured on figure 3.2.

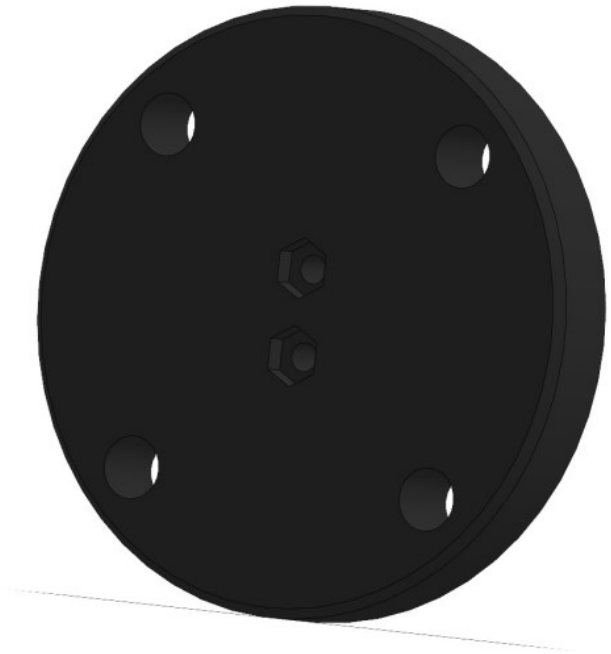


Figure 3.2. Adaptor

3.2 Pneumatic System

In the testing process, described in section 1.3 on page 2, the tube tester is inserted into the tube, and pressure is applied to determine if the tube is sealed correctly. In order to detect if the tube is leaking it is needed to separate the pressure in the tubes from the external input pressure, and hold the pressure for a time period. If the air pressure drops below a threshold value there is a leak in the system. In the pneumatic system, the following components are used.

- Pressure sensor 1-10 bar of the type 3100S0010G01B000RS from WIKA.
- Two 3-way valves of the type 5794600220 from Aventics.
- Microcontroller Arduino Mega 2560.
- An pressure supply in the range of 1-6 bar.
- A pressure limiter of the max 15 bar from Prevost.
- Pneumatic rectifier.

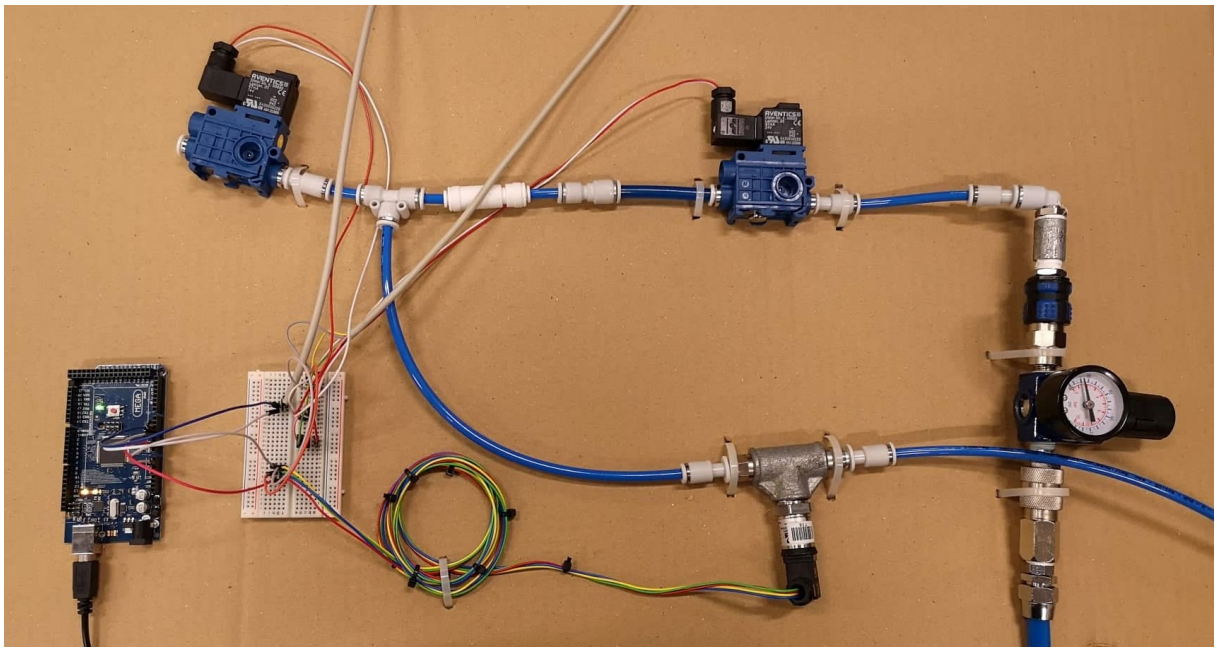


Figure 3.3. Pneumatic system

The pressure supply is connected to the pressure limiter. The limiter is adjusted to a value below the unadjusted pressure level to obtain a stable independent reference pressure. From the pressure limiter, the tube is connected to the first valve. The valve is controlled by an on/off signal, which results in the pressure being blocked or connected to the pressure limiter. The two stages from the other side of the valve correspond to connected to the input pressure, or closed for the input pressure and connected to the exhaust of the valve.

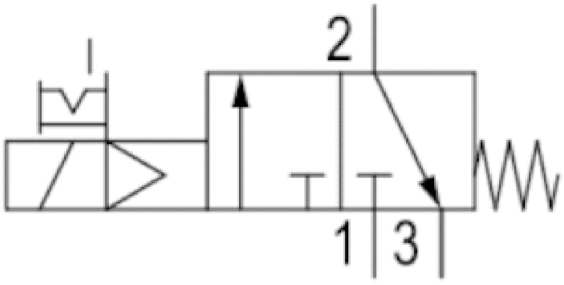


Figure 3.4. Valve diagram

When the first valve is open, the output port is connected to the rectifier input side. The rectifier's other end is connected to a T connector. One end of the T connector is connected to the tube tester/pressure sensor, and the other end is connected to valve number two. Valve number two is used for exhausting the tube tester when the test procedure is done.

The system is automated and therefore should be controlled by the robotics system. The robotics system is connected to the valves for the stage switching according to the steps

in the program. The Arduino is used for collecting the data from the pressure sensor. For knowing when the test procedure is in progress, the robotics system is connected to the Arduino. The robotics system sends a signal to the Arduino when the testing is in progress.

The pressure sensor output is a voltage signal between 0-10V. The output channel is connected to the A0 input port of the Arduino. The analog signal is converted to a digital signal with a 10-bit resolution and sends it to the computer. You can convert the reading to a pressure reading the following equation.

$$x_{A/D\ reading} \cdot \frac{10\ \text{bar} - 1\ \text{bar}}{2^{10}} = x_{bar}$$

The electronic used in the pneumatic system is shown in fig. 3.5.

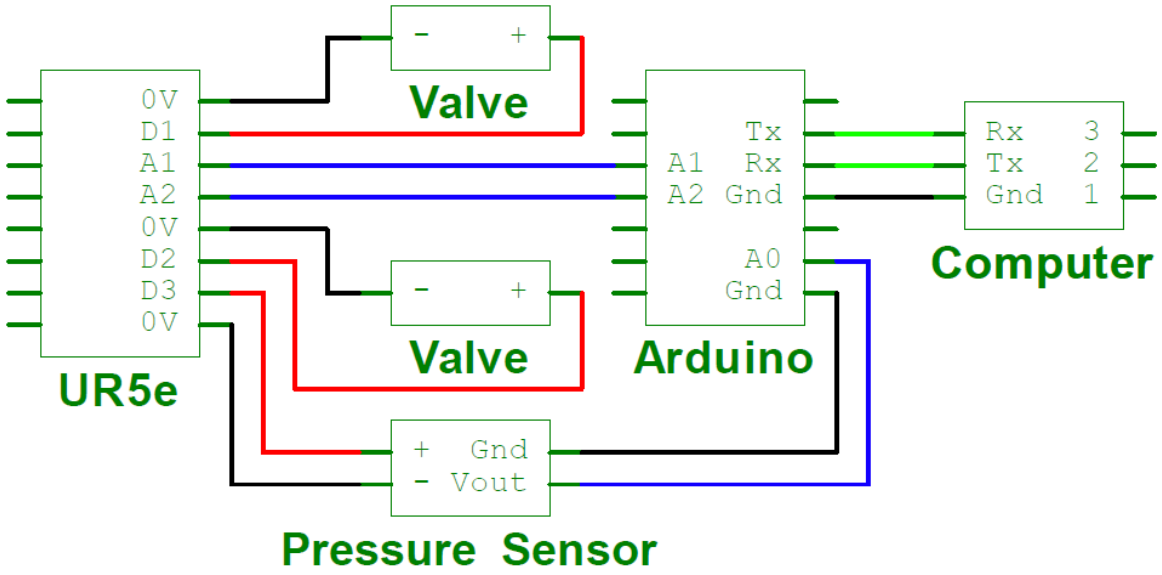


Figure 3.5. Electrical diagram for pneumatic system

3.3 Camera

To use visual recognition a camera needs to be mounted as part of the gripper. We are using a 2D camera of good resolution, provided by teaching assistant, Xingyu Yang. The camera is simple and seen without casing in figure 3.6. The connection to the computer is made through a USB to USB-Micro-B cable.

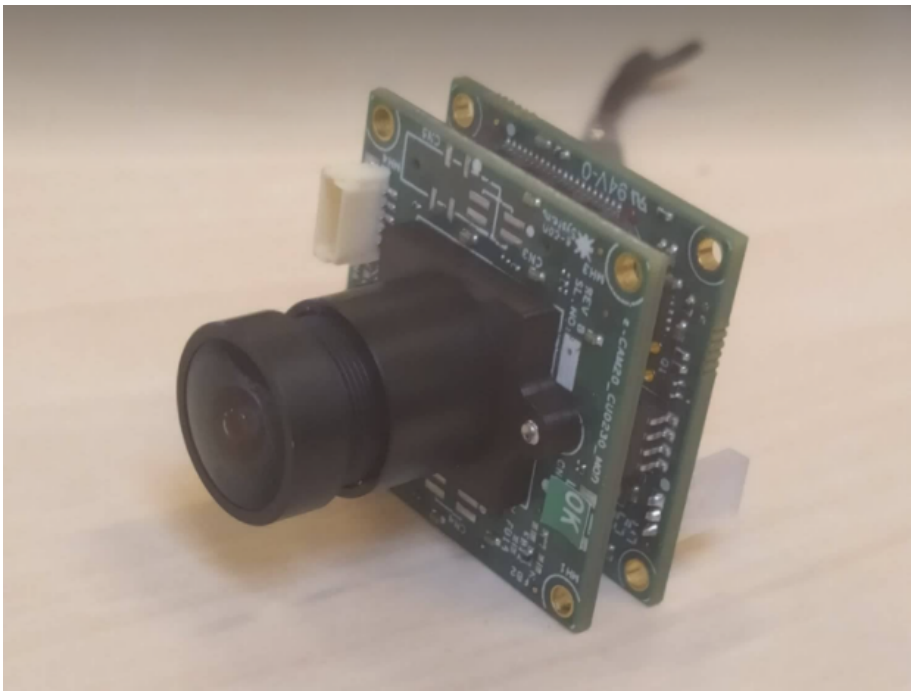


Figure 3.6. Camera used to implement visual recognition

A casing was designed and 3d-printed to protect and mount the camera. It was made so it was possible to adjust the focus of the camera lens while being in the case. In figure 3.7 the two parts which together make up the casing are shown.

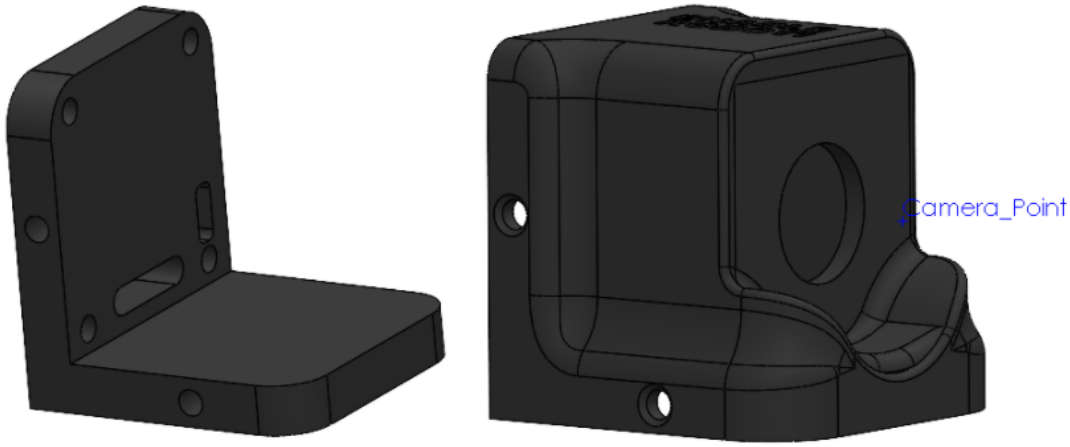


Figure 3.7. Case for the 2d camera

The camera is held in place above the tube tester so it's in the XY-plane of the tool frame. It's mounted on a bend aluminium piece which is held in place between the adaptor for the tube tester and the end of the UR5e robot. The aluminium is used for better stiffness than a 3d-printed option would provide, as it's imported the camera stays accurately in the same position. In figure 3.8 in the next section, the combined assembly of the gripper is pictured

3.4 Conclusion

The work done on the individual parts of the gripper leads to a finished gripper seen in figure 3.8, with an associated pneumatic system capable of applying pressure and measuring if a leak is present. The gripper as a whole is rigid so the camera stays in the same relative spot throughout the whole procedure.

