TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Patrick Laansalu 221875IASM

COLLABORATIVE INDUSTRIAL ROBOT WORKSTATION LOCALIZATION LOGIC USING LASER SENSOR

Master's thesis

Supervisor: Andres Rähni
MSc
Co-supervisor: Kadir Mert Unlu

Tallinn 2024

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Patrick Laansalu 221875IASM

KOLLABORATIIVSE TÖÖSTUSROBOTI TÖÖJAAMA LOKALISEERIMISE LOOGIKA LASERANDURI BAASIL

Magistritöö

Juhendaja: Andres Rähni
MSc
Kaasjuhendaja: Kadir Mert Unlu

Tallinn 2024

Author’s Declaration of Originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Patrick Laansalu

30.04.2024

Abstract

Localization is a critical operation in industrial robotics, ensuring the precise positioning and handling of products by robots. This thesis explores the use of laser sensors to enhance localization accuracy in robot work process. The study focuses on implementing laser sensor-based localization in two different robot workstations. Both robots must localize the product and create product plane based on the laser measurement results. Specifically, during the robots work the orientation of the robot tool must be 90 degrees relative to the robot base in the first workstation and relative to the product plane in the second.

All the work in this thesis was done on collaborative industrial robots UR10e. Thesis work shows that using lasers for localization gives highly accurate measuring results and operates without localization errors.

The thesis is written in English and is 36 pages long, including 6 chapters, 30 figures and 5 tables.

Annotatsioon

KOLLABORATIIVSE TÖÖSTUSROBOTI TÖÖJAAMA LOKALISEERIMISE LOOGIKA LASERANDURI BAASIL

Lokaliseerimine on kriitiline protsess tööstusrobootikas, mis tagab täpse toodete positsioneerimise kasutatava roboti jaoks. See lõputöö uurib võimalust lokaliseerimise protsessi tugevdamiseks kasutades toote lokalisserimiseks laserandurit. Uuring keskendub laseranduri baasil lokaliseerimise lisamisele kahele erinevale roboti tööjaamale. Mõlemad robotid peavad lokaliseerima toote ning looma tootepõhise tasandi, mis põhineb laseranduri abil mõõdetud mõõtetulemustel. Esimese tööjaama robot liigub tööriistaga toote peal 90-kraadise nurgaga, lähtudes roboti enda tasandist. Teise tööjaama robot liigub tööriistaga toote peal 90-kraadise nurgaga lähtudes toote arvutatud tasandist.

Antud lõputöö on tehtud UR10e roboti baasil. Lõputöö näitab, et laseri kasutamine toote lokaliseerimisel annab täpsed mõõtetulemused ning robot töötab ilma erroriteta, mis tulenevad lokaliseerimisest.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 36 leheküljel, 6 peatükki, 30 joonist ja 5 tabelit.

List of Abbreviations and Terms

TCP	Tool Center Point
2D	Two dimensional
3D	Three dimensional
Rx	Coordinate that displays rotation around x axis
Ry	Coordinate that displays rotation around y axis
Rz	Coordinate that displays rotation around z axis.
GUI	Graphical User Interface
UR	Universal Robots
RPC	Remote Procedure Call
Cobot	Collaborative robot

Table of Contents

List of Figures

List of Tables

1. Introduction

Nowadays robots are used widely in the production process all over the world and their usage will continue to grow. Based on statistics, in 2021 global Mobile Cobots Market was valued at 656.1 million USD and the market size is predicted to reach 7.660.4 million by 2023. [1]

The localization process is a crucial part of the industrial robot workflow, as it dictates the precise positioning of the product for robot operation. Industrial robot workstations employ various methods of product localization, each utilizing different hardware. For instance, options include 2D cameras, which capture images in two dimensions and provide flat object representations; 3D cameras, offering three-dimensional object representations; touch sensors, yielding analog/digital values upon product contact; ultrasonic sensors, utilizing sound waves to detect object presence and distance; and laser sensors, providing digital/analog values when the laser beam detects the product.

The problem that author is going to solve is based on the fact that the company lacked a consistent and sustainable localization logic for Universal Robot workstations. Different robot lines had their own localization methods, leading to inconsistency. For instance, some lines relied on 2D cameras for product localization, but due to fluctuations in product surface light reflection, these cameras often produced errors. Additionally, 2D cameras were unable to measure the height of the product, necessitating the use of separate laser sensors for height measurement on each robot. In some instances, within the company, certain robots operated without any localization process, relying on fixed product positioning within the robot workplace. This solution had also problems as individual products or fixation equipment often possess unique offsets. Consequently, when high accuracy was required, the robot frequently encountered difficulties and failed to execute the process successfully.

The author has proposed a solution to use laser sensor for collaborative robot localization process. This enables the company to have one consistent localization logic for all the collaborative robot workstations. Since the products that laser sensor must measure variable surface reflectance it is not certain that laser sensors are suitable for the localization process. Since there is no logic for laser sensor measurement localization available in the company, the author will program the logic for localization process.

Thesis will cover the testing of the laser sensor on real products to find out accuracy and possibility to use laser sensors for localizing surfaces that have no consistent reflection. Based on the test results, the author will choose the sensors for the projects. Two projects are done during thesis work where localization process is added to the two separate *UR* robot workstations. When localization process is added to the workstations author will perform tests on both workstations to ensure how well this way of localizing products works.

Next chapter covers the requirements that are set for the products and the robot workstations. Chapter 3 focuses on explaining the updates necessary for the robot so it can manage the required laser localization process. Chapter 4 has information on how the author chose the laser sensor suitable for the project. Chapter 5 shows the way program was created for localization process and in Chapter 6 the test results about localization process are explained and shown.

2. Requirements

For the laser localization project, the first step was to understand the required process, outcomes and select the correct hardware. This chapter explains the project and its requirements. The chapter is divided to requirements that are set for the product and requirements set for the robots and its workstations. The layout of the robot workstation can be seen on Figure 1.

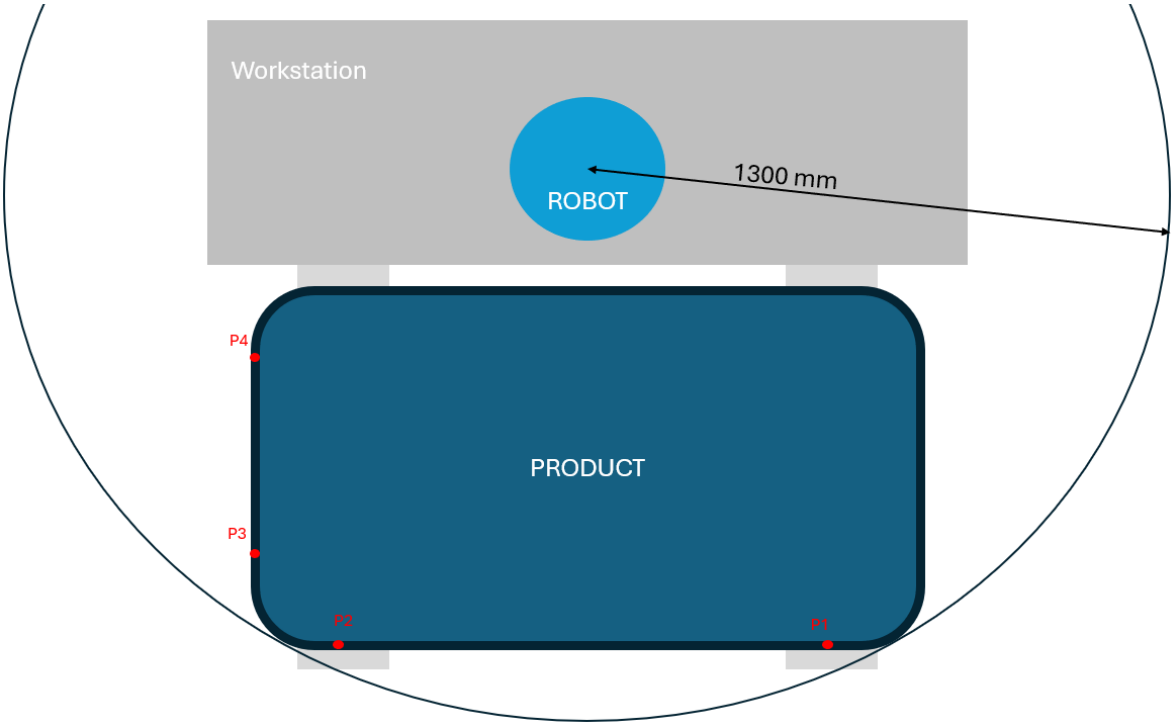

Figure 1. Top side view of robot workstation layout with product placement, robot reach 1300 mm and measurement points P1, P2, P3, P4.

2.1 Product

Laser localization logic that was programmed during this thesis works only for specific products. Main requirements for the products were following:

1. The shape of the product is rectangle or must have at least two measurable edges that have 90-degree angle between them like shown on Figure 1.
2. Product dimensions are enough for robots to take measurements of product edges. It means that from the robot's perspective, product dimensions must enable robot

to take two measurements of products outer edge and products right side edge as shown on Figure 1.

3. To get the correct measurement results robot must be able to measure height of the product in points *P1, P2, P3* and *P4*. The spots where the height is measured must share the same surface level.

2.2 Robot workstations

Laser measurement logic had to be added for two separate robot workstations that performed different tasks. Both workstations were using robot UR10e. This section explains both workstations and the requirements to the localization logic for both workstations.

2.2.1 Robot workstation 1

Workstation 1 robot had the task of dispensing material on the product. After the localization process robot had to dispense the material to the edges of product as shown on Figure 2**Error! Reference source not found.**. During the dispensing process, the dispensing tool that was mounted on the robot had to have vertical orientation based on robot base. It means that no matter how big of a tilt the product had, robot dispensing tool had to always have the same rotational orientation so *Rx, Ry, Rz* would always remain the same as visible on Figure 3.

Figure 2. Workstation 1 process shown on product. The red line near the edges of the product is shown as the material that workstation 1 robot dispenses on product.

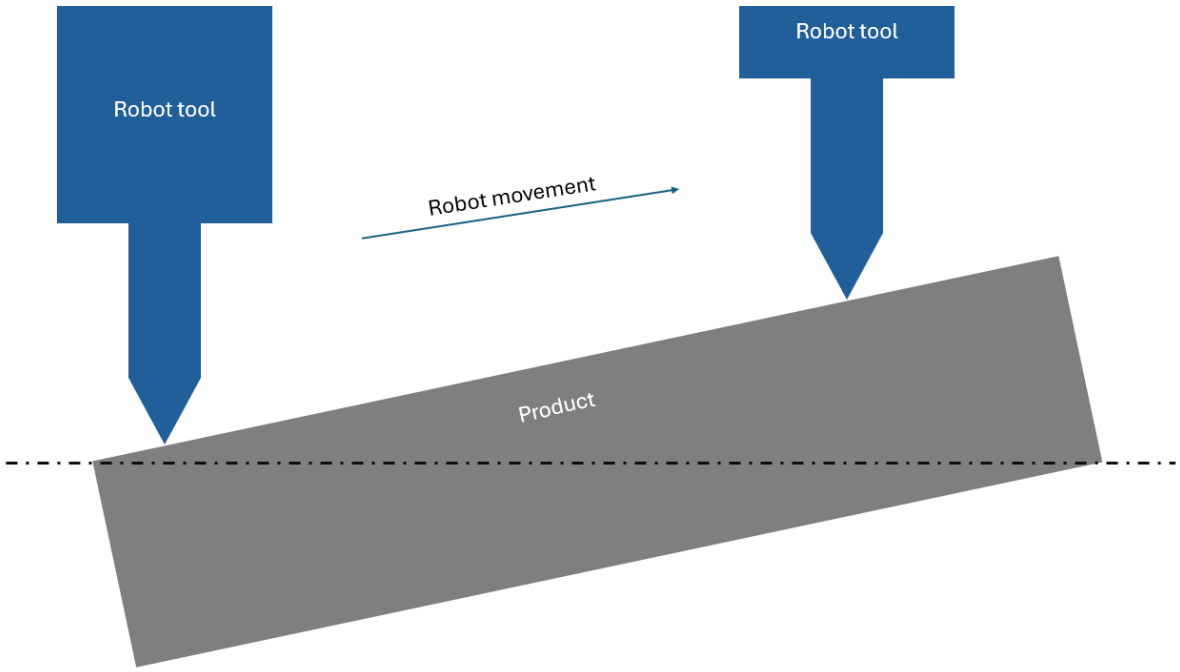

Figure 3. Robot tool orientation shown on tilted product on workstation 1. Tool will not take the products orientation but will remain with the same orientation as robots base.

2.2.2 Robot workstation 2

Workstation 2 placed tighteners to fix the components on the product shown on Figure 4. Tool of workstation 2 had to take the vertical orientation based on calculated

product plane. It means that when product was placed on the workstation with having tilted position then robot had to be able to give its tool the same rotational orientation. Because of that the *Rx, Ry, Rz* were not static but always changing with different product processes as shown on Figure 5.

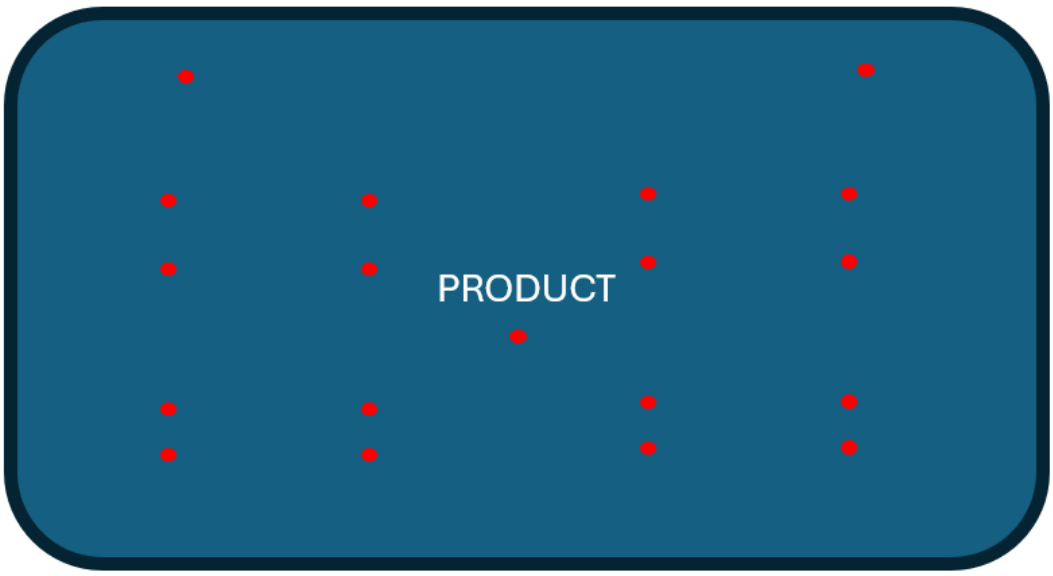

Figure 4. Visualized version of workstation 2 work on product. The red dots simulate places on product where robot must do the tightening process. This is just a visualized figure meaning that the tightening amount and places differ when compared to the real product process.

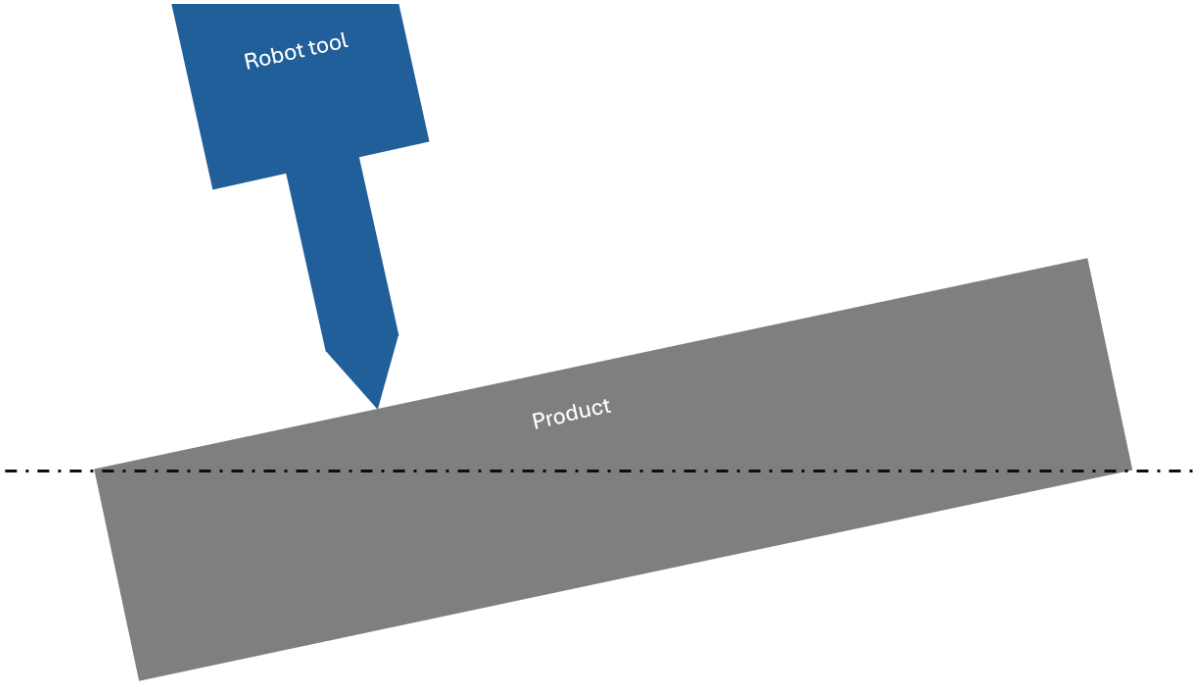

Figure 5. Robot tool orientation shown on tilted product on workstation 2. Robot will give its tool rotational orientation based on calculated product plane.

3. Updating the robots

This chapter explains the necessary updates that were done for the UR robots so they can support laser measurement logic. Chapter 3.1 has information about updates regarding polyscope version and Chapter 3.2 has updates regarding the changes in robots operating system.

3.1 Updating the polyscope software

UR robots have *GUI* software named polyscope. Using polyscope it is possible to do the programming of the robot. Polyscope can be seen at **Error! Reference source not found.**. It is recommended to always update to latest version of the software, as each new release will fix bugs, add new features, and in general improve performance [2]. Robots were updated to the latest polyscope version 5.15.