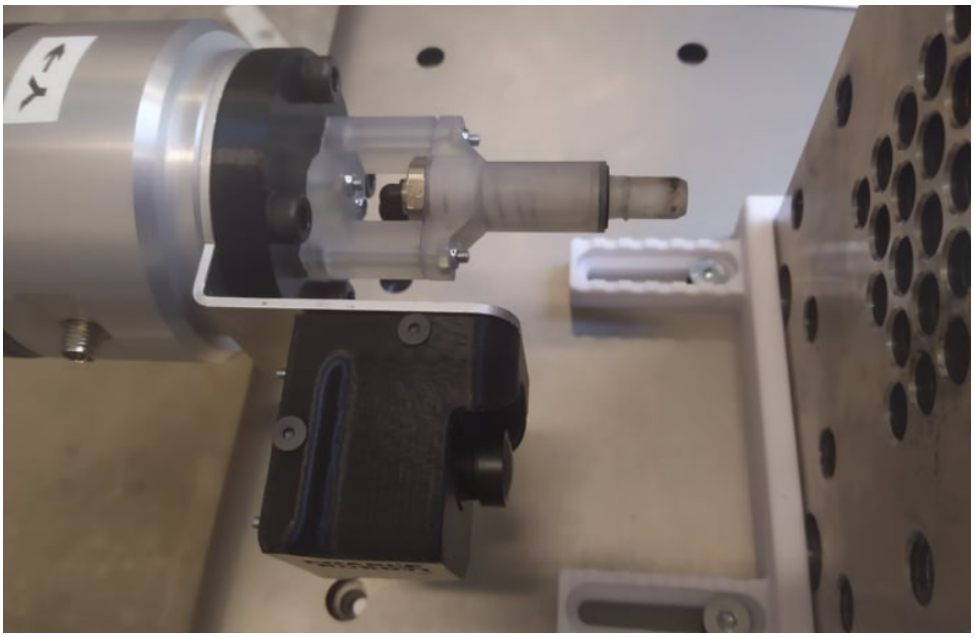


Figure 3.8. The full assembly of the gripper mounted on the UR5e

4 | Frame attachment and physical parameters

4.1 Frames of UR5e

With the UR5e robot positioned in the zero position, the frames can be identified. The frames for our robotic system can be seen in figure fig. 4.1

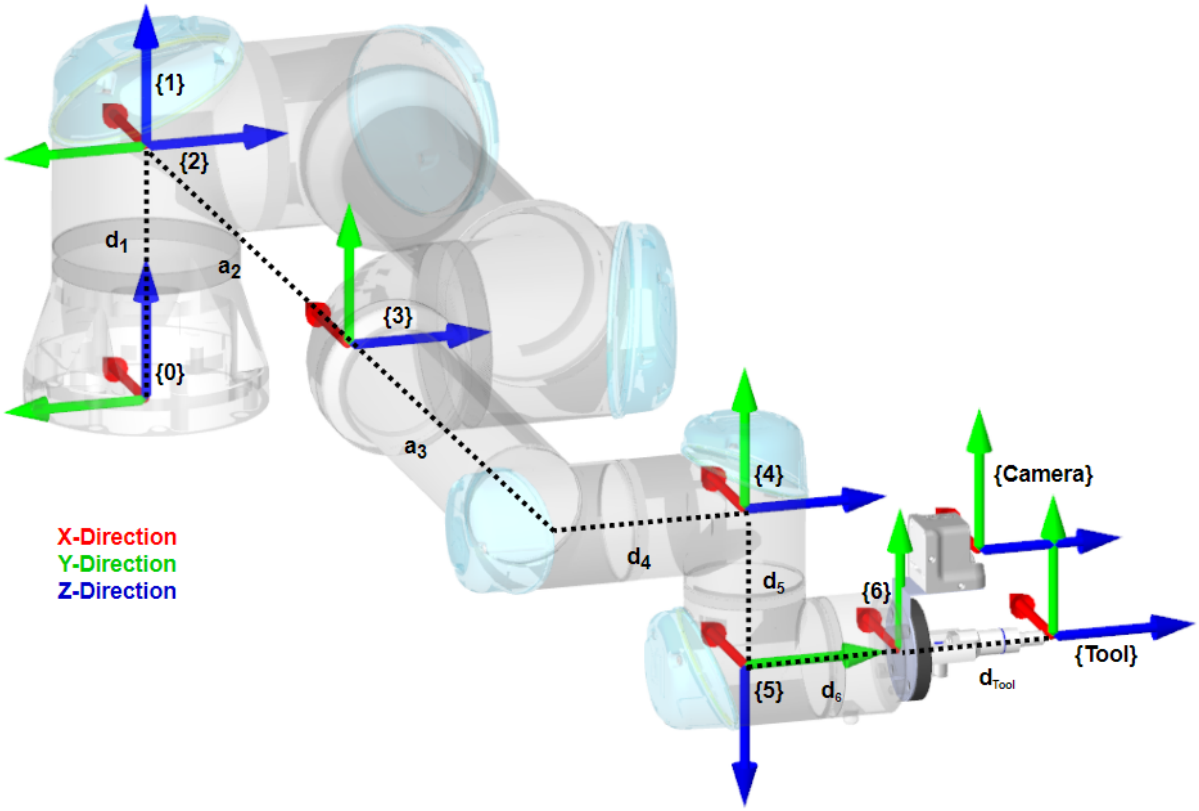


Figure 4.1. Frames attached to the UR5e with DH-parameters visualized

A reference coordinate system is identified for each joint, in order to label the link parameters. Thus, there are seven frames from 0 to 6, one frame for the tool and one for the camera.

4.2 Link Parameters

A link is considered only as a rigid body that connects two neighboring joint axes of a manipulator. In order to define a link, the following parameters have to be considered:

1. Link length $a_{(i-1)}$: The distance between two axis (i-1) and i.
2. Link twist $\alpha_{(i-1)}$: The angle measured from the axis (i-1) to the axis i in the right-hand sense about $a_{(i-1)}$.
3. Link offset d_i: The distance from link (i-1) to link i at the common joint i.
4. Joint angle θ_i: The angle from $a_{(i-1)}$ along the axis of joint i.

For each link, a mass, a center of mass, and inertia tensors can be identified.

In table 4.1 the parameters of each link is presented.

Link	$\alpha_{(i-1)}$	$a_{(i-1)}$	d_i	θ_i	$Mass_i$ [Kg]	$Center\ of\ Mass_i$ [mm]
From 0 to 1	0	0	d_1	θ_1	3.761	[0; 1.93; -25.61]
From 1 to 2	90°	0	0	θ_2	8.058	[-212.5; 0; 113.36]
From 2 to 3	0	a_2	0	θ_3	2.846	[-242.2; 0; 26.5]
From 3 to 4	0	a_3	d_4	θ_4	1.37	[0; -16.34; -1.8]
From 4 to 5	90°	0	d_5	θ_5	1.3	[0; 16.34; -1.8]
From 5 to 6	-90°	0	d_6	θ_6	0.3	[0; 0; -1.159]

Table 4.1. Link Parameters

The center of the end of the tube tester on the gripper is located on the z-axis for frame 6. Therefore the last frame can be moved to instead represent the end of the gripper by adding the length of the gripper, d_{Tool}, to the d_6 parameter in table 4.1.

$$d_{Tool} = 0.07m \tag{4.1}$$

For what concerns the inertia tensors they can be collected in the following matrix for each link

$$I = \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

The main diagonal contains the mass moment of inertia, which is always positive, the other values are the mass products of inertia, which are zero for a solid, symmetric about all three axes. In the project, inertia tensors are assumed to be symmetric, so the mass products of inertia are equal to zero. The Values used for the mass moment of inertia are not correct since they are taken from a step file of the UR5e, which is missing the gears and motors in each joint.

4.3 DH-Parameters

The DH parameters is the table containing the link parameters for each joint. In our case of UR5e robot the DH parameters are collected in table 4.2

i	$\alpha_{(i-1)}$	$a_{(i-1)}$ [m]	d_i [m]	θ_i
1	0	0	0.1625	θ_1
2	90°	0	0	θ_2
3	0	-0.425	0	θ_3
4	0	-0.3922	0.1333	θ_4
5	90°	0	0.0997	θ_5
6	-90°	0	0.0996	θ_6

Table 4.2. DH-Parameters

4.4 Conclusion

It is important to individuate the frames of the UR Robot in order to define the position of each joint and to get the forward kinematics, inverse kinematics and dynamics as a consequence. In particular, the DH parameters are used to represent the pose (position and orientation) of one body with respect to another with the transformation matrix.

5 | Programming of Robot

The different parts of this project has been developed in different programming languages and programs, depending on what was the most obvious for each task. All different parts was then collected in Polyscope. In figure 5.1 is an overview of the full program as a flow chart, and the txt. file from Polyscope is in Appendix A. The flowchart starts after the gripper is attached, the cooler placed, the pneumatic system is set up, and the camera is calibrated. The different parts of the program will be further explained through this chapter.

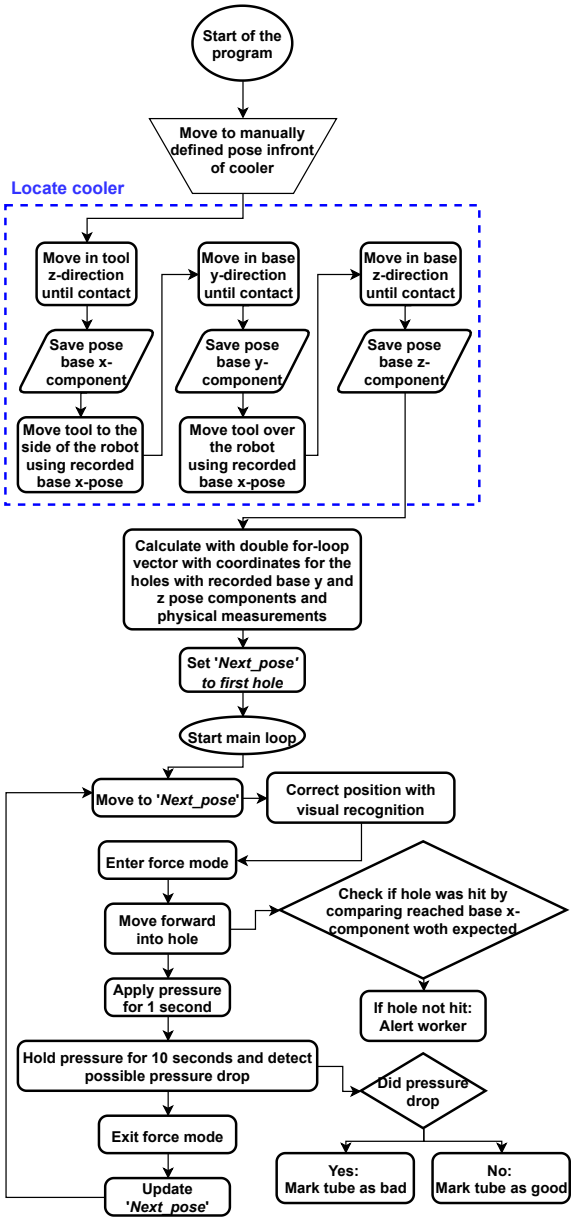


Figure 5.1. Process Flow Chart

5.1 Movement of Robot

Programming of the robot movement is achieved using PolyScope at teach pendant. PolyScope is the graphical user interface which lets you operate the robot arm, execute robot programs, and create new ones. Three possible basic orders that the robot can be moved are moveJ, moveL and moveP.

- **moveJ** - movement calculated in joint space of the robot arm. Movement results in a curved path for the tool
- **moveL** - is used for moving the tool linearly between waypoints. It results in more complicated motion of each joint to keep the tool on a straight-line path
- **moveP** - moves the tool linearly with constant speed with circular blends, and is intended for some process operations

List of the features and commands used, to understand the program: If statement, while loop, waypoint, relative waypoint, variable waypoint, wait, force mode...

5.1.1 Independent of Cooler Placement

At the beginning, the robot is unaware of the cooler exact position, only an approximate position is known. Robot initial movement is guiding robot approximately in front of the cooler, this movement must be programmed manually. This is considered only manual movement, everything after this is done in a way where moves are dependent of the previous movement, using relative movements and variables.

To automatically find the position of the cooler, a function called 'Direction' is used. In this function the robot can be programmed to move in a linear direction relative to the tool, until contact is detected through the build in force sensor. This is firstly done from the front to locate the cooler plane and save the planes base x-component by utilizing forward kinematics. Hereafter the robot just needs to locate the top and the side through the same procedure.

5.1.2 Force Mode

One of the challenges was to imitate the pressure exerted by the hand of the operator while holding the air pressure gun. Unlike the operator, the robot cannot stop moving 'by the feel', but it must be programmed to a specific procedure. This is achieved using a command called: 'Force Mode' with the Polyscope command window shown in figure 5.2. When entering 'Force Mode' the robot can move in a defined direction until a certain force

is reached. Usually and in this project the 'Simple Mode' is used, where movement and force measurement is done on one axis, in this case the tool z-axis. To secure a tight seal the robot inserts the tube tester into the hole and pushed with a force of 15 N. This value is calculated by a simple force analysis, where the tube tester has to push back against the build up air pressure, and still seal tight. With a pressure of, $p = 1.5bar$, and tube tester diameter of $d = 9mm$, the force exerted by the air pressure on the tube tester can be calculated as in equation 5.1.

$$\begin{aligned} A &= \frac{\pi}{4} \cdot d^2 = 63.6mm^2 \\ F &= p \cdot A = 9.5N \end{aligned} \tag{5.1}$$

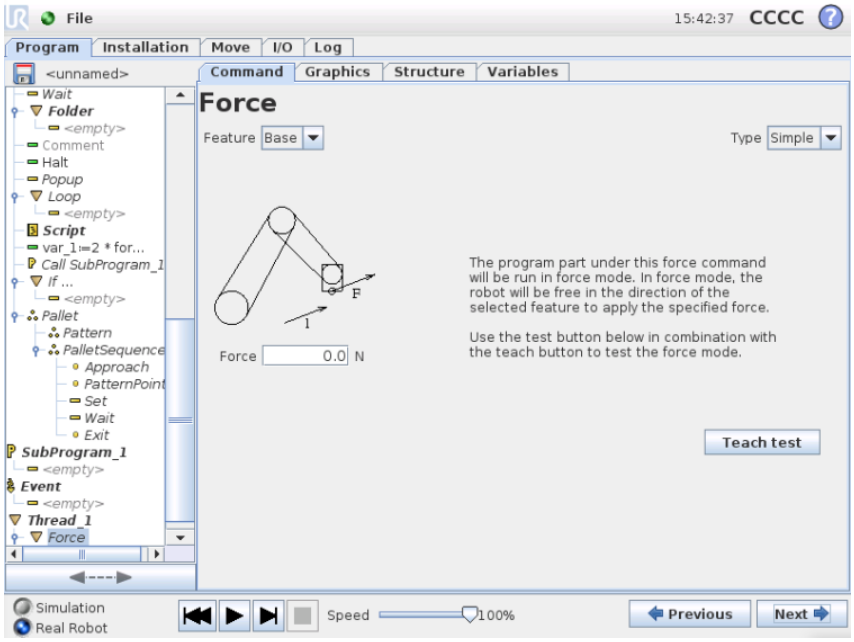


Figure 5.2. Force Mode Menu

5.1.3 Programming of the pneumatic system

The pneumatic system is used to detect if the tested hole is leaking under the pressure test. The system is operated if the following steps for each hole. When a tight seal is achieved by the force control, the system is pressurized by opening the first valve. Then the program waits one second to obtain a stable pressure. Then the first valve is closed, and the pressure is held for 10 sec. If the pressure drops below a threshold value, a leak is detected. After the pressure is held for 10 sec the pressure is exhausted and the robot move to the next hole.