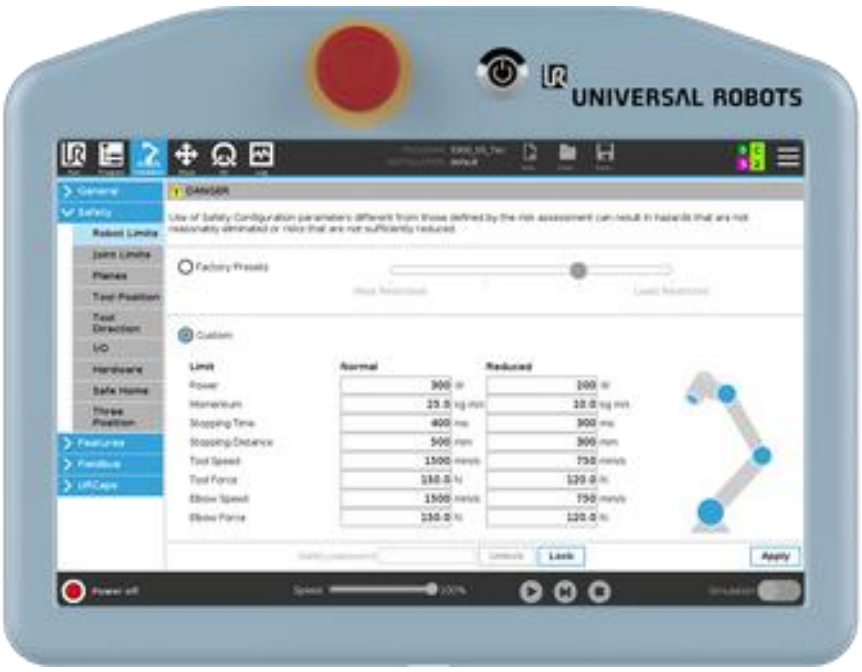

Figure 6. Universal Robot programming interface named polyscope [3].

3.2 Adding python packages to the robots operating systems

In addition to the polyscope software update polyscope operating system had to be modified. Polyscope is based on the Linux operating system with Debian GNU/Linux

distribution, and it uses python 2 programming language. To implement laser measurement logic there had to be NumPy and SymPy python libraries available for calculation. Procedure of work to get these libraries into *UR* robot Linux system were following:

1. Access the robot Linux terminal with connecting the keyboard to the polyscope and press following buttons together: *CTRL+ALT+F*
2. Log in as the root user. Default login and password for Universal Robots are:
   login: *root*
   password: *easybot*
3. Go to the folder where sources.list file is located: *cd /etc/apt*
4. Open sources.list file for modification: nano *sources.list*
5. Write following lines into *sources.list*:
   deb http://archive.debian.org/debian/ jessie main non-free contrib
   deb-src http://archive.debian.org/debian/ jessie main non-free contrib
   deb http://archive.debian.org/debian-security/ jessie/updates main non-free contrib
   deb-src http://archive.debian.org/debian-security/ jessie/updates main non-free contrib
6. Use following commands in the following order:
   apt-get update
   sudo apt install python-pip
   sudo apt install build-essential
   sudo apt install python-sympy
   sudo apt install python-numpy
7. Exit the Polyscope terminal screen by pressing together following buttons: *CTRL+ALT+F7*

4. Choosing the laser sensor

It was also necessary to choose the laser sensors for workstations. At first the choice had to be made between the analog laser sensor and the digital sensor. Project that was done during this thesis had to have a high accuracy measurement process. The maximum offsets that were allowed were ±1 mm. If the offsets were higher than ±1 mm the process cycle would fail or have unaccepted end-product quality. This chapter explains the process of choosing the laser sensors based on the specification and tests that author made. Panasonic Analog laser sensor that was included in the tests is shown at Figure 7.

Figure 7. Panasonic Analog laser sensor HG-C1200-P [4].

4.1 Laser sensor specification

In the specification analog and digital sensors both qualified for the project. Specification of those laser sensors are shown in **Error! Reference source not found.** and **Error! Reference source not found.**.

Table 1. Digital laser sensor characteristics [5].

Characteristics	**Characteristics value**
Company	Panasonic
Name	HG-C1200L3-P
Output	IO-Link
Measurement center distance (mm)	200
Measurement range (mm)	±80
Repeatability (µm)	200
Linearity	±0.2 %F.S.
Beam diameter (µm)	300

Table 2. Analog laser sensor characteristics [4].

Characteristics	**Characteristics value**
Company	Panasonic
Name	HG-C1200-P
Analogue output (current)	0 to 5 V
Analogue output (voltage)	4 to 20 mA
Measurement center distance (mm)	200
Measurement range (mm)	±80
Repeatability (µm)	200
Linearity	±0.2 %F.S.
Beam diameter (µm)	300

4.2 Testing the laser sensors

While testing the Digital laser sensor in the measurement process it was noticed that robot has offsets larger than ±1 mm. Because of this, test runs were made with both analog- and digital laser sensors. New programs were programmed into robot for testing purposes. One program was for the testing of analog laser sensor and the other program was made for the digital laser sensor testing. Based on these programs robot went to measure products height on one point 50 times. Robot movement speed and measurement point were the same in both testing programs. For digital sensor the Figure 8 flowchart shows the flowchart of the program created for testing.

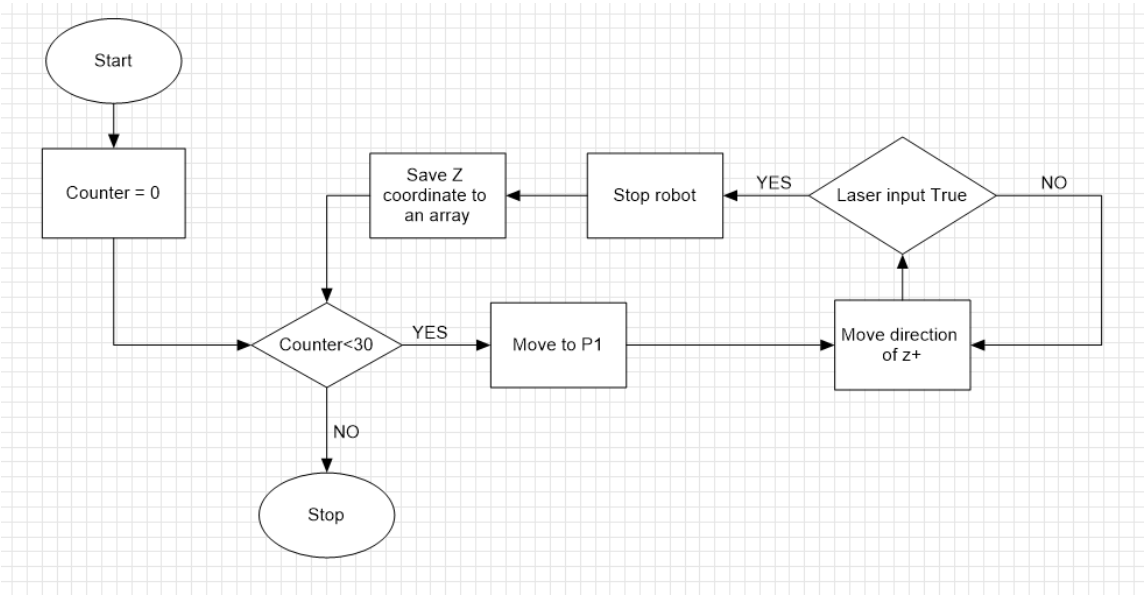

Figure 8. Flowchart of digital laser accuracy testing program.

When testing the analog sensor on **Error! Reference source not found.** it is shown that laser can be configured to show output as voltage or current. For testing it was decided to use electrical current as configuration since its measuring distance of 4-20 mA is wider than voltage 0-5V. The wider the measuring result distance the more accurate measurements can be received. Testing program of analog laser sensor is shown as flowchart in Figure 9. It is important to notice that the captured output of laser sensor was set to 0.012 (12mA).

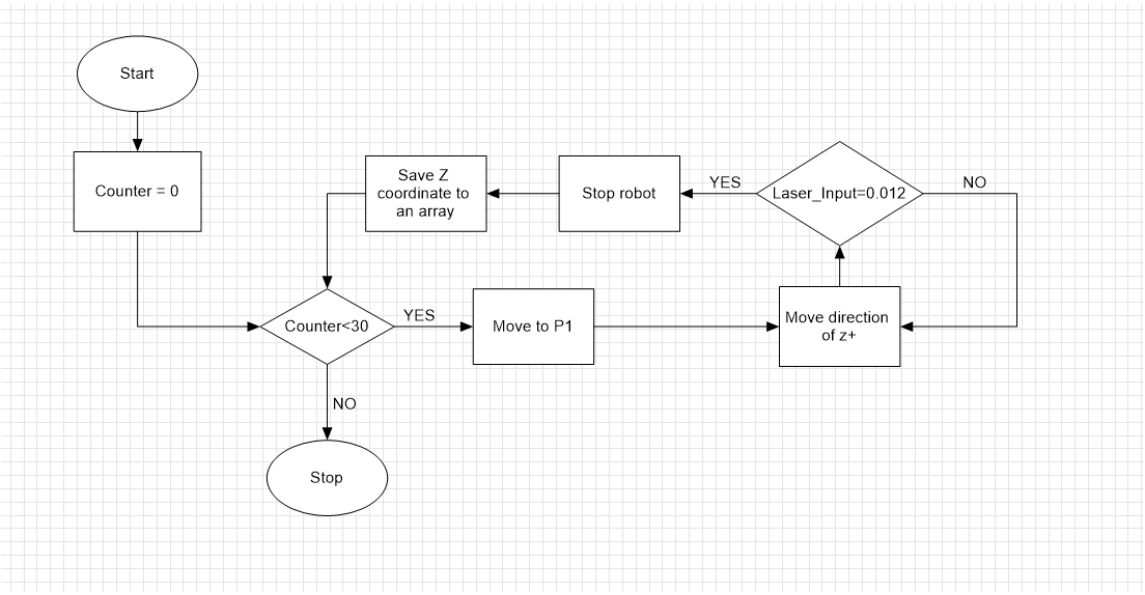

Figure 9. Testing program for analog laser sensor.

Testing results are shown in the **Error! Reference source not found.**. It was decided to use analog sensor for the project because test results were much more accurate with analog device. The reason of differences might be because of the surface reflection of product.

Table 3. Laser sensor testing results where max offset is calculated Z max – Z min.

Sensor	**HG-C1200L3-P**	**HG-C1200-P**
Type	Digital	Analog (Current)
Maximum offset (mm)	1.97	0.21

4.3 Choosing the laser sensor for workstations

Workstation 1 and workstation 2 had different tools that were different sizes. Figure 10 shows how laser was mounted to the robot according to the tool. When choosing the laser, it was important to know the *Tool Z offset* visible on Figure 10 according to the laser because different lasers have different measurement range and measurement distance.

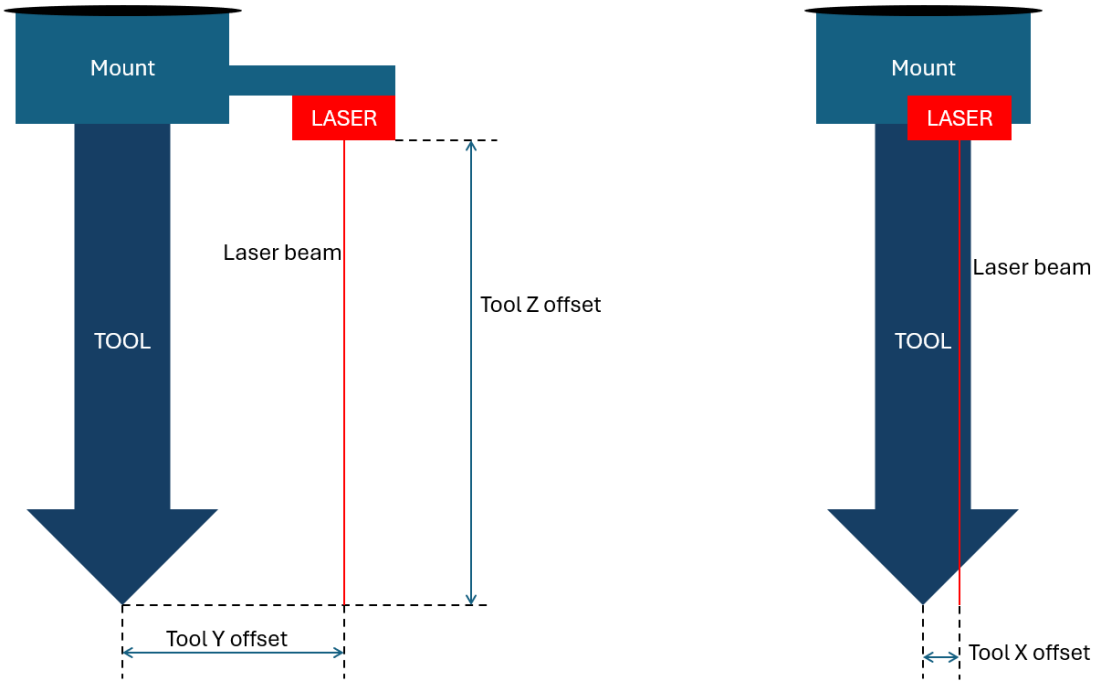

Figure 10. Laser location shown with the perspective of the tool. On the left side it is shown as front view where Y and Z offsets are displayed. On the right side it shows the side view where X offset is visible.

Table 4 shows the parameters of all the analog laser sensors available.

Table 4. Panasonic analog laser sensors specification based on measuring distance [6]

Parameter	**HG-C1030-P**	**HG-C1050-P**	**HG-C1100-P**	**HG-C1200-P**	**HG-C1400-P**
Measurement center distance	30 mm	50 mm	100 mm	200 mm	400 mm
Measurement range	±5 mm	±15 mm	±35 mm	±80 mm	±200 mm
Furthest measurement distance	35 mm	75 mm	135 mm	280 mm	600 mm
Nearest measurement distance	25 mm	35 mm	65 mm	120 mm	200 mm

Laser sensors selection was based on the measurement range of the laser sensor that is visible on Table 4 and based on *Tool Z offset* visible on Figure 10. For workstation 1 laser sensor HG-C1200-P was chosen and for the workstation 2 laser sensor HGC1100-P was selected.

5. Programming of the logic

This section explains the laser measurement logic that was implemented into both workstations. Laser measurement process consists mainly of two parts:

1. Product measurement and plane calculation
2. Calculating waypoint coordinates based on the calculated plain and robot tool *TCP*

Product measurement and plane calculation logic was the same for both workstations. Waypoint calculation process was different when considering two workstations because of the tool rotation requirements explained in the chapter 2.2.

This chapter is divided into sections that explain the flow of the programs that were made for both workstations, product measuring process logic, and plane calculation logic that was programmed for the robot.

5.1 Flow of the program

Programs of both workstations were divided into sections using URScript *Switch Case* statements since it makes it possible to program clear and understandable code as shown at Figure 11 and Figure 12.

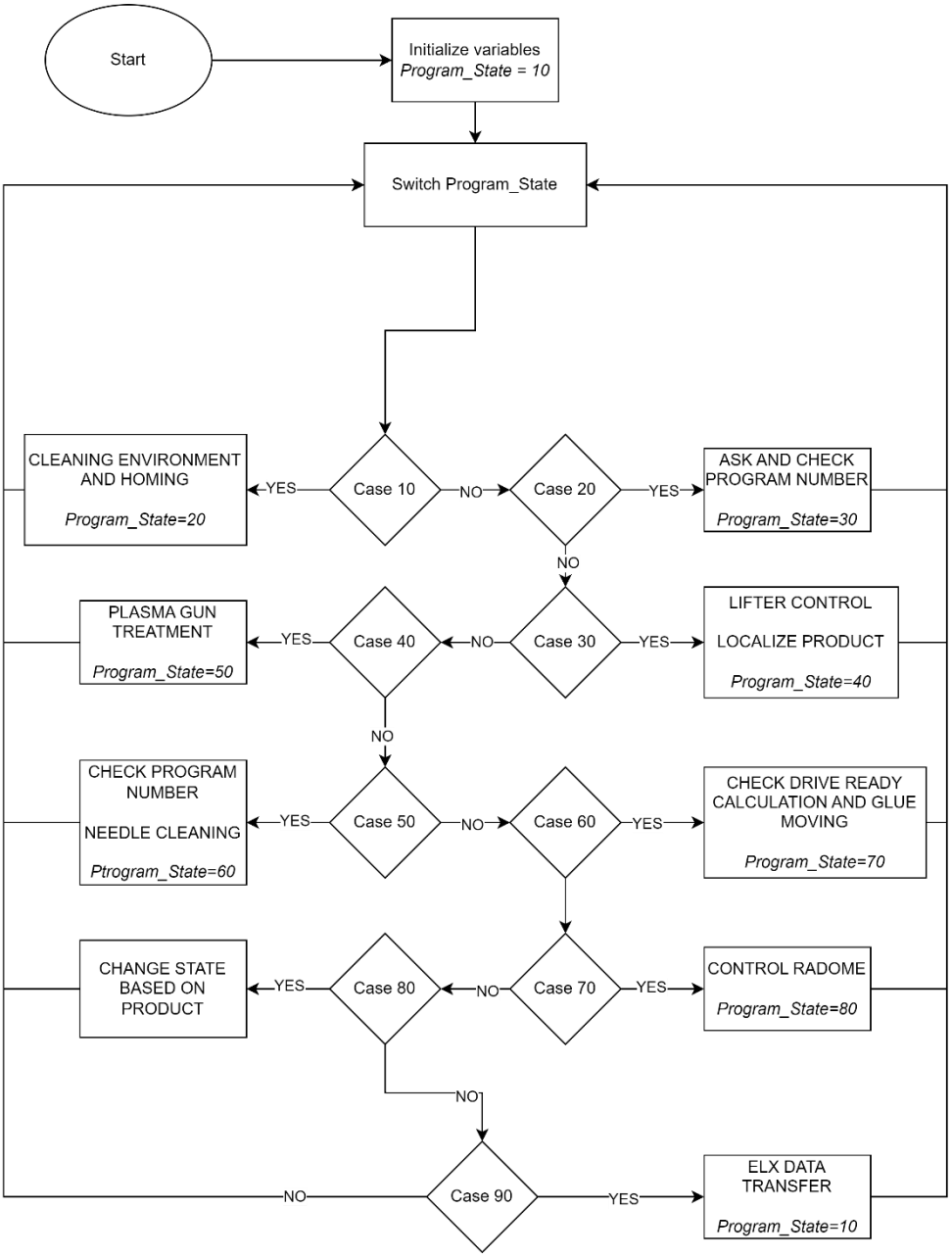

Figure 11 Flowchart of the program flow inside workstation 1 robot.