In figure fig. 5.3 on the next page the output is shown in the red curve as a 10-bit output format. The blue curve is the reference for when the detection is evaluated. It can be seen

in the figure, the pressure is leaking. The cooler isn't sealed in the back, which therefore makes the leaking off all holes expected.

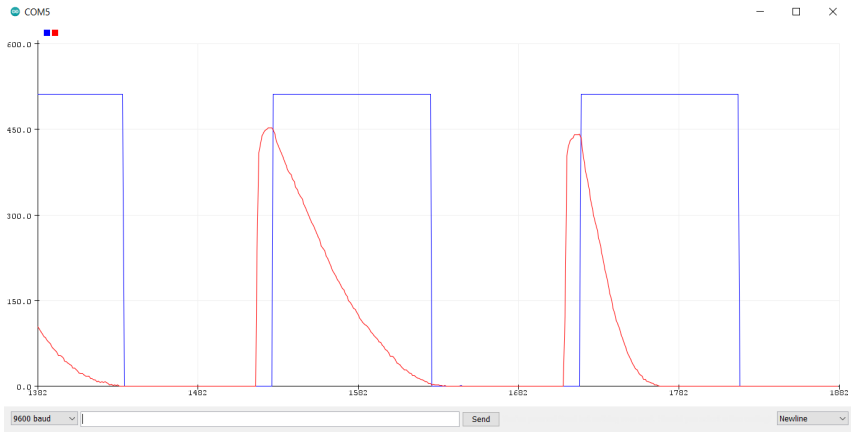


Figure 5.3. Output of testing

5.2 Computer vision

Computer vision can be used to locate objects or shapes in images. The points found on the image can be used to determine the position of the object, in relation to a known frame from the robot. In the case of the tube tester, which needs to be inserted into a specific shape, eg a circular hole, the use of computer vision is suitable and advantageous. By using computer vision the system can correct itself during the operation, which makes it more robust. The downside of using a computer vision is that the system becomes more complex.

5.2.1 Theory

There are many areas in computer vision which purpose is to generate information from images. In the scope of this project 3D estimation and object recognition is used. Object recognition is used to locate the holes in an image and the 3D estimation is used to reference the image pixel to a coordinate system in reference to the robotics system. The procedure for 3D estimation is depending on the camera type.

A 3D camera or a stereo camera is a camera with more than one lens. This allows the user the generate depth information from the image, as in the case of the human eyes. This is an expensive solution and is therefore not used in this project.

A 2D camera can also be used to make a 3D estimation if you know the distance from the camera frame to the 3D point. In this project, a 2D pinhole camera is used. The principle of the pinhole camera in 3D estimation is shown in figure fig. 5.4 on the following page.

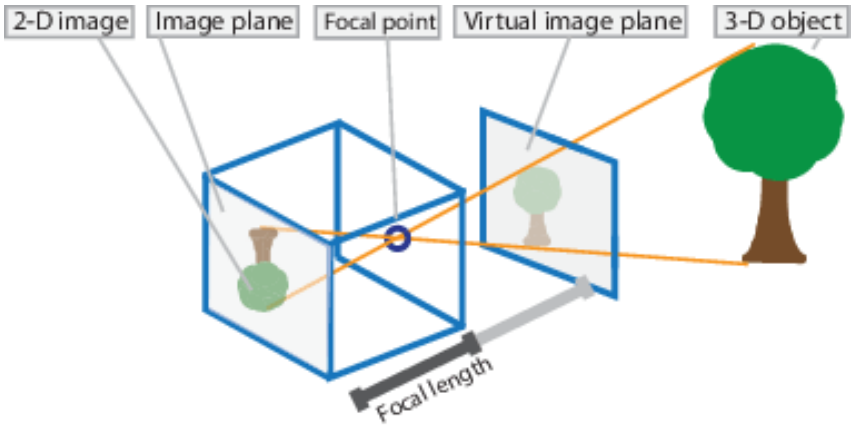


Figure 5.4. Pinhole Camera principal Matlab [2021]

Light passes through the focal point and a 2D image is projected on the image plane.

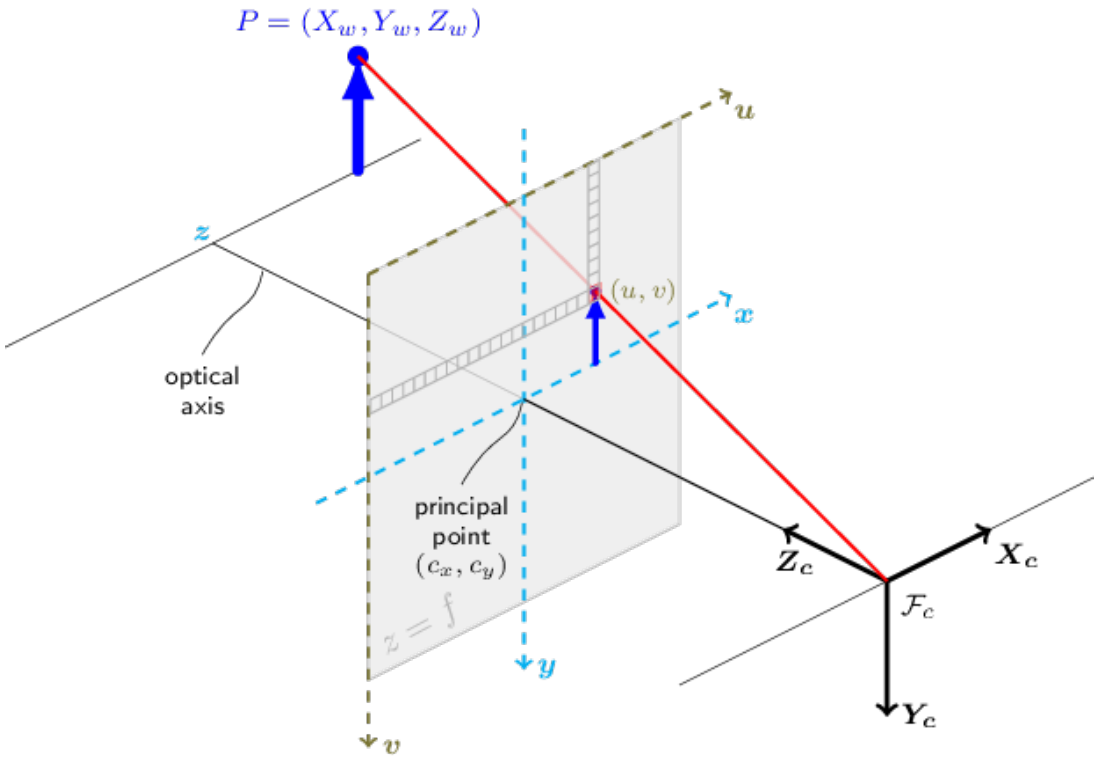


Figure 5.5. Pinhole Camera OpenCV [2021]

For this system, you can set up the equation for the camera center point (X_c, Y_c, Z_c), camera matrix, the pixels (u,v), and the scaling s. The camera matrix consists of the focal length f_y and f_x and the principal point c_y and c_x.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

This relation is the intrinsic parameter and is determined by doing the camera calibration. In order to reference the pixels to the world coordinate the Extrinsic parameters are needed.

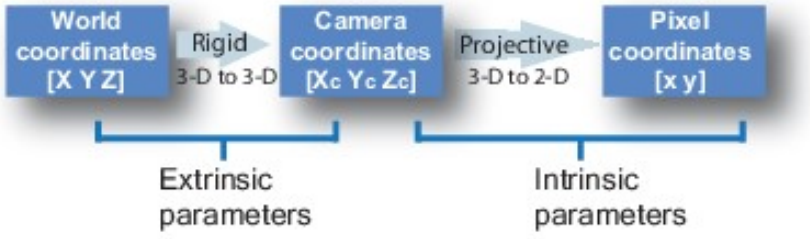


Figure 5.6. Extrinsic- and Intrinsic parameters Matlab [2021]

The extrinsic parameters are defined as the following, where P_w is the point in the world frame and R is the rotation matrix and t is the translation matrix.

$$P_c = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} P_w$$

By combining the extrinsic and the intrinsic parameters we obtain a relation between the pixel and the world point.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

5.2.2 Camera calibration

To calculate the position of the point in the world from the pixels, you need the camera matrix. To find the camera matrix you need to perform the camera calibration. This is done by knowing an object in the world frame. For that, a checkerboard is used. A checkerboard is a board with squares of accurate and equal size. By the distortion of each square in relation to the other squares and the length of sides, you can determine the camera parameters including the camera matrix. To obtain good results multiple images are used to do the camera calibration. In fig. 5.7 on the next page one of the pictures used for the camera calibration is shown. In the picture, the checkerboard is located.

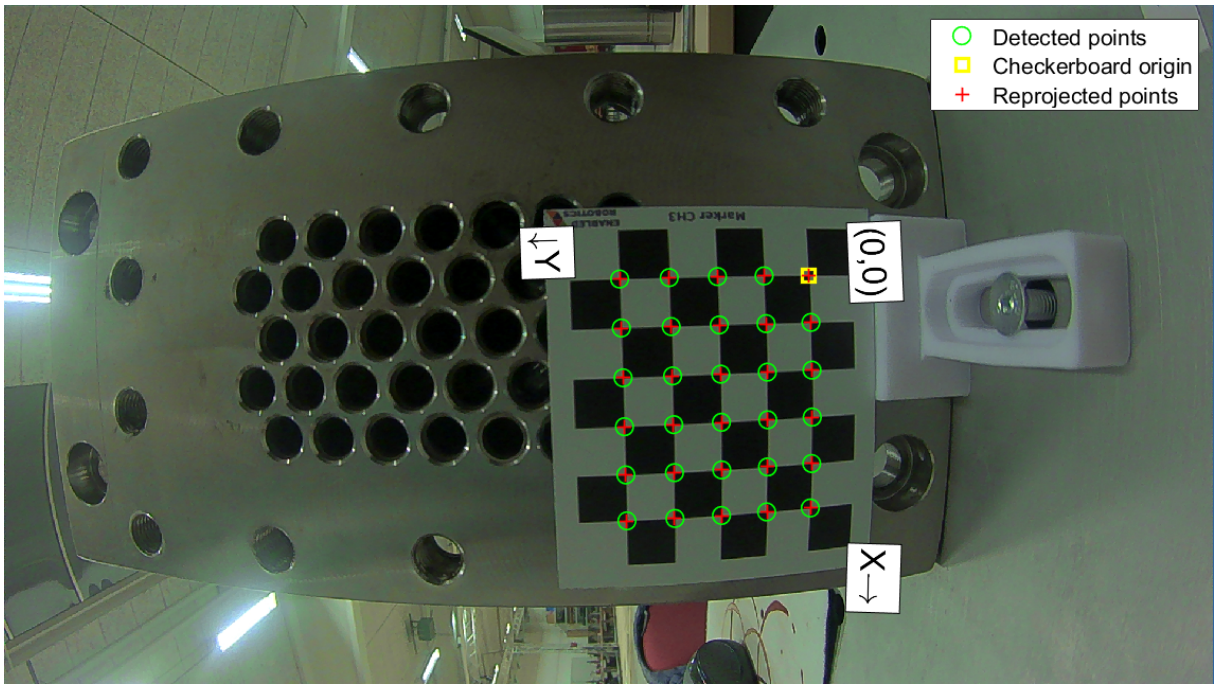


Figure 5.7. Camera calibration

The camera matrix for the calibration is found to be:

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2174 & 0 & 912 \\ 0 & 2169 & 548 \\ 0 & 0 & 1 \end{bmatrix}$$

5.2.3 Distortion

Most cameras have distortion on the image which is caused by the lens. The distortion results in an image where straight lines become curved around the edge of the image. In the camera calibration process, you also find the distortion of the images. By knowing the distortion of the image you can compensate for the effect and make real-world straight lines straight in the image. The effect of distortion can be seen on the distorted image fig. 5.8 on the following page and the undistorted image fig. 5.9 on the next page. In relation to the project, it is important to remove the distortion of the image, in order to be able to find the tube holes. However, the downside of removing the distortion is that the worldview area becomes smaller.

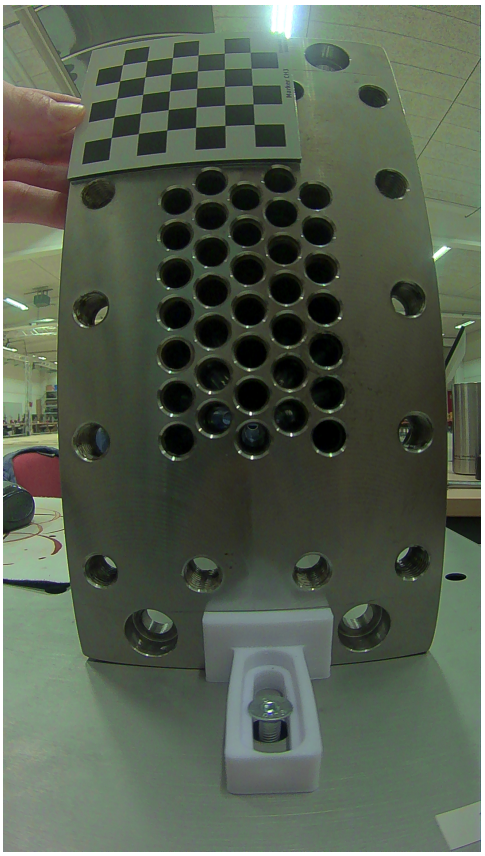


Figure 5.8. Original image

Figure 5.9. Undistorted Image

5.2.4 Circle detection

For detecting shapes or colors you need advanced algorithms. Conveniently these algorithms are implemented in Matlab computer vision toolbox and open source library OpenCV. The Matlab function "imfindcircles", takes the argument for the image and the radius of the circle and returns the center coordinates in reference to the image pixels. In order to make the algorithm work better, you need to convert the image to grayscale, which makes the shape of the circle more visible to the algorithm.

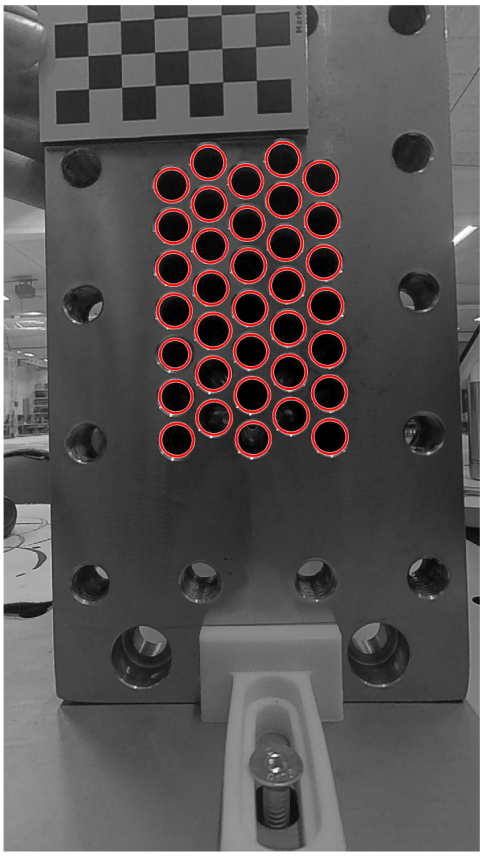


Figure 5.10. Circle detection (Red circle indicates found hole).

5.2.5 Hand eye calibration

The hand eye calibration is a method to determine the location of the camera frame. The camera frame is used to determine the world point from the image point. The camera center point can't be measured since it isn't a physical point.

To do the camera calibration you need to define a world frame. To define the world frame you use the checkerboard, which needs to be in a fixed position in relation to the robotics base frame. To find the transformation matrix from the base to the world frame you can set up the following equation:

$$T_{base}^{world} = T_{base}^{gripper} \cdot T_{gripper}^{camera} \cdot T_{camera}^{world}$$

The transformation matrix from the base to the gripper can be determined by forward kinematics. The transformation matrix from the camera to the world can be determined

with computer vision. By using two positions you will be able to set up one equation with one unknown.

$$T(1)_{base}^{gripper} \cdot T_{gripper}^{camera} \cdot T(1)_{camera}^{world} = T(2)_{base}^{gripper} \cdot T_{gripper}^{camera} \cdot T(2)_{camera}^{world}$$

This equation can be simplified, where A and B figuration of the robotics system. This problem is illustrated in fig. 5.11.

$$AX = XB$$

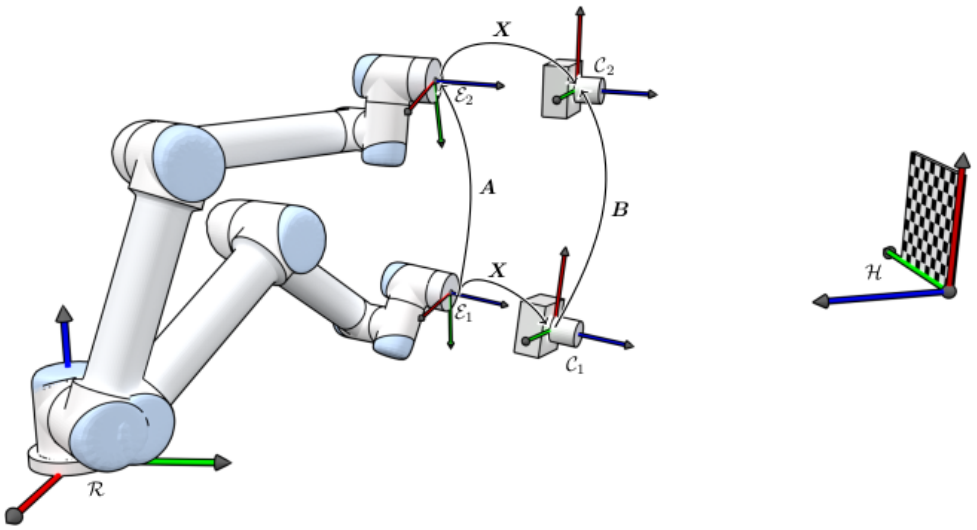


Figure 5.11. Hand eye calibration Torstein Myhre [2021]

5.2.6 Image points to world point

In order to calculate the world point, you need the camera intrinsic parameters, rotation vector, translation vector, and the image point.

Since the camera is a 2D camera, it can not measure the depth in the images. Therefore the depth 'd' is measured by the robotics system. The depth information is used to construct the translation vector to the world frame. In the figure the world frame is projected on the cooler surface in the optical z-axis with the translation vector:

$$T = \begin{bmatrix} 0 \\ 0 \\ d \end{bmatrix}$$

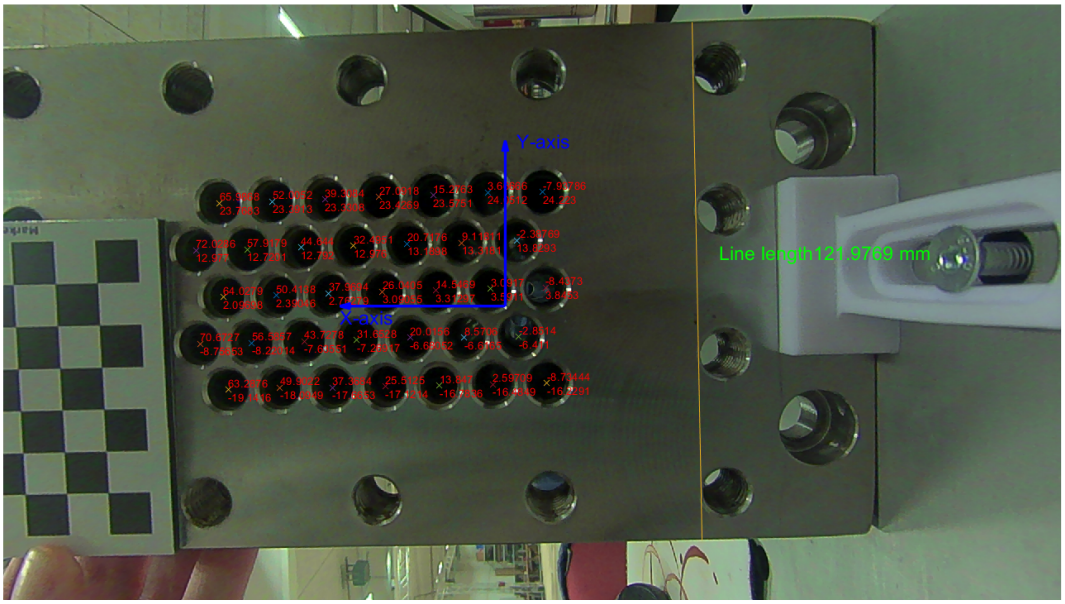
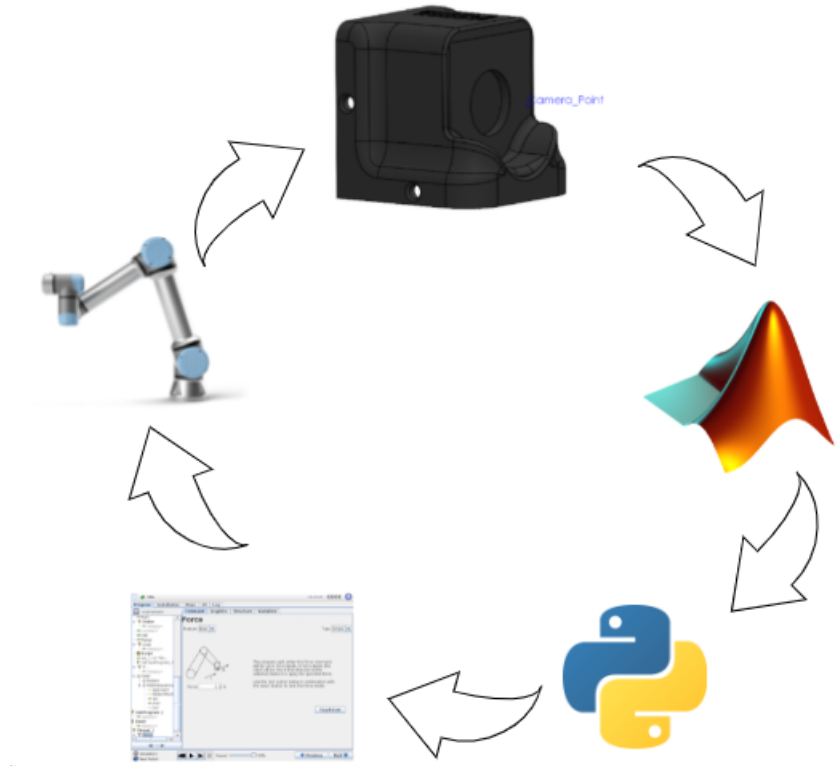


Figure 5.12. World coordinate of tube holes

The coordinates are calculated in real-time and used for adjusting the robotic system. By knowing the center of the hole from the vision control the system can adjust for productions error and cumulated errors in the robotic system.

5.2.7 XML-RPC COMMUNICATION

After obtaining the correct coordinates of the first hole, MatLAB algorithm 'Image points to world point' sends the corrected coordinates to Python algorithm 'XMLServer'. XML-RPC is a Remote Procedure Call method that uses XML to transfer data between programs over sockets. With it, the UR controller can call methods or functions on a remote program/server and get back structured data.
What XMLServer in Python does, is creating an server that is waiting to receive robot current position in front of the cooler. After camera sends the picture to the MatLAB and MatLab calculates the correct position, Python orders PolyScope to move the robot to the corrected position. This is useful function in a way that user can still program a main program on PolyScope, but an advanced functions for Camera vision and calculations are taking place at a server outside the robot system. Whole process is visualized in fig. 5.13 on the next page.



s

Figure 5.13. Programming Process diagram

5.3 Conclusion

In beginning, it was easy to understand the concept how the used programs should operate, but once the programming actually starts, many errors occur, which requires a lot of 'try and error'.

Various programming methods have been used during the project. It was interesting to see how to actually implement programming in order to move the robot, and how many different systems need to work simultaneously and parallel to each other. Main robot program is written on PolyScope, the Computer vision part was programmed in MatLAB, while the connection between robot and program was made through Python. In addition, the pressure system was controlled with Arduino, and all robot calculations were made in MatLAB.

6 | Robot calculations

This chapter will describe the calculations for UR5e kinematics, which studies the relationship between the dimensions and connectivity of kinematic chains and the position, velocity and acceleration of each of the links in the robotic system, in order to plan and control movement and to compute actuator forces and torques. Also, will describe the calculations for dynamics, which studies the relationship between mass and inertia properties, motion, and the associated forces and torque.

6.1 Forward Kinematics

The forward kinematics equation, which describes the transformation from frame 0 to frame 6 is derived by multiplying the transformation matrices between each frame and can be written as in equation 6.1 and equation 6.2. The transformation matrix from one frame to the next is derived using the D-H parameters and equation 6.3. To replace the frame 6 with the Tool frame at the end of the tube tester, we simply ad together the parameters d_6 and d_{Tool} for the last frame.

$$\begin{aligned} {}^{0}_{6}\mathrm{T}(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6,) &= \begin{bmatrix} {}^{0}_{6}\mathrm{R} & {}^{0}\mathrm{P}_6 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} {}^{0}\hat{X}_6 & {}^{0}\hat{Y}_6 & {}^{0}\hat{Z}_6 & {}^{0}\mathrm{P}_6 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} {}^{0}\hat{X}_{6x} & {}^{0}\hat{Y}_{6x} & {}^{0}\hat{Z}_{6x} & {}^{0}\mathrm{P}_{6x} \\ {}^{0}\hat{X}_{6y} & {}^{0}\hat{Y}_{6y} & {}^{0}\hat{Z}_{6y} & {}^{0}\mathrm{P}_{6y} \\ {}^{0}\hat{X}_{6z} & {}^{0}\hat{Y}_{6z} & {}^{0}\hat{Z}_{6z} & {}^{0}\mathrm{P}_{6z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{6.1}$$

$${}^{0}_{6}\mathrm{T}(\theta_1,\theta_2,\theta_3,\theta_4,\theta_5,\theta_6,) = {}^{0}_{1}\mathrm{T}(\theta_1)\cdot{}^{1}_{2}\mathrm{T}(\theta_2)\cdot{}^{2}_{3}\mathrm{T}(\theta_3)\cdot{}^{3}_{4}\mathrm{T}(\theta_4)\cdot{}^{4}_{5}\mathrm{T}(\theta_5)\cdot{}^{5}_{6}\mathrm{T}(\theta_6) \tag{6.2}$$

$${}^{i-1}_{i}\mathrm{T} = \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_{i-1} \\ sin(\theta_i)\cdot cos(\alpha_{i-1}) & cos(\theta_i)\cdot cos(\alpha_{i-1}) & -sin(\alpha_{i-1}) & -sin(\alpha_{i-1})\cdot d_i \\ sin(\theta_i)\cdot sin(\alpha_{i-1}) & cos(\theta_i)\cdot sin(\alpha_{i-1}) & cos(\alpha_{i-1}) & cos(\alpha_{i-1})\cdot d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.3}$$

With the DH-parameters inserted, and using a change in notation for sine and cosine functions as in 6.4, the transformation frames for each link looks as following:

$$sin(\theta_i)/cos(\theta_i) = s_i/c_i \tag{6.4}$$

$$sin(\theta_i + \theta_j)/cos(\theta_i + \theta_j) = s_{ij}/c_{ij} \tag{6.5}$$

$${}^{0}_{1}\mathrm{T} = \begin{bmatrix} c_1 & -s_1 & 0 & 0 \\ s_1 & c_1 & 0 & 0 \\ 0 & 0 & 1 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{1}_{2}\mathrm{T} = \begin{bmatrix} c_2 & -s_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ s_2 & c_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{2}_{3}\mathrm{T} = \begin{bmatrix} c_3 & -s_3 & 0 & a_2 \\ s_3 & c_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{3}_{4}\mathrm{T} = \begin{bmatrix} c_4 & -s_4 & 0 & a_3 \\ s_4 & c_4 & 0 & 0 \\ 0 & 0 & 1 & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$${}^{4}_{5}\mathrm{T} = \begin{bmatrix} c_5 & -s_5 & 0 & 0 \\ 0 & 0 & -1 & -d_5 \\ s_5 & c_5 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad {}^{5}_{6}\mathrm{T} = \begin{bmatrix} c_6 & -s_6 & 0 & 0 \\ 0 & 0 & 1 & d_6 + d_{Tool} \\ -s_6 & -c_6 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The resulting forward kinematics of the UR5e robot, from base to the tube tester tip, represented by the transformation matrix from frame 0 to frame 6 can be seen in Appendix C.