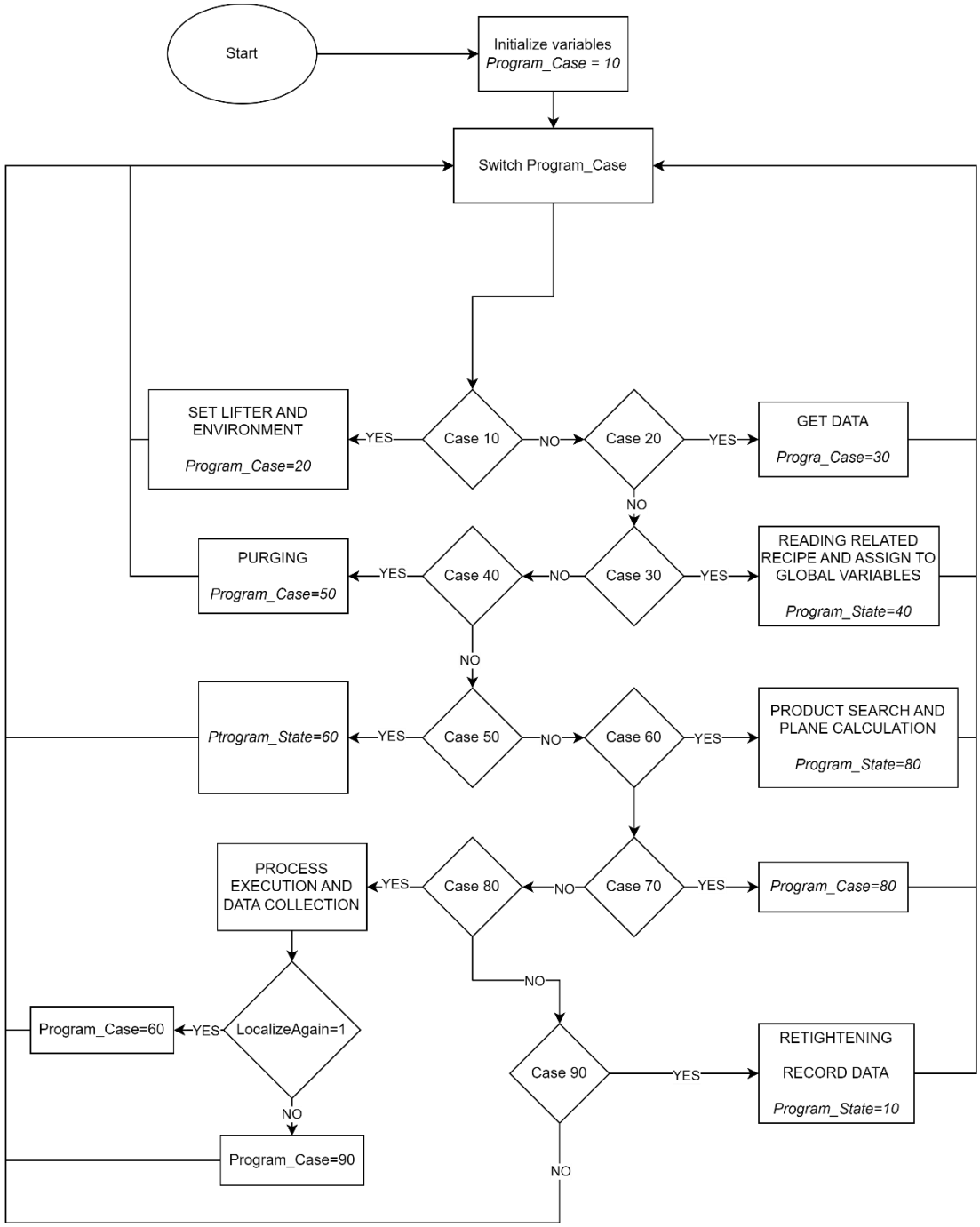

Figure 12. Flowchart of the program flow inside workstation 2 robot.

5.2 Measuring product with laser

Four waypoints were defined in the robot code to measure the product. At the start of the measurement step robot moves to the first measurement position based on the defined tool TCP in robot's base coordinate system. Waypoints P1, P2, P3, P4 were defined

manually by programmer by moving the robot in desired position and saving the robot's current position. Defining waypoints like that makes it easy to modify waypoints in the future. Location of waypoints P1, P2, P3, P4 are visible on Figure 13.

Figure 13. Waypoints at the start of the measurement process.

The flowchart of the measuring process is visible at Figure 14.

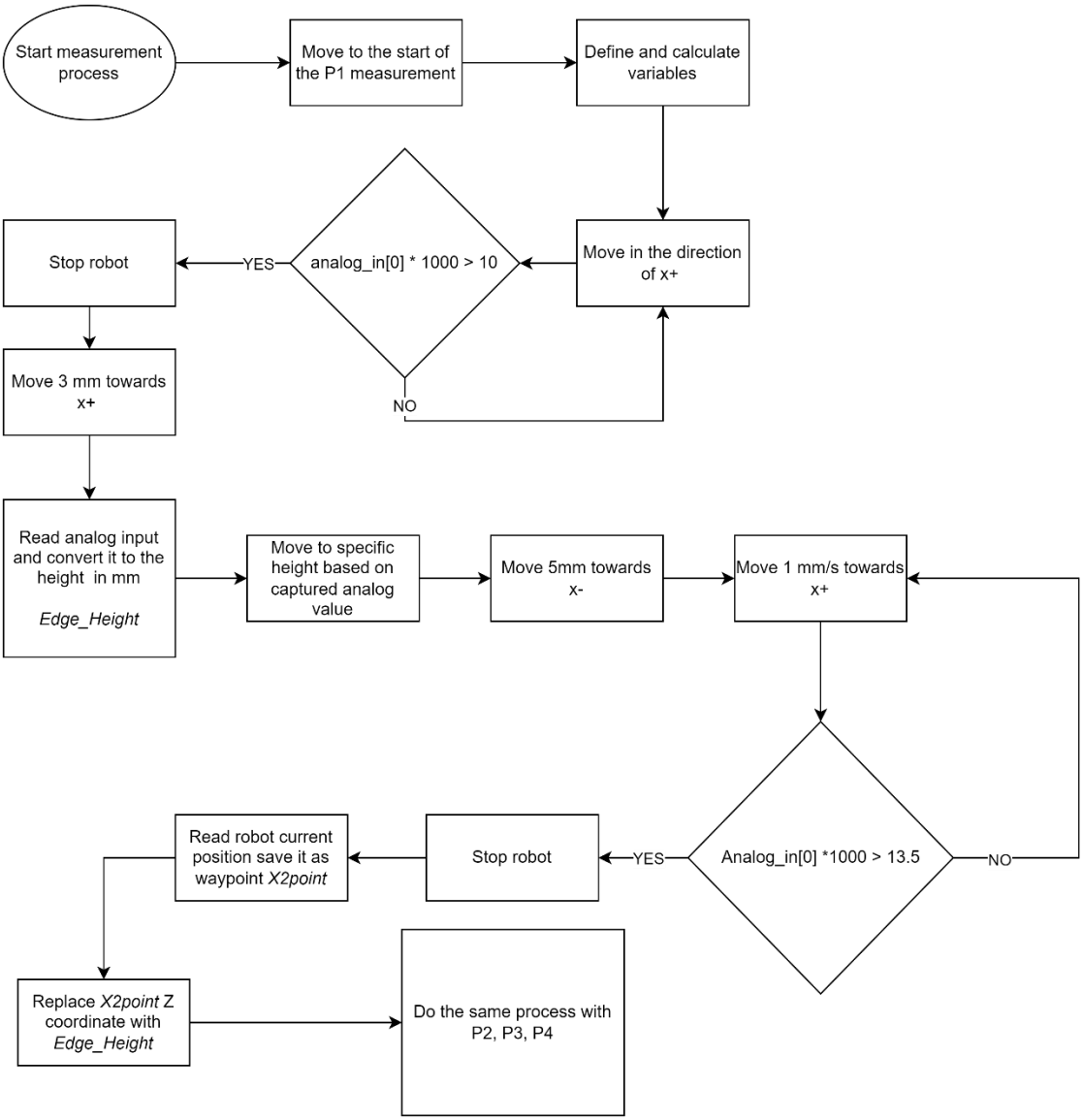

Figure 14. Flowchart of the measuring process.

5.2.1 Defining variables

First step in the measurement process is variable definition. Following variables were defined:

1. *First_H_Analog* – Height limit that is set for the first edge capturing process.
2. *Sensor_Min_H_mm* - Analog laser sensor minimum measuring distance.
3. *Sensor_Max_H_mm* – Analog laser sensor maximum measuring distance.
4. *Sec_H_Analog* – Height limit that is set for the second edge capturing process.

5. *Scale_Sens_Val* – Line equation calculated variable based on laser sensor measuring distance.

Figure 15 shows the declaration of the variables in polyscope code.

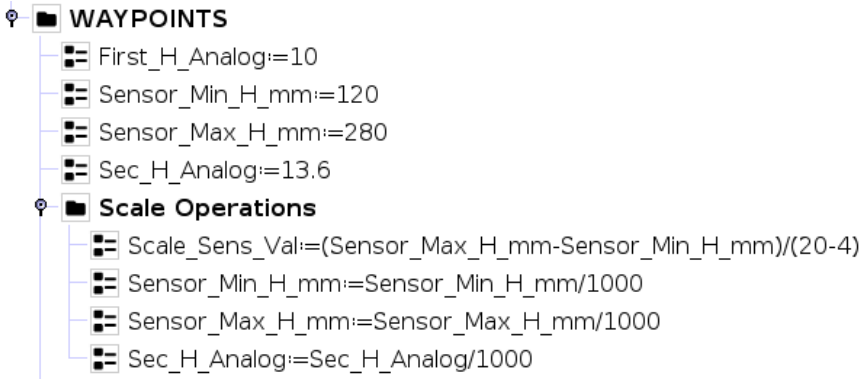

Figure 15. Defining necessary variables in Polyscope at the beginning of the measurement process.

To calculate received laser sensor mA value to the mm it is needed to use line equation. Defined *Scale_Sens_Val* is the variable that holds the constant value that is located inside the line equation:

1. Line equation formula:
$$\frac{y-y1}{y2-y1} = \frac{x-x1}{x2-x1}$$

2. Line equation based on laser minimum and maximum values:
$$\frac{y-120}{280-120} = \frac{x-20}{4-20}$$

3. After solving this the equation looks like this:
$$y = \big((20 - x) * 10\big) + 120$$

4. In this equation x will be the analog input robot receives, y represents the height value that laser sensor recorded, and number 10 is the constant value of variable *Scale_Sens_Val*.

5.2.2 Moving to the first waypoint and setting tool rotation

Since waypoints were defined manually laser might have rotational angle when faced downwards. This will result in inaccurate measurements since in different measurement

heights laser will catch product in different distance. To overcome this problem after moving to the start of the measurement point robot must create a new waypoint by getting the current location TCP coordinates and setting the rotational coordinates Rx, Ry, Rz to 0. This process is visible in Figure 16.

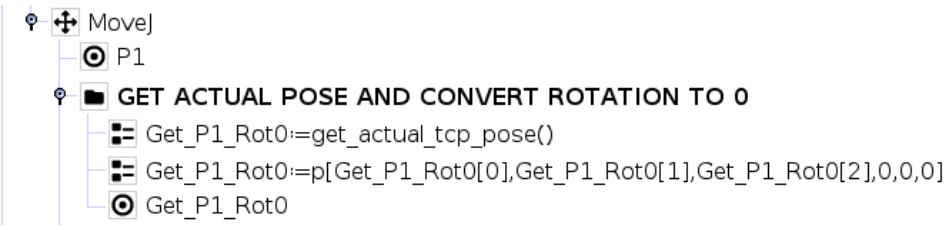

Figure 16. Part of the Polyscope code where robot moves to the location of P1, creates new waypoint Get_P1_Rot0 and moves to that point.

5.2.3 Height measurement

Next step for the robot is to start moving with linear movement in direction of the product until product is detected by laser. To detect a product using laser robots must read constantly its analog input where the laser is connected. The analog input value that laser will send to robot is shown as mA and the range is 4 mA to 20 mA. In the program robot will move towards the product until input value where laser is connected is over *First_H_Analog* variable value that is set to 10. Also, in case the edge is not detected robot will give error message. After the edge of the product is detected, robot will move 3 mm in the same direction so that laser will point in the center of products edge surface. Polyscope code of this process is shown at Figure 17.

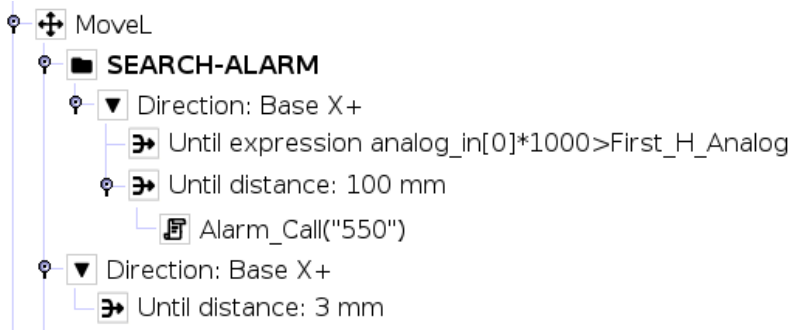

Figure 17. Part of the Polyscope code where robot moves to the direction of the product until it catches the edge.

At this point laser is pointing to the products edge surface as P1 in Figure 1. Next step for the robot is to measure height of the edge surface. Using bool variable *Height_AVG* main program will control height measurement thread. This thread reads lasers analog data

around 400 times during which it records minimum and maximum analog value. This process is necessary to make data recorded by laser sensor more accurate. Process of thread control is visible in Figure 18 and thread is visible in Figure 19.

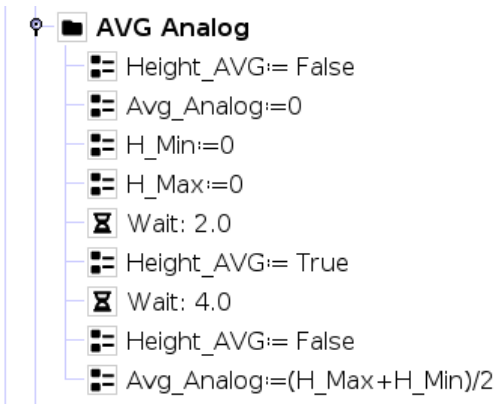

Figure 18. Controlling the thread and calculating average analog data with minimum and maximum values.

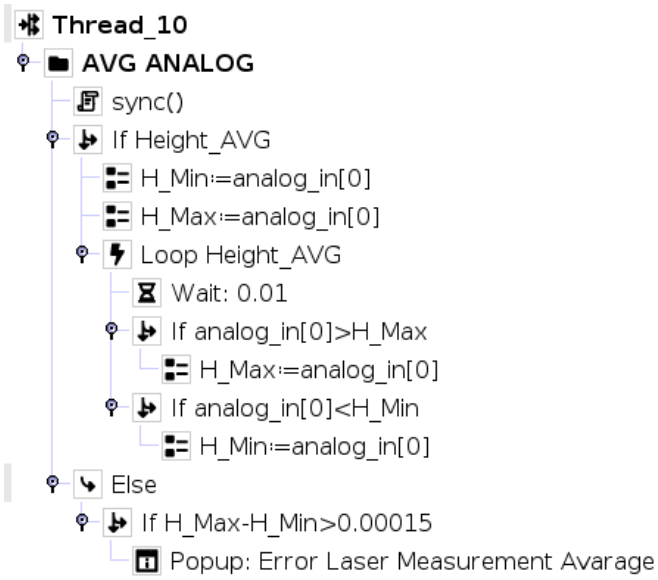

Figure 19. Thread that captures laser sensor minimum and maximum values.

After reading the laser sensor analog values robot will calculate variable Edge_Height. This variable represents the robot's waypoint coordinate Z in the respective laser sensor. It is calculated by converting laser sensor analog value into height in mm and subtracting

it from the robot's current height. Also, robot will be moved to the height based on the recorded analog value. It is important because this way when measuring X and Y coordinates of the four spots laser captures the product edge always around the same height. It makes the process more accurate. Lastly in this step robot moves 5mm away from the product to be ready to capture the edge X and Y. Code of this process is visible at Figure 20.

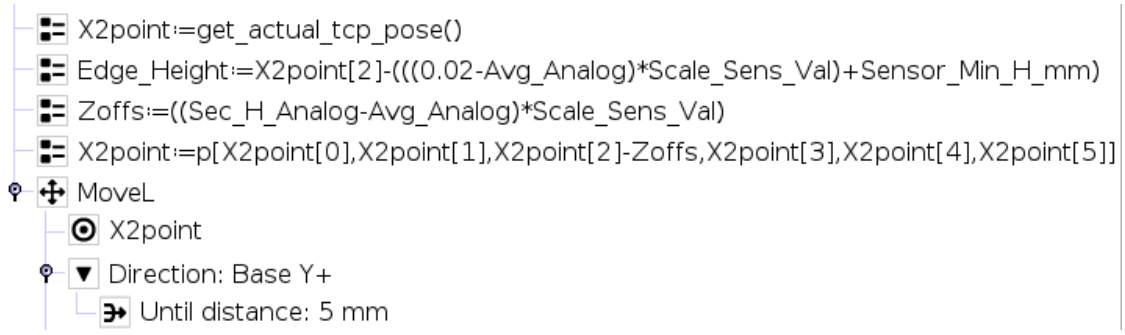

Figure 20. Calculating edge height and moving to the position to start measuring edge X and Y coordinates.

5.2.4 Measure edge X and Y coordinates

Last step in the products edge measuring process is capturing X and Y coordinates of the products edge. Now robot will start to move with the speed of 1 mm/s towards the product. When Edge is detected by robot it will stop, save its current coordinates and define a new waypoint where it will overwrite current height value by earlier calculated variable Edge_Height. The code of this part is visible at Figure 21. Now robot has successfully measured point P1 that is shown at Figure 13. Robot will repeat this process in rest of the points.

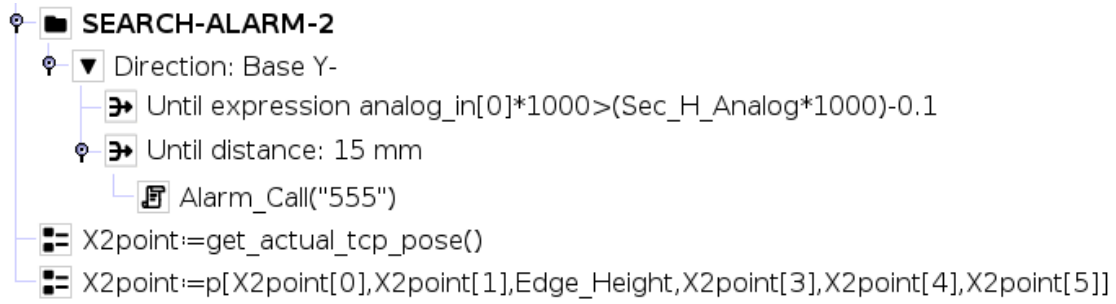

Figure 21. Measure X and Y coordinates of the product and save the data as waypoint.

5.3 Plane equation calculations

To calculate plane equation, it is necessary to first find the origin of the product. In this section robot will use functions from *RPC* server to make necessary calculations.

5.3.1 *RPC* Server

In the background of the robot *GUI* runs *RPC* server python script. This program contains functions that can be called from the polyscope. Having the possibility to use python functions with robot code makes programming Universal robots much more effective and flexible. For plane calculation the *RPC* server function named *Find_Cp_Dist(X1, X2, Y1, Y2)* is used to calculate origin of the product.

For origin calculation firstly program makes two vector equations in three-dimensional space based on the measured points. With the help of Eulers theorem the origin is calculated in the *RPC* server function and returned to the polyscope. Now the plain equation can be calculated using origin, P1 and P4. The location of origin on product is visible on Figure 22. The program of origin calculation can be seen on Appendix 2 and polyscope code is visible on Figure 23.