The Forward kinematics can be expanded to include the World frame, the Hole frame, and the Camera frame. Through the visual recognition, a transformation matrix from the Hole frame to the World frame is calculated. With a known transformation matrix from the World frame to the 0 frame of the robot, a full transformation from the Hole frame to the Tool frame can be calculated by multiplying the transformation matrices as in equation 6.6. This is in its simplest form the operation done by visual recognition.

$${}^{Hole}_{Tool}\mathrm{T} = {}^{0}_{6/Tool}\mathrm{T} \cdot {}^{World}_{0}\mathrm{T} \cdot {}^{Hole}_{World}\mathrm{T} \tag{6.6}$$

6.2 Inverse kinematics

The forward kinematic problem uses the kinematics equations to determine the position and orientation ${}^{0}_{t}T$ with given joint angles $\theta_i = (i, \ldots, n)$. The inverse kinematics problem computes the joint angles $\theta_i = (i, \ldots, n)$ with given position and orientation of end-effector ${}^{0}_{t}T$.

Calculating the needed joint angles that will give a desired position and orientation (inverse kinematics) is a more complex problem than that of forward kinematics. In some cases, there may be closed form solutions, but for robots with more than a couple joints it could be very difficult, if not impossible, to derive a close form solution.

All manipulators with either revolute and/or prismatic joints having a total of six degrees of freedom in a single series chain are solvable with Numerical Solutions, which is iteration procedure and as such very slow.

Closed - form Solutions, based on analytic expressions and efficient in calculation, only exist for a special robot having several intersecting joint axes or having many $\alpha_i = 0$ or $\pm 90°$. If manipulator with revolute joints has three neighboring joint axes intersecting at a point, then it has a closed-form solution. This is the design of most of manipulators manufactured today. The UR5 robot, for which an inverse kinematics calculation will be made, has such a design.

In this part will be presented calculation for obtaining the joint angles θ_{1-6} based on given desired position and orientation of the end-effector ${}^0_6T$. Calculations and the pictures are based on Kelsey P. Hawkins [2013], Ryan Keating [2017] and Rasmus Skovgaard Andersen [2017].

6.2.1 Finding θ_1 (Base)

To find θ_1, we begin with finding the location of frame 5 in relation to the first frame, which is also at the same location as base frame. First, we need to find origin of the 5th frame, ${}^0P_5$. It can be found by translating frame 6 in negative direction of Z by distance of d_6 to frame 5.

Translation of ${}^0P_5$:

$$
\begin{aligned}
{}^0P_5 &= {}^0P_6 - d_6 \cdot \hat{Z} \\
{}^0P_5 &= {}^0_6T \cdot \begin{bmatrix} 0 \\ 0 \\ -d_6 \\ 0 \end{bmatrix}
\end{aligned} \tag{6.7}
$$

With the overhead view of the robot, we can find θ_1, considering the wrist, P_5, which is seen from frame 0 and frame 1. Rotation θ_1 should equal rotations from 0 to 5 and 1 to 5. From fig. 6.2 on the following page we can conclude:

$$
\theta_1 = \phi_1 + \left(\phi_2 + \frac{\pi}{2}\right) \tag{6.8}
$$

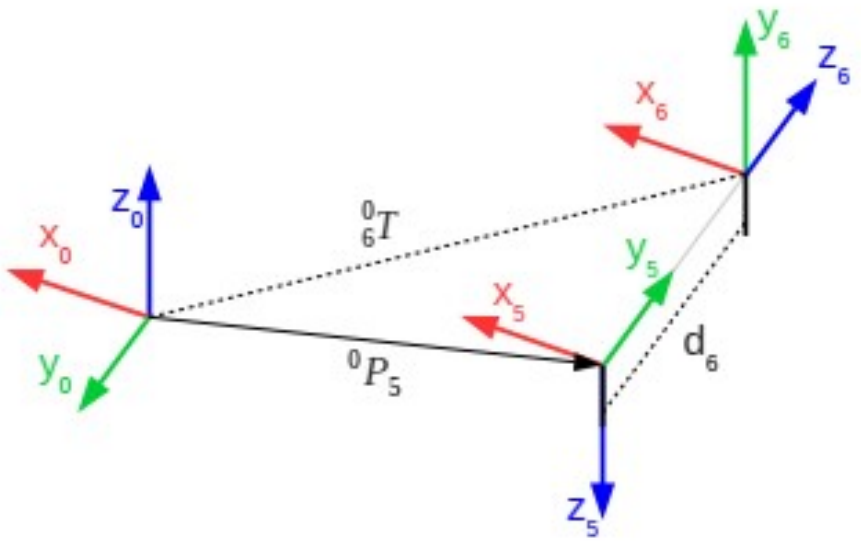


Figure 6.1. Finding the origin of frame 5

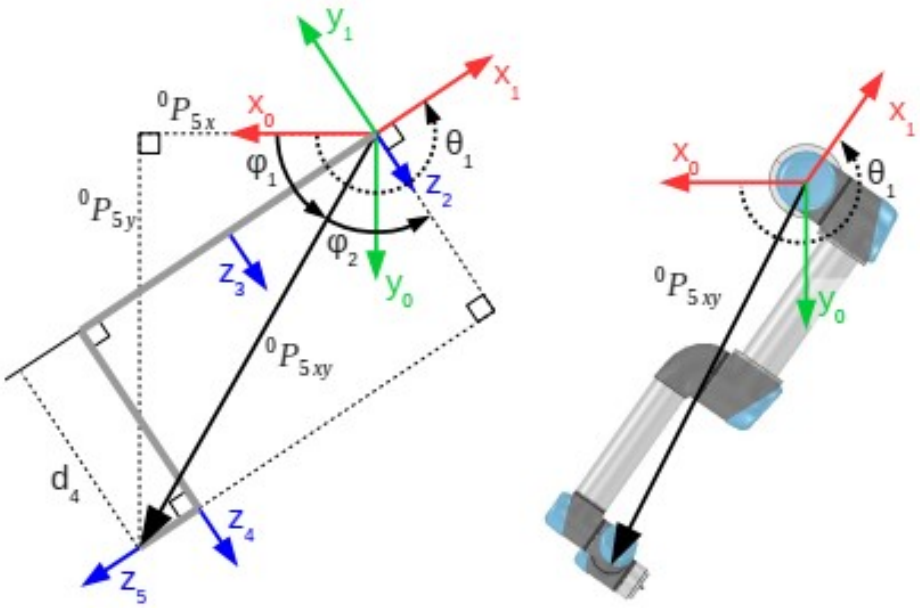


Figure 6.2. Robot seen from above until frame 5

Angle ϕ_1 can be found calculating the angle between ${}^0P_{5x}$ and ${}^0P_{5y}$

$$\phi_1 = atan2\left({}^0P_{5y}, {}^0P_{5x}\right) \tag{6.9}$$

Angle of ϕ_2 can be found as an angle in triangle between d_4 and hypotenuse of triangle with ${}^0P_{5x}$ and ${}^0P_{5y}$ as sides of a triangle:

$$\cos(\phi_2) = \frac{d_4}{|{}^0P_{5xy}|}$$

$$\phi_2 = \pm acos\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right) \tag{6.10}$$

With known angles ϕ we can calculate θ_1:

$$\theta_1 = atan2\left({}^0P_{5y}, {}^0P_{5x}\right) \pm acos\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right) + \frac{\pi}{2} \tag{6.11}$$

We can have two solutions corresponding to the shoulder being "left" or "right".

6.2.2 Finding θ_5 (Wrist 2)

In fig. 6.3 there is again view of robot from above, but now with frame 6 included.

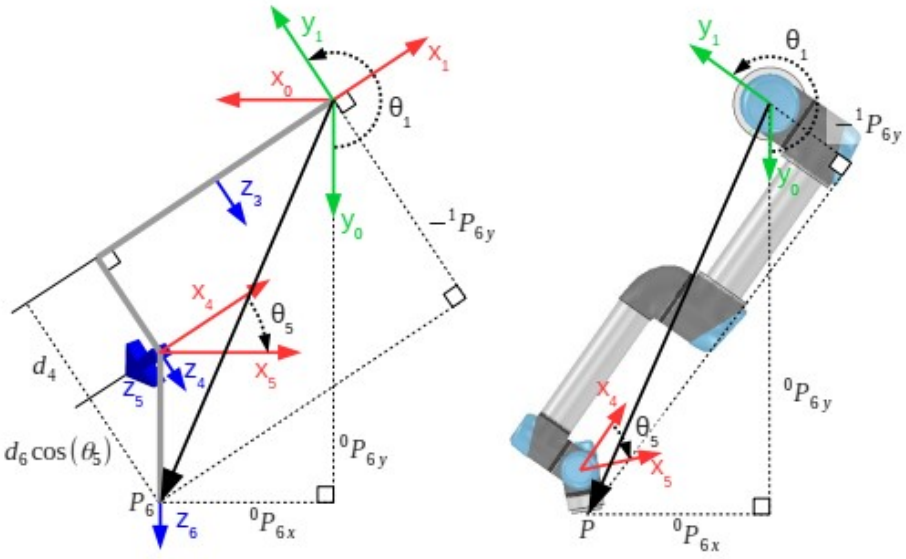


Figure 6.3. Robot seen from above until including 6

We can notice that y-component of ${}^1P_6$ only depends on θ_5. We can trace ${}^1P_{6y}$ backwards to find θ_5.

$$-{}^1P_{6y} = d_4 + d_6 \cos\theta_5 \tag{6.12}$$

The component ${}^1P_{6y}$ can also be expressed by finding position vector ${}^1P_6$ as rotation of ${}^0P_6$ around z_1:

$${}^0P_6 = {}^0_1R \cdot {}^1P_6$$

$$^1P_6 = {}^0_1R^T \cdot {}^0P_6$$

$$\begin{bmatrix} ^1P_{6x} \\ ^1P_{6y} \\ ^1P_{6z} \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \begin{bmatrix} ^0P_{6x} \\ ^0P_{6y} \\ ^0P_{6z} \end{bmatrix}$$

$$\begin{bmatrix} ^1P_{6x} \\ ^1P_{6y} \\ ^1P_{6z} \end{bmatrix} = \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 \\ 0 & 0 & 1 \end{bmatrix}^T \begin{bmatrix} ^0P_{6x} \\ ^0P_{6y} \\ ^0P_{6z} \end{bmatrix}$$

$$^1P_{6y} = {}^0P_{6x} \cdot (-\sin\theta_1) + {}^0P_{6y} \cdot \cos\theta_1 \tag{6.13}$$

We combine obtained equations eq. (6.12) on the preceding page and eq. (6.13) to isolate θ_5:

$$-d_4 - d_6 \cos\theta_5 = {}^0P_{6x}(-\sin\theta_1) + {}^0P_{6y}\cos\theta_1$$

$$\theta_5 = \pm acos\left(\frac{{}^0P_{6x}\sin\theta_1 - 0P_{6y}\cos\theta_1 - d_4}{d_6}\right) \tag{6.14}$$

We can have two solutions corresponding to the wrist being "up" or "down". This solution is possible as long as $|^1P_{6y} - d_4| \leq |d_6|$.

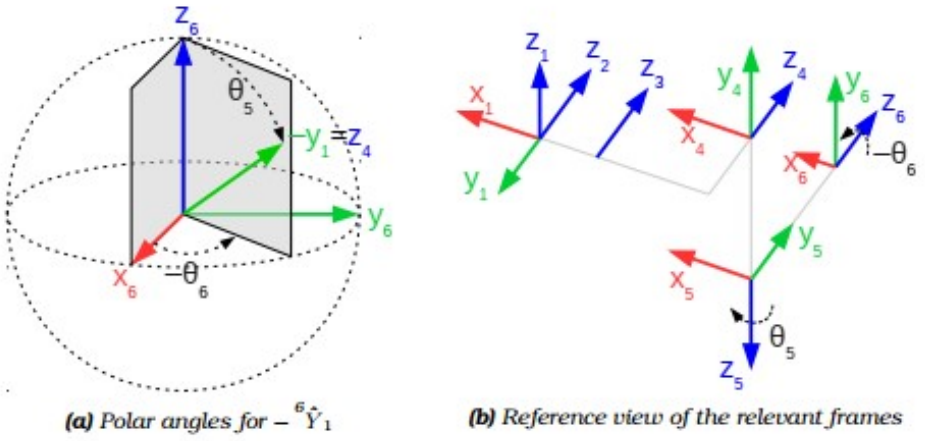
6.2.3 Finding θ_6 (Wrist 3)

Finding θ_6 can be done by using spherical coordinates, where azimuth is θ_6 and polar angle is θ_5, as it is shown in fig. 6.4 on the following pagea. From the fig. 6.4 on the next pageb we can see that ${}^6\hat{Y}_1$ will always be parallel to ${}^6\hat{Z}_{2,3,4}$, so it only depends on θ_5 and θ_6. In fig. 6.4 on the following pagea axis ${}^6\hat{Y}_1$ is denoted as y_1.

We convert ${}^6\hat{Y}_1$ from spherical to Cartesian coordinate system:

$$-{}^6\hat{Y}_1 = \begin{bmatrix} \sin\theta_5\cos(-\theta_6) \\ \sin\theta_5\sin(-\theta_6) \\ \cos\theta_5 \end{bmatrix}$$

$${}^6\hat{Y}_1 = \begin{bmatrix} -\sin\theta_5\cos\theta_6 \\ \sin\theta_5\sin\theta_6 \\ -\cos\theta_5 \end{bmatrix} \tag{6.15}$$



(a) Polar angles for $-{}^{6}\hat{Y}_1$ **(b)** Reference view of the relevant frames

Figure 6.4. ${}^{6}\hat{Y}_1$ expressed in spherical coordinates

From eq. (6.13) on the previous page we could isolate θ_6 and make expression of θ_6 in relation to ${}^{6}_{1}T$. We can identify ${}^{6}\hat{Y}_1$ as a rotation of θ_1 in x-y plane of frame 0, so we have θ_6 all the way from ${}^{6}_{0}T$.

$$ {}^{6}\hat{Y}_1 = {}^{6}\hat{X}_0 \cdot (-\sin\theta_1) + {}^{6}\hat{Y}_0 \cdot \cos\theta_1 $$

$$ {}^{6}\hat{Y}_1 = \begin{bmatrix} -{}^{6}\hat{X}_{0x} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0x} \cdot \cos\theta_1 \\ -{}^{6}\hat{X}_{0y} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0y} \cdot \cos\theta_1 \\ -{}^{6}\hat{X}_{0z} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0z} \cdot \cos\theta_1 \end{bmatrix} \tag{6.16} $$

Combining eq. (6.15) on the preceding page and eq. (6.16) we can get theta 6 isolated:

$$ \begin{aligned} -\sin\theta_5 \cos\theta_6 &= -{}^{6}\hat{X}_{0x} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0x} \cdot \cos\theta_1 \\ \sin\theta_5 \sin\theta_6 &= -{}^{6}\hat{X}_{0y} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0y} \cdot \cos\theta_1 \end{aligned} \rightarrow \tag{6.17} $$

$$ \theta_6 = atan2\left(\frac{-{}^{6}\hat{X}_{0y} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0y} \cdot \cos\theta_1}{\sin\theta_5}, \frac{-{}^{6}\hat{X}_{0x} \cdot \sin\theta_1 + {}^{6}\hat{Y}_{0x} \cdot \cos\theta_1}{\sin\theta_5} \right) \tag{6.18} $$

This solution can't be obtained if sine of θ_5 is 0. That means that axes 2,3,4 and 6 are aligned. As well if both of numerator are equal to 0, we can't obtain the solution, unless sine of θ_5 is also 0.

6.2.4 Finding θ_3 (Elbow)

If we look at frame 4 in relation to frame 1, because at this point we know ${}^{0}_{1}T$, ${}^{4}_{5}T$ and ${}^{5}_{6}T$, we can see that length of the translation ${}^{1}P_{4xz}$ is determined only by θ_3, or similarly ϕ_3

seen from fig. 6.5. We can find angle θ_3 by using the law of cosine.

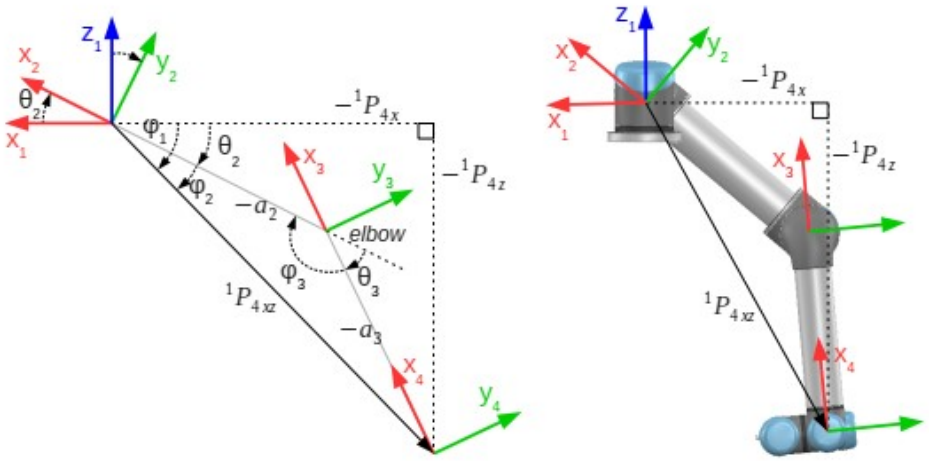


Figure 6.5. Joint 2,3 and 4 which together make a 3R planar manipulator

$$\cos\phi_3 = \frac{(-a_2)^2 + (-a_3)^2 - |{}^1P_{4xz}|^2}{2 - (a_2)(-a_3)} = \frac{{a_2}^2 + {a_3}^2 - |{}^1P_{4xz}|^2}{2a_2a_3} \tag{6.19}$$

Relation between θ_3 and ϕ_3:

$$\cos\theta_3 = \cos(\pi - \phi_3) = -\cos(\phi_3) \tag{6.20}$$

Combining eq. (6.19) and eq. (6.20):

$$\cos\theta_3 = -\frac{{a_2}^2 + {a_3}^2 - |{}^1P_{4xz}|^2}{2a_2a_3}$$

$$\theta_3 = \pm acos\left(\frac{|{}^1P_{4xz}|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3}\right) \tag{6.21}$$

Solution of θ_3 only exists if $|{}^1P_{4xz}| \in |a_2 - a_3|; |a_2 + a_3|$. In cases where solution exists, there will be two different solutions, which correspond to "elbow up" and "elbow down".

6.2.5 Finding θ_2 (Shoulder)

The angle θ_2 can be found as $\phi_1 - \phi_2$. It can be seen in fig. 6.5

$$\phi_1 = atan2\left(-{}^1P_{4z}, -{}^1P_{4x}\right) \tag{6.22}$$

$$\frac{\sin\phi_2}{-a_3} = \frac{\sin\phi_3}{|^1P_{4xz}|} \rightarrow$$

$$\phi_2 = asin\left(\frac{-a_3\sin\phi_3}{|^1P_{4xz}|}\right) \tag{6.23}$$

We replace ϕ_3 with θ_3 by noticing $\sin\phi_3 = \sin\left(180° - \theta_2\right)$:

$$\theta_2 = \phi_1 - \phi_2 = atan2\left(-^1P_{4z}, -^1P_{4x}\right) - asin\left(\frac{-a_3\sin\phi_3}{|^1P_{4xz}|}\right) \tag{6.24}$$

6.2.6 Finding θ_4 (Wrist 1)

The angle θ_4 if defined as an angle from X_34 to X_4 measured about Z_4. It can be derived from transformation matrix ${}^3_4T$, and using first column ${}^3\hat{X}_4$:

$$\theta_4 = atan2\left({}^3\hat{X}_{4y}, {}^3\hat{X}_{4x}\right) \tag{6.25}$$

6.2.7 Conclusion

There is a total of 8 solutions exist in general for the general inverse kinematic problem of the UR5e: $2\theta_1 \times 2\theta_5 \times 1\theta_6 \times 2\theta_3 \times 1\theta_2 \times 1\theta_4$

The verification of the code and resulting inverse kinematics code of the UR5e robot calculation for obtaining the joint angles θ_{1-6} based on given desired position and orientation of the end-effector ${}^0_6T$ can be seen in Appendix E

6.3 Dynamics

When the dynamics is calculated the wanted end result is the torque, τ, in each joint. To achieve this a lot of intermediate calculations is needed. The following calculations will follow the Newton-Euler method as it's the easiest way to set up the calculations in a MATLAB-script, which is desired as the analytical equations will be very long.

Link parameters like, mass, mass center point, mass moment of inertia and DH-parameters has values as in section 4.2. The different rotation matrices and position vectors are the same as used to calculate the forward kinematics and follow the notation in equation

eq. (6.1) on page 25.

The calculation in MATLAB is done in two for loops. First angular velocity, angular acceleration, linear acceleration with respect to frame origin, linear acceleration with respect to link center of mass, force in the link, and moment in the link, is calculated from the frame 0 to frame 6. Here some definitions for values in frame 0 is defined prior to the for loop.
As the frame 0 always is stationary the angular velocity and acceleration is set to zero. To disregard the gravity in the subsequent links, it's added in the z-component of the linear velocity in frame 0.

In MATLAB all matrices are collected in a set, and all scalars are collected in a matrix, so it will be possible to index through them in the loop. The first loop look as seen below, with the link parameters, rotation matrices, and position vectors defined before:

```
% Empty cells
omega      = cell(1,7);
omega_dot  = cell(1,7);
v_dot      = cell(1,7);
v_c_dot    = cell(1,6);
F          = cell(1,6);
N          = cell(1,6);
% initial values
g            = 9.807;           %m/s^2
v_dot{1}     = g*[0;0;1];       % Acceleration first joint
omega{1}     = [0;0;0];         % Frame 0 fixed
omega_dot{1} = [0;0;0];         % Frame 0 fixed
for i = 1:6
    omega{i+1}     = R{i}'*omega{i}+theta_dot(i)*[0;0;1];
    omega_dot{i+1} =
        R{i}'*omega_dot{i}+cross((R{i}'*omega{i}),(theta_dot(i)*[0;0;1]))+theta_dotdot(i)*[0;0;1];
    v_dot{i+1}     =
        R{i}'*(cross(omega_dot{i},P{i})+cross(omega{i},cross(omega{i},P{i}))+v_dot{i});
    v_c_dot{i}     =
        cross(omega_dot{i+1},P_c{i})+cross(omega{i+1},cross(omega{i+1},P_c{i}))+v_dot{i+1};
    F{i}           = M(i)*v_c_dot{i};
    N{i}           = I_c{i}*omega_dot{i+1}+cross(omega{i+1},(I_c{i}*omega{i+1}));
end
```

The second loop takes the results generated in the first loop and calculate forces and moments in each joint. The calculation is this time done from frame 6 to frame 0. The calculation needs the forces and moment at frame 6 defined with respect to the 3 axes. In the MATLAB code they are set to 0 in all directions, but can easily be changed along with each joints angle, angular velocity and angular acceleration for a specific scenario. The second loop looks as below, and afterwards the torques are collected in a vector with torque in each joint from joint 0 to joint 6.

```
%Forces and Moments empty cells
f = cell(1,7);
n = cell(1,7);
tau = cell(6,1);
%inital values
f{7} = [0;0;0];
n{7} = [0;0;0];
for i = [6 5 4 3 2 1]
    f{i} = R{i}*f{i+1}+F{i};
    n{i} = N{i}+R{i}*n{i+1}+cross(P_c{i},F{i})+cross(P{i},(R{i}*f{i+1}));
    tau{i} = n{i}(3);
end

TAU = [tau{1};tau{2};tau{3};tau{4};tau{5};tau{6}];
```

In this projects there is not a lot of rapid movement with a heavy payload and the torque limits of the UR5e is not feared to be surpassed. An interesting scenario related to our project would be the static case, pictured in figure 6.6, where the tube tester is inserted into the hole with a force big enough to have a tight seal and overcome the force from the pressure build up inside the tube. For our test 15 N was sufficient, but that could increase if the same setup would be used on a cooler with different tube diameter. The torque in each joint could be calculated in the two for-loops with input:

$$\Theta = \begin{bmatrix} 62.0° & -37.3° & 106.3° & -68.4° & -28.2° & -90.3° \end{bmatrix}^T \qquad f_6 = \begin{bmatrix} 0 & 0 & 15N \end{bmatrix}^T$$
$$\dot{\Theta} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \qquad n_6 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$$
$$\ddot{\Theta} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

This results in a torque in $[N \cdot m]$ in each joint of:

$$\tau = \begin{bmatrix} -18.39 & -18.39 & -1.41 & -1.16 & -0.26 & 0 \end{bmatrix}^T$$

Figure 6.6. Position of the robot, where 15N is statically applied.

If the MATLAB-script is solved symbolically the force and moment in each link, denoted by the F and N vectors, can be used together with the symbolic torque values to derive the jacobian matrix. The F and N vectors would be collected as a general force vector. The relation between the generalized force vector and the joint torque vector would be given by the transposed jacobian. With all values expressed as symbolic functions of Θ, $\dot{\Theta}$, and $\ddot{\Theta}$, the expressions become very large and the process of actually deriving the jacobian

matrix very tidious.

$$\mathcal{F} = \begin{bmatrix} F \\ N \end{bmatrix}$$

$$\tau = J(\Theta)^T \cdot \mathcal{F}$$

With a defined jacobian matrix its possible to find the singularities of the UR5e, by seeing which angles fulfills 6.26

$$DET[J(\Theta)] = 0 \tag{6.26}$$

6.4 Conclusion

Kinematics and dynamics are a fundamental and classical topics in robotics. Kinematics can yield very accurate calculations in many problems, such as positioning a gripper at a place in space, designing a mechanism that can move a tool from point A to point B, or predicting whether a robot's motion would collide with obstacles.

The dynamical performance of robot manipulators is greatly affected by the different payloads handled by the end-effector. Hence, it is very important, especially for industrial applications, to study the different interconnected relationships between the manipulator's joints, speeds, loads and actuation forces.

7 | Test and simulation

The following chapter describes the procedure followed while testing.