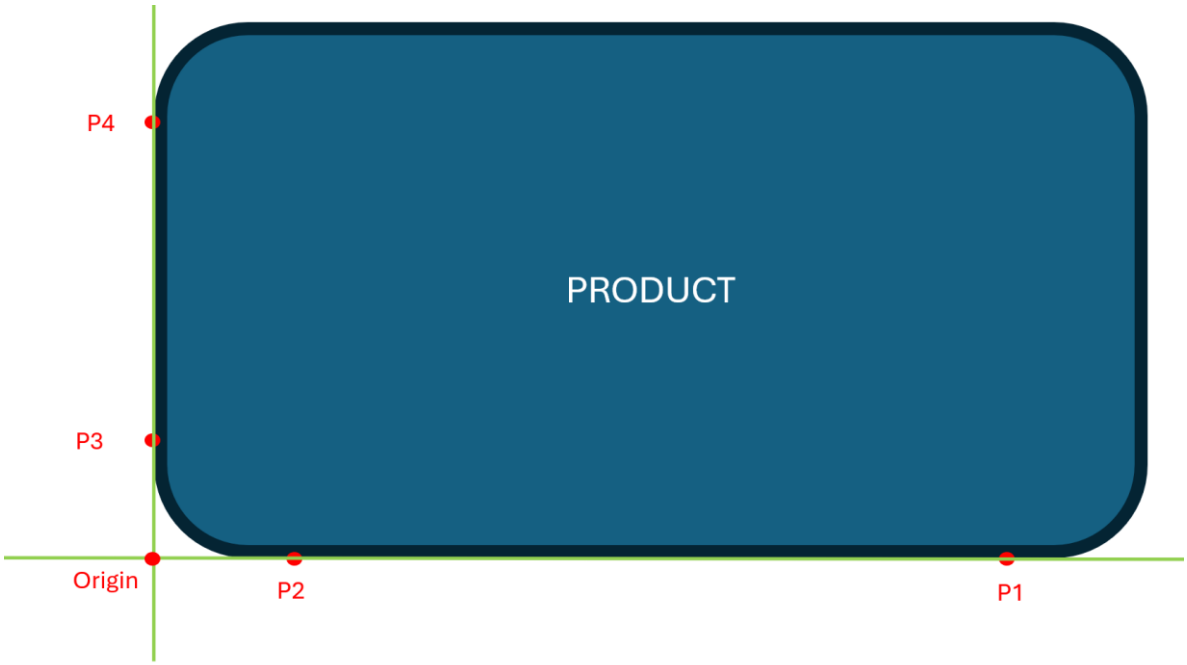

Figure 22. Product point of origin with 4 measurement points.

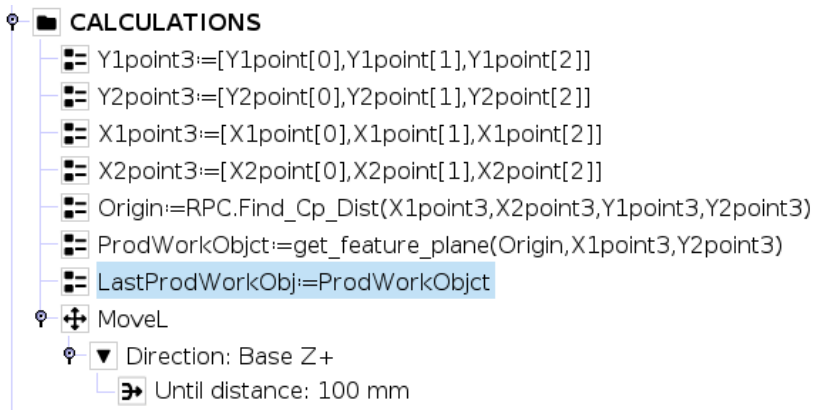

Figure 23. Polyscope code that calculates product plane equation.

5.4 Moving the robot based on product

Movement coordinates that robot receives are from the perspective of the calculated product plane origin. Getting X and Y coordinates about the product and defining them in the URScript programming language as a waypoint can be seen on Figure 24.

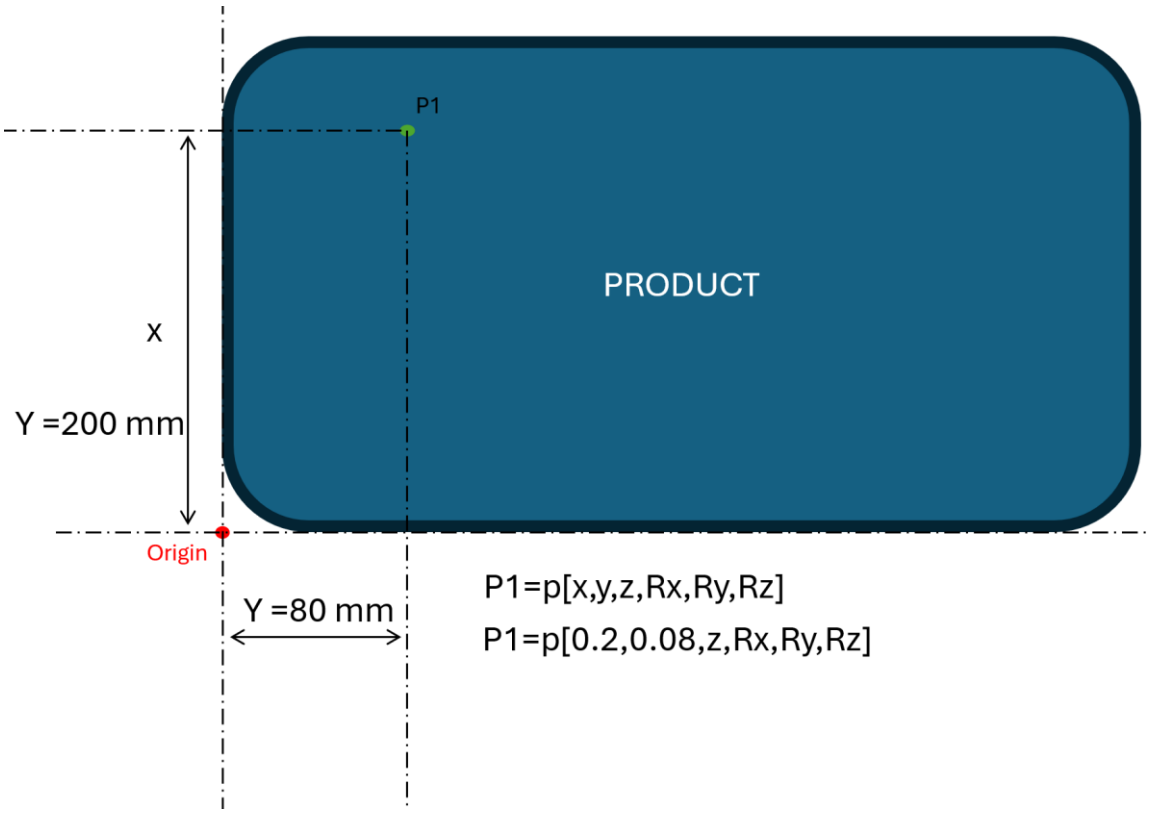

Figure 24. Defining point P1 X and Y coordinates based of the distance on product.

Coordinates that are defined for the products are based on the calculated product plane. Since robot will make its movements according to its base the given coordinates must be converted from the plane coordinate system to the robot's base coordinate system. This process is done with the help of URScript *pose_trans()* function that is based on transformation matrix multiplication.

„For calculation of kinematics, a transformation matrix can be defined as a 4-by-4 matrix, consisting of rotation matrix and position vector. The rotation vector and/or RPY will be converted to the rotation matrix. We can calculate the robot position and orientation based on the transformation matrix multiplication." [7]

„*pose_trans()* is using the principle of the transformation matrix. The calculated position and orientation is referred to the tool frame. With respect to *pose_add()*, the calculated position is the sum of two position inputs, but the resulted orientation is the matrix multiplication of two rotation matrix. In other words, in *pose_add()*, the position is corresponding to the base frame but the orientation is referred to the tool frame." [7]

Syntax of the function is following:

- *resulting_pose = pose_trans(*p_feature, p_wrt_feature*)* [8]

Parameters:

- *p_feature:* starting pose (spatial vector representing feature frame) [8]
- *p_wrt_feature: pose* relative to feature-frame (spatial vector w.r.t feature frame as new origin) [8]

Return Value:

- *resulting_pose:* pose relative to base-frame [8]

Both workstations have the same logic of converting the x, y, z coordinates, but differences are in rotation coordinates Rx, Ry, Rz.

5.4.1 Workstation 1 coordinate calculation

At the beginning of the calculation rotational variables are defined as a waypoint in robot as zero since workstation 1 robot must move on the product with vertical position as shown in Figure 3. Figure 25 below shows the definition of rotational coordinates.

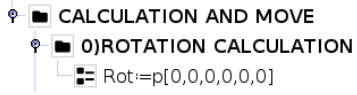

Figure 25. Defining rotational variables to zero as a waypoint.

After rotational coordinates definition movement waypoints are defined that are based on the product plane. Definition is visible in Figure 26. Array named *Product_Info* that is used to define coordinates of the waypoints consists of product dimensions what are explained on Figure 27.