Punctual timing and sequencing of tasks were crucial in creation of the procedure. For testing procedure to work, three different systems should work in parallel to each other, receiving and sending signals for execution of tasks at exact point in time. Systems are:

- Physical system – UR5e robot
- MatLAB - Camera vision
- Python code – Connection with robot
- Arduino code – Pressure system control

Simplified procedure can be found in the following algorithm, it is a substitute to flow chart from programming section

```
Attach adaptor (tube tester + camera + pressure)
Position the cooler
Python requires input (Is tube adaptor attached Y/N)
Python requires input (Is cooler placed on the table correctly inside the limits Y/N)
Start Robot
Move robot to the 1st point - approximately in front of the cooler (predefined position)
Start calibration process of aligning the gripper properly relative to cooler
    Move robot forward cooler, until impact (set force sensor to low value)
    Retract Gripper to predefined value in Z-direction
    Move to left of the cooler and behind the front face plane
    Move robot from left to right until impact
    Retract Gripper to predefined position in X-direction
    Move robot to right of the cooler and behind the front face plane
    Move robot from right to left until impact
    Retract Gripper to predefined position in Y-direction
    Move robot to front face
Turn ON the camera
Receive feedback from the camera about position of one of the corner holes, using checkerboard
Align the tube tester with the corner hole
Insert the tube tester into the hole
Give input to Arduino: PRESSURE ON
Hold pressure for 5 seconds
Give input to Arduino: PRESSURE OFF
Hold 5 seconds
Check if the pressure dropped
    If yes
        Mark hole as LEAK
    Else
        Mark hole as SAFE
Repeat for all holes
Move robot to HOME Position

OUTPUT: Table with the list of all holes, marked LEAK (must be checked and repaired) or SAFE
```

7.1 Simulation

Before starting the test on UR5e robot, basic conditions were translated to RoboDK software. RoboDK is a powerful simulator for industrial robots and robot programming. By re-creating the imagined robot path, RoboDK can visualize the following path and at the same time check for collisions. By doing this, it is possible to avoid expensive mistakes of robot hitting and damaging the table, cooler or itself that could occur with mistakes in robot programming in PolyScope.

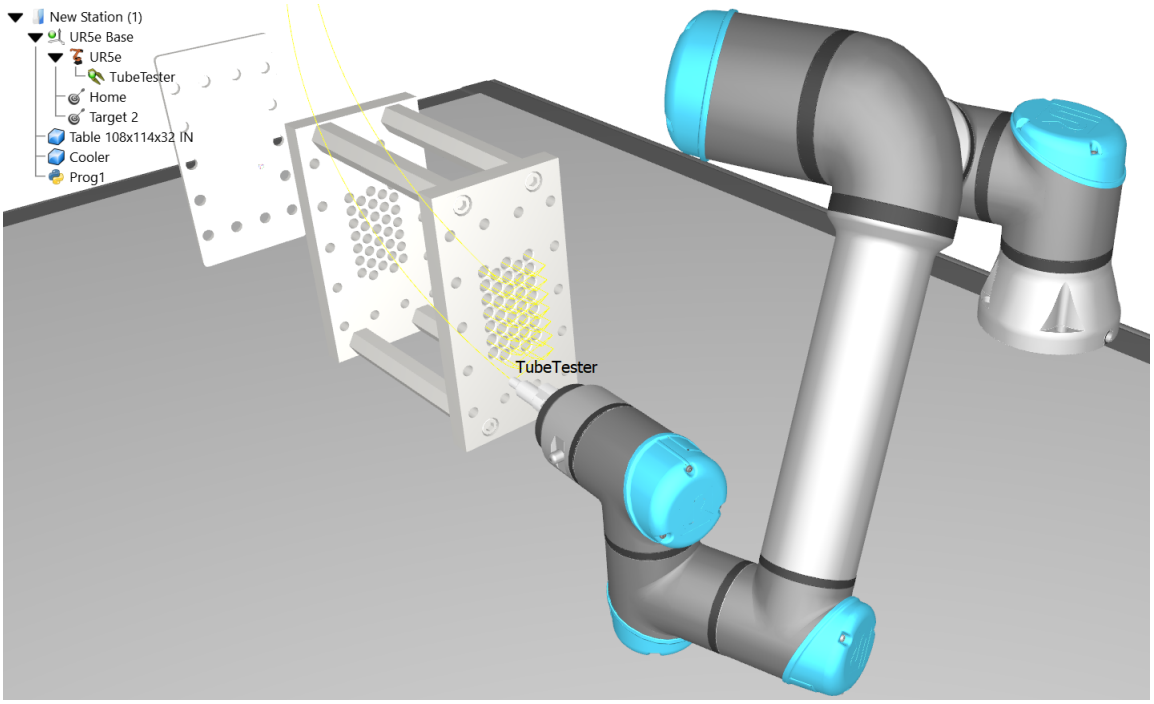


Figure 7.1. RoboDK Overview

From the fig. 7.1 it is clear that the yellow lines represent the tool path. From the organization tree it is possible to notice similarities between PolyScope program and RoboDK Simulation (Home Position, First Position, Tube Tester, Cooler etc.) Delimitation of the software is that only free trial edition is available to students, which doesn't allow saving program after initial 30 days, which significantly reduces possibilities of programming the robot on high level.

7.2 Results

Final testing can be seen at the video, which can be found by scaning the QR code in fig. 7.2 on the following page. From the video it is possible to see how all systems work

together, Starting from the robot, and ending with the pressure system, with pressure intervals shown at laptop.

Figure 7.2. QR code for videolink

7.3 Conclusion

The main objective of real-life testing was to prove that idea of automatization process was viable. Leaving some room for improvement such as: exploiting maximum speed of the robot while testing or combining all subprograms into one central Python code, group stands firm that the concept of automatization was proven even in such a short period of one semester.

8 | Discussion and future work

It can be argued that the programming of the robot should have been developed in one program. On the other hand, it has provided an overview of the pros and cons, that comes with the different program platforms. For example, it has been easy to get started on programming in Polyscope, but we have encountered difficulties when the programs became more complicated. At the other end of the spectrum, we have worked with Python, which has shown complicated to get started with but gives the opportunity to combine the entire program in one place. Seen from a learning perspective, we have benefited from becoming familiar with the different program platforms. On the other hand, it would also have provided valuable learning to achieve a more finished program, which is possible with a combined program.

One development potential for the system could be to optimize the working time of the robot. This could be done by working with more than a single tube tester or examining whether the speed of the robot could be increased.

The implementation of the Computer vision system is not part of the current program. The practical implementation of handle eye calibration is not implemented. It will be a work point for further work.

9 | Conclusion

The testing procedure of cooler tubes from Vestas Aircoile has been automated with the use of the Universal robot UR5e. The result of the automation is the gripper system and a robotic program. The gripper design consists of a mounting plate for the tube tester, a camera fitting, and a pneumatic system.

In the process of developing the robotic program, a simulation of the automated system has been constructed. The successful simulation has later on been verified in a physical test. In the physical test the "Force mode" has been used in order to determine the position of the cooler. The pneumatic system is autonomous and controlled by the robotic system through Universal Robots' Polyscope.

A computer vision system has been developed to correct the positioning of the gripper in relation to the stationary cooler. The method for correcting the position is basis shape detecting and 3D reconstruction.

The computer vision program is developed in Matlab, the robot program in Polyscope, the adjustment of the robotic position in Python, and the data acquirement in Arduino.

The basis of robotics theories used in the robotics system is derived and described. It is including the frame attachment, DH- parameters, forward kinematics, inverse kinematics, and robot dynamics.

Bibliography

Kelsey P. Hawkins, **2013**. Kelsey P. Hawkins. *Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 Arms*. https://smartech.gatech.edu/bitstream/handle/1853/50782/ur_kin_tech_report_1.pdf, 2013. Downloadet: 7-12-2021.

Matlab, **2021**. Matlab. *What Is Camera Calibration?* https://se.mathworks.com/help/vision/ug/camera-calibration.html, 2021. Downloadet: 12-12-2021.

OpenCV, **2021**. OpenCV. *Camera Calibration and 3D Reconstruction*. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html, 2021. Downloadet: 12-12-2021.

Rasmus Skovgaard Andersen, **2017**. Rasmus Skovgaard Andersen. *Kinematics of a UR5*. http://rasmusan.blog.aau.dk/files/ur5_kinematics.pdf, 2017. Downloadet: 7-12-2021.

Ryan Keating, **2017**. Ryan Keating. *M.E. 530. 646 UR5 Inverse Kinematics*. http://rasmusan.blog.aau.dk/files/ur5_kinematics.pdf, 2017. Downloadet: 7-12-2021.

Torstein Myhre, **2021**. Torstein Myhre. *Hand Eye Calibration*. https://www.torsteinmyhre.name/snippets/robcam_calibration.html, 2021. Downloadet: 18-12-2021.

A | Polyscope program

```
Program
  Robot Program
    MoveJ
      home
    MoveJ
      infront
      IF=infront
      MoveL
        Direction: Tool Z+
          Until (tool_contact_detection)
            AR=get_actual_tcp_pose()
    MoveL
      ymove=p[AR[0],(AR[1]+0.13),AR[2],AR[3],AR[4],AR[5]]
      ymove
      xmove=p[(AR[0]+0.03),ymove[1],AR[2],AR[3],AR[4],AR[5]]
      xmove
      Direction: Base Y-
        Until (tool_contact_detection)
          ydis=get_actual_tcp_pose()
      zmove=p[ydis[0],ydis[1],(ydis[2]+0.240),AR[3],AR[4],AR[5]]
      zmove
      move_over
      Direction: Base Z-
        Until (tool_contact_detection)
          zdis=get_actual_tcp_pose()
      Waypoint_1
    d=39.2/1000
    e=161/1000
    startpattern=p[AR[0],(ydis[1]-0.00758-0.01-d),(zdis[2]-0.00758-0.01-e),AR[3],A
    i=0
    j=0
    a=11.3/1000
    c=6.5/1000
    Y=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    Z=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]
    off=0
```

```
Loop i=<6
  j=0
  Loop 5 times
    If j=0
      Y[j+off]=a*4
    If j=1 or j=2 or j=3  or j=4 or j=5
      Y[j+off]=-a
    If j=0
      Z[j+off]=2*c
    If j=2 or j=4
      Z[j+off]=-c
    If j=1  or j=3  or j=5
      Z[j+off]=c
    j=j+1
  i=i+1
  off=off+5
Y[0]=0
Z[0]=0
MoveL
  startpattern
k=1
Set DO[1]=On
Loop 35 times
  pos=get_actual_tcp_pose()
  'visu recog to correct pos[1] and pos[2]'
  Force
    MoveL
      moL_tcp_z_to15N
      'check if it moves into hole by comparing distances'
      Set DO[3]=On
      Wait: 1.0
      Set DO[3]=Off
      Set AO[1]=5.0
      Wait: 10.0
      Set AO[1]=0.0
      Set DO[2]=On
      Wait: 1.0
      Set DO[2]=Off
  MoveL
    pos
```

```
    nextpos=p[pos[0],(pos[1]+Y[k]),(pos[2]+Z[k]),pos[3],pos[4],pos[5]]
    MoveL
      nextpos
    k=k+1
  Set DO[1]=Off
  MoveL
    IF
  MoveJ
    Waypoint_2
```

B | Computer vision

Introduction: In this appendix the Matlab code used in the section 5.2 on page 16 is included. The code is handling the camera calibration, undistortion, circle detection and computing the world points from the image.

```
% Computer vision

close all
clear
clc

% Define images to process
imageFileNames = {'C:\Users\Emil B Hansen\Desktop\Billeder\Ikke brugte\Photo-1.jpeg',...
    'C:\Users\Emil B Hansen\Desktop\Billeder\Ikke brugte\Photo-2.jpeg',...
    'C:\Users\Emil B Hansen\Desktop\Billeder\Ikke brugte\Photo-3.jpeg',...
    'C:\Users\Emil B Hansen\Desktop\Billeder\Ikke brugte\Photo-4.jpeg',...
    'C:\Users\Emil B Hansen\Desktop\Billeder\Ikke brugte\Photo-5.jpeg',...
    };

% Detect checkerboards in images
[imagePoints, boardSize, imagesUsed] = detectCheckerboardPoints(imageFileNames,
    'HighDistortion', true);
imageFileNames = imageFileNames(imagesUsed);

% Read the first image to obtain image size
originalImage = imread(imageFileNames{3});
[mrows, ncols, ~] = size(originalImage);

% Generate world coordinates of the corners of the squares
squareSize = 10; % in units of 'millimeters'
worldPoints = generateCheckerboardPoints(boardSize, squareSize);

% Calibrate the camera
[cameraParams, imagesUsed, estimationErrors] = estimateCameraParameters(imagePoints,
    worldPoints, ...
    'EstimateSkew', false, 'EstimateTangentialDistortion', false, ...
    'NumRadialDistortionCoefficients', 2, 'WorldUnits', 'millimeters', ...
    'InitialIntrinsicMatrix', [], 'InitialRadialDistortion', [], ...
    'ImageSize', [mrows, ncols]);

% View reprojection errors
h1=figure; showReprojectionErrors(cameraParams);

% Visualize pattern locations
h2=figure; showExtrinsics(cameraParams, 'CameraCentric');

% Display parameter estimation errors
displayErrors(estimationErrors, cameraParams);

% Remove effects of lens distortion.
undistortedImage = undistortImage(originalImage, cameraParams);
imwrite(undistortedImage,'UndistortImage.jpeg')

```

```
47 % Define camera center point
48 CameraFrame = cameraParams.Intrinsics.PrincipalPoint;
49
50 % Define rotation vector and translation vector
51 RotationVector = [cos(pi) -sin(pi) 0;sin(pi) cos(pi) 0; 0 0 1];
52 TranslationVector = [0;0;250];
53
54 % Convert image to gray scale
55 gray_image = rgb2gray(undistortedImage);
56 imwrite(gray_image,'GrayImage.jpeg')
57
58 % Show image
59 imshow(undistortedImage)
60 hold on
61
62 % Find circels
63 [center, radii] = imfindcircles(gray_image,[26
       43],'ObjectPolarity','dark','Sensitivity',0.9275);
64
65 % Draw center points and calculate world coordinate
66 for i = 1:length(center)
67     % Undistort centerpoints
68     newcenter = undistortPoints(center(i,:),cameraParams);
69     % Plot centerpoints
70     plot(center(i,1),center(i,2),'x')
71     % Calculate world coordinates
72     worldPoints =
           pointsToWorld(cameraParams.Intrinsics,RotationVector,TranslationVector,[newcenter(1),newcenter(2)]);
73     % Convert number to string
74     str = string(worldPoints);
75     % Print centerpoints coordinates
76     text(center(i,1),center(i,2),str,'Color','red','FontSize',8)
77 end
78 % Draw coordinate system
79 quiver(CameraFrame(1),CameraFrame(2),-300,0,0,'b','LineWidth',2)
80 text(CameraFrame(1)-300,CameraFrame(2)+20,'X-axis','Color','blue','FontSize',14)
81 quiver(CameraFrame(1),CameraFrame(2),0,-300,0,'b','LineWidth',2)
82 text(CameraFrame(1)+20,CameraFrame(2)-300,'Y-axis','Color','blue','FontSize',14)
83
84 % Draw reference
85 plot([1268.8 1250.8], [971.75 32.75])
86 Ref1_point = undistortPoints([1268.8 971.75],cameraParams);
87 Ref2_point = undistortPoints([1250.8 32.75],cameraParams);
88 Ref1 =
       pointsToWorld(cameraParams.Intrinsics,RotationVector,TranslationVector,[Ref1_point(1),Ref1_point(2)])
89 Ref2 =
       pointsToWorld(cameraParams.Intrinsics,RotationVector,TranslationVector,[Ref2_point(1),Ref2_point(2)])
90 Linelength = abs(Ref1(2)) + abs(Ref2(2))
91 str = string(Linelength);
92 text(1300,450,'Line length'+str+ ' mm','Color','green','FontSize',14)
```

C | Forward kinematics result

Introduction: In this appendix resulting matrix from forward kinematics is shown. The matrix was too big to include in the main rapport. Here $d_6 = d_6 + d_{Tool}$.

$$
{}^{0}_{6}\mathrm{T} = \begin{bmatrix}
c_6(s_1s_5 + c_1c_5c_{234}) - c_1s_6s_{234} & -s_6(s_1s_5 + c_1c_5c_{234}) - c_1c_6s_{234} & c_5s_1 - c_1c_{234}s_5 & d_4s_1 + d_6(c_5s_1 - c_1c_{234}s_5) + a_2c_1c_2 + a_3c_1c_{23} + c_1d_5s_{234} \\
-c_6(c_1s_5 - c_5s_1c_{234}) - s_1s_6s_{234} & s_6(c_1s_5 - c_5s_1c_{234}) - c_6s_1s_{234} & -c_1c_5 - c_{234}s_1s_5 & a_2c_2s_1 - d_6(c_1c_5 + c_{234}s_1s_5) - c_1d_4 + a_3c_{23}s_1 + d_5s_1s_{234} \\
c_{234}s_6 + c_5c_6c_{234} & c_6c_{234} - c_5s_6s_{234} & -s_5s_{234} & d_1 + a_2s_2 + a_3s_{23} - d_5(c_4c_{23} - s_4s_{23}) - d_6s_5(c_4s_{23} + c_{23}s_4) \\
0 & 0 & 0 & 1
\end{bmatrix}
$$

D | XML-RPC COMMUNICATION

Introduction: In this appendix the Python code used is included. The code is creating the server and after MatLab Analysis is done, it gives command to the robot about alignment with the first hole.

```
1  import sys
2  #UR Library
3  import urlib
4
5  # Server libraries
6  is_py2 = sys.version[0] == '2'
7  if is_py2:
8     from SimpleXMLRPCServer import SimpleXMLRPCServer
9  else:
10    from xmlrpc.server import SimpleXMLRPCServer
11
12 #Input from MatLab, corrected X and Y coordinates from the Camera vision
13 x = [input from Matlab]
14 y = [input from Matlab]
15
16 def get_next_pose(p):                    # receive current robot pose
17     assert type(p) is dict
18     ptl = urlib.poseToList(p)
19     print("Received pose: " + str(ptl))
20     pose = {'x': ptl[0] - x, 'y': ptl[1] - y, 'z': ptl[2], 'rx': ptl[3], 'ry': ptl[4], 'rz':
           ptl[5]}
21     print(pose)                          # Corrected pose, all parameters teh same except X and Y
22     return pose;
23
24
25 #Choose the correct PORT used on the robot, in this case 50000
26 server = SimpleXMLRPCServer(("", 50000), allow_none=True)
27 server.RequestHandlerClass.protocol_version = "HTTP/1.1"
28 print("Listening on port 50000...")
29
30 server.register_function(get_next_pose, "get_next_pose")
31
32 server.serve_forever()
```

E | Inverse kinematics result

Introduction: In this appendix complete code for forward kinematics of UR5e is shown. Due to the fact that the end code for individual joint angles θ_{1-6} is too large and complex to display, an example of the end result with the given position and orientation of the end-effector ${}^{0}_{6}T$ is shown.

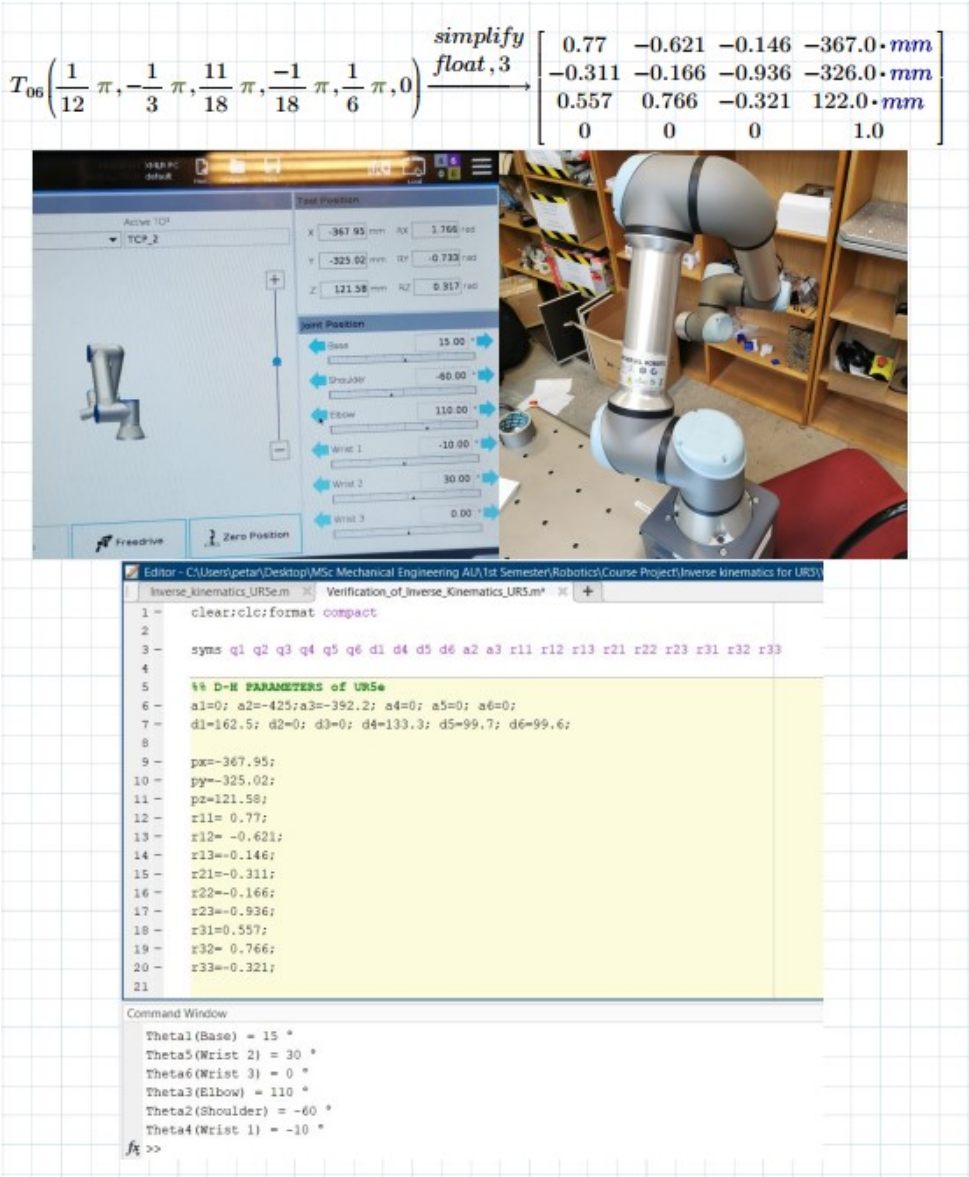


Figure E.1. Verification of inverse kinematics for UR5e

```
clear;clc;format compact

syms q1 q2 q3 q4 q5 q6 d1 d4 d5 d6 a2 a3 px py pz rx ry rz r11 r12 r13 r21 r22 r23 r31 r32 r33

%% INPUT:
% Desired position and orientation of end - effector (T_06) in shape:
% px=
```

```
% py=
% pz=
% r11=
% r12=
% r13=
% r21=
% r22=
% r23=
% r31=
% r32=
% r33=

%% D-H PARAMETERS of UR5e
% "q" sign used instead of "theta" in purpose of simplification of code
a1=0; a2=-425;a3=-392.2; a4=0; a5=0; a6=0;
d1=162.5; d2=0; d3=0; d4=133.3; d5=99.7; d6=99.6;

T_01 = [cos(q1) -sin(q1) 0 0; sin(q1) cos(q1) 0 0; 0 0 1 d1; 0 0 0 1];
T_12 = [cos(q2) -sin(q2) 0 0; 0 0 -1 0; sin(q2) cos(q2) 0 0; 0 0 0 1];
T_23 = [cos(q3) -sin(q3) 0 a2; sin(q3) cos(q3) 0 0; 0 0 1 0; 0 0 0 1];
T_34 = [cos(q4) -sin(q4) 0 a3; sin(q4) cos(q4) 0 0; 0 0 1 d4; 0 0 0 1];
T_45 = [cos(q5) -sin(q5) 0 0; 0 0 -1 -d5; sin(q5) cos(q5) 0 0; 0 0 0 1];
T_56 = [cos(q6) -sin(q6) 0 0; 0 0 1 d6; -sin(q6) -cos(q6) 0 0; 0 0 0 1];

T_06 = T_01*T_12*T_23*T_34*T_45*T_56;
T_06 = simplify(T_06);


T_06(1,1)=r11;
T_06(1,2)=r12;
T_06(1,3)=r13;
T_06(2,1)=r21;
T_06(2,2)=r22;
T_06(2,3)=r23;
T_06(3,1)=r31;
T_06(3,2)=r32;
T_06(3,3)=r33;
T_06(1,4)=px;
T_06(2,4)=py;
T_06(3,4)=pz;


%% Finding theta1(q1)
%We Compute given joint angles theta, from 1 to 6 with
%given the position and orientation of end-effector T_06
% To find theta1, we begin with finding the location of frame 5 in relation
% to the first frame, which is also at the same place as base frame
% First we need to find origin of the 5th frame,P_05org

P_05org=T_06*[0;0;-d6;1];
P_05org_x=P_05org(1,1);
P_05org_y=P_05org(2,1);

psi=atan2(P_05org_y,P_05org_x);
fi=acos(d4/sqrt((P_05org_x)^2+(P_05org_y)^2));
q1=psi+fi+pi/2;
q1=round(q1,20);
```

```
theta1 = rad2deg(q1);
theta1=round(theta1,1);
fprintf('Theta1(Base) = %.f ', theta1); disp('')
% We have two possible solutions fo q1 which corresponds to the shoulder
% being "left" or "right". Equation for q1 has solution in all cases
% except that d4>P_05org. That happens when origin of 3rd frame is close to
% the z axis of frame 0.

%% Finding theta5(q5)
% If we know q1, we can solve for q5
P_06org=T_06(1:3,4);
P_06org_x=P_06org(1,1);
P_06org_y=P_06org(2,1);
P_16org_z=P_06org_x*sin(q1)-P_06org_y*cos(q1);

q5=acos((P_16org_z-d4)/d6);
theta5 = rad2deg(q5);
theta5=round(theta5);
fprintf('Theta5(Wrist 2) = %.f ', theta5); disp('')
%Again, there are two possible solutions, corresponding to the wrist being
%"down" and "up"

%% Finding theta6(q6)
%If we ignore translations between frames, z1 can be represented with
%respect to frame 6 as a unit vector defined with spherical coordinates. We
%can find the x and y components of this vector projecting it onto the x-y
%plane and then onto the x and y axes

%First we find transformation from frame 6 to frame 1
T_60=(T_06)^-1;
X_60_x=T_60(1,1);
X_60_y=T_60(2,1);
Y_60_x=T_60(1,2);
Y_60_y=T_60(2,2);

q6=atan2((-X_60_y*sin(q1)+Y_60_y*cos(q1))/sin(q5),(X_60_x*sin(q1)-Y_60_x*cos(q1))/sin(q5));
theta6 = rad2deg(q6);
theta6=round(theta6);
fprintf('Theta6(Wrist 3) = %.f ', theta6); disp('')



%% Finding theta3(q3)
% we consider the three remaining joints as forming a planar 3R
% manipulator. First we find location of frame 3 with respect to frame 1

%T_14=T_16*T_64
T_16=((T_01)^-1)*(T_06);
T_14=T_16*(T_45*T_56)^-1;
T_14=subs(T_14);
P_14org_x=T_14(1,4);
P_14org_y=T_14(2,4);
P_14org_z=T_14(3,4);
P14=sqrt(((P_14org_x)^2)+((P_14org_z)^2));

q3=acos((((abs(P14))^2)-((a2)^2)-((a3)^2))/(2*a2*a3));
theta3 = rad2deg(q3);
```

```
122 theta3=round(theta3);
123 fprintf('Theta3(Elbow) = %.f ', theta3); disp('')
124
125
126 %% Finding theta2(q2)
127 delta=atan2(-(P_14org_z),-(P_14org_x));
128 epsilon=asin((-a3*sin(q3))/abs(P14));
129 q2=delta - epsilon;
130 theta2 = rad2deg(q2);
131 theta2=round(theta2);
132 fprintf('Theta2(Shoulder) = %.f ', theta2); disp('')
133
134
135 %% Finding theta4(q4)
136 %T_34=T_31*T_14
137 T_34=((T_12*T_23)^-1)*T_14;
138 T_34=subs(T_34);
139 X_34_x=T_34(1,1);
140 X_34_y=T_34(2,1);
141 q4=atan2(X_34_y,X_34_x);
142 theta4 = rad2deg(q4);
143 theta4=round(theta4);
144 fprintf('Theta4(Wrist 1) = %.f ', theta4); disp('')
```