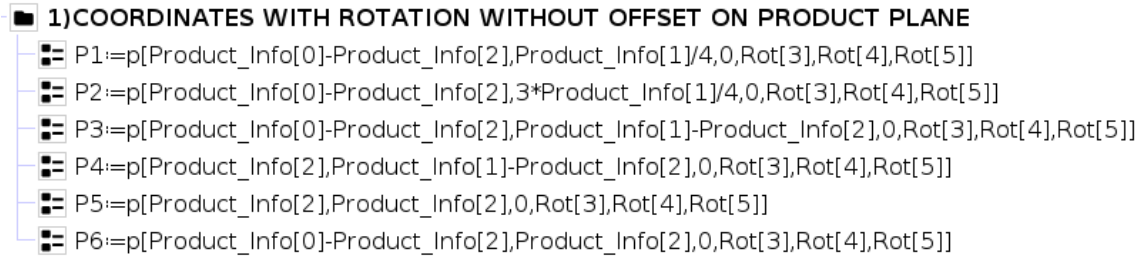

Figure 26. Defining 6 waypoints in Polyscope.

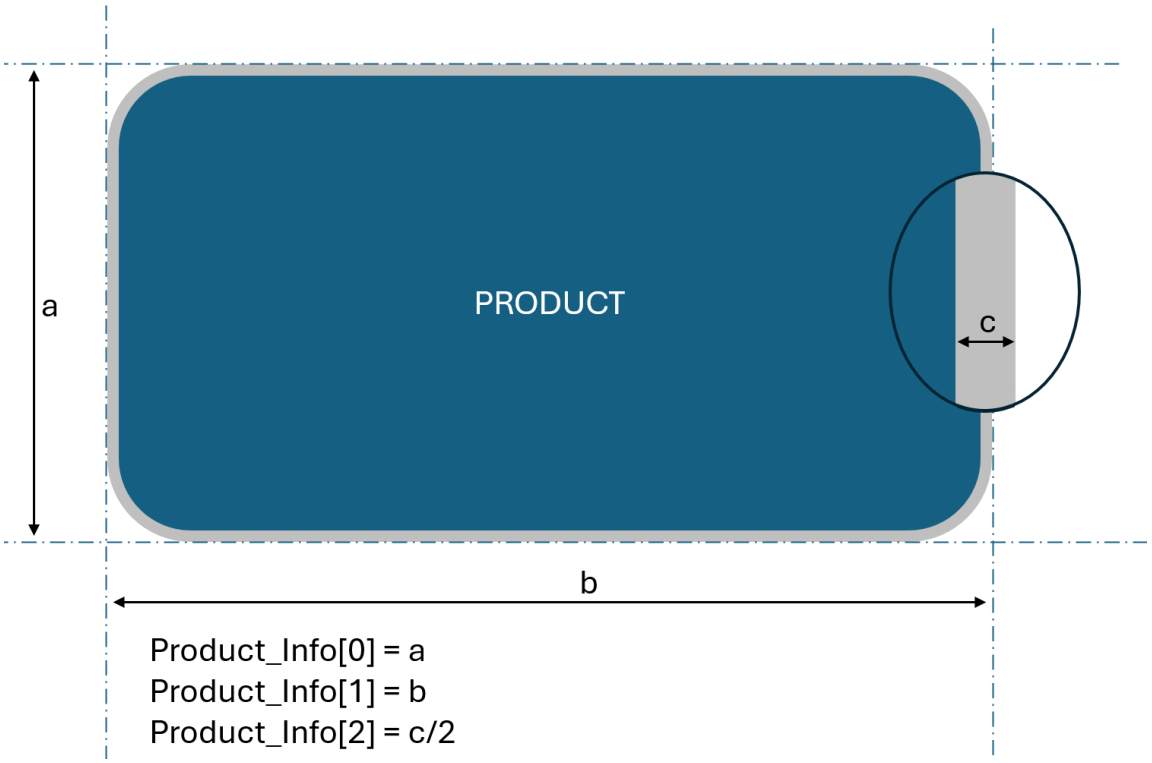

Figure 27. Product_Info variable explained with the simulated product.

After coordinates are defined based on the product it is needed to convert them from plane coordinate system to the base coordinate system. This process was performed with earlier described *pose_trans()* function. Conversion of coordinates is visible on Figure 28.

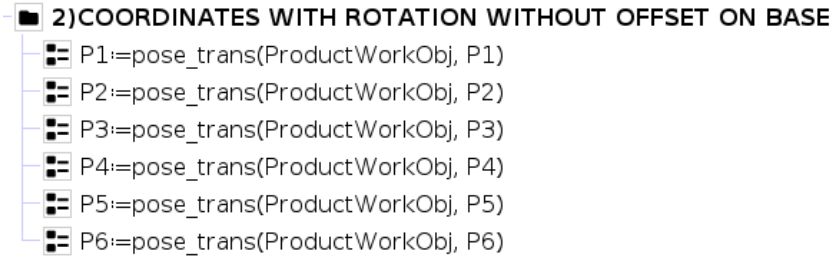

Figure 28. Using URScript pose_trans function to convert coordinates from the product plane coordinate system to the robot's base coordinate system.

At this point coordinates are converted to the robot's base coordinate system but since origin was measured according to the laser, the points are also defined according to the laser sensor. To define coordinates in respective of robot's tool tip, all the coordinates are given offsets from laser to robot's tool tip in base coordinate system. The offsets have constant value based on the robot workstation and are visible at Figure 10. Polyscope code where the offsets are added is visible on Figure 29.

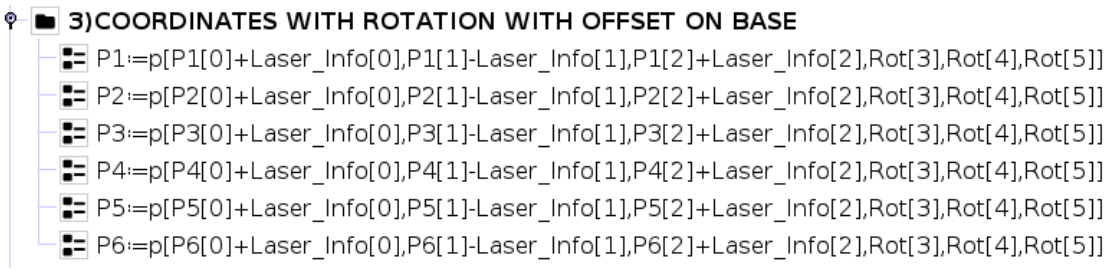

Figure 29. Offsetting coordinates in base coordinate system with Laser_Info array values

5.4.2 Workstation 2 coordinate calculation

In workstation 2 robot reads coordinates from the excel file and moves accordingly. This process is performed in the while loop what returns when all the waypoints are read from excel. With each loop cycle robot will:

1. Read coordinates from excel,
2. Perform calculations with coordinates,
3. Move to the calculated coordinate and perform the task.

Waypoint variables that workstation 2 robot uses are following:

1. *SafeStart* – location where robot goes before its target position. This waypoint is located above the *Pose* waypoint.
2. *Pose* – Target location of robot read from excel.
3. *SafeEnd* - location where robot goes in the end of the process. This waypoint is located above the *Pose* waypoint.

The calculations are similar to those in Chapter 5.4.1, but this time, rotational variables are also included in the *pose_trans()* function. The objective is to set rotational variables to zero, based on the product plane, and then convert them from product plane to base coordinates. This approach allows us to move the robot tool with the same tilt as the

current product plane. Coordinate calculations in workstation 2 robot are visible on Figure 30.

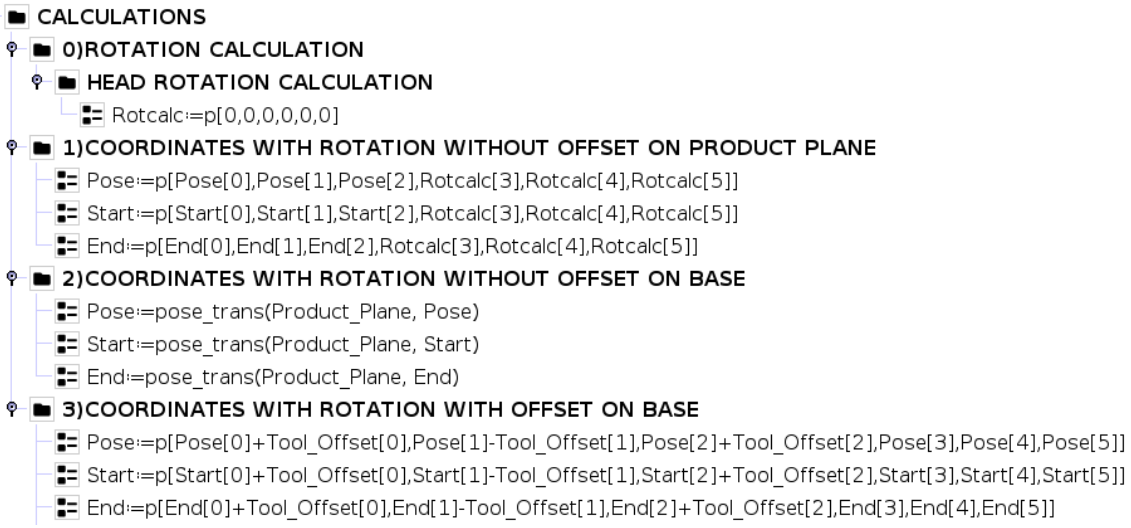

Figure 30. Coordinate calculation process on robot of workstation 2.

6. Testing of the program

In case of errors in laser localization process robot will move to the assigned locations with inaccurate coordinates. This chapter explains how both workstation project localization processes were tested and reveals the results.

6.1 Testing of workstation 1 localization

The localization process for Workstation 1 was tested on four different products by running it 30 times under normal conditions and 30 times with the products deliberately tilted. The accuracy of each process was assessed through visual inspection and the robot's error-handling mechanism. The error rate for Workstation 1 was 0, indicating that localization yielded correct results in 100% of cases.

6.2 Testing of workstation 2 localization

The localization process for Workstation 2 underwent testing 30 times under normal conditions and an additional 20 times with intentionally tilted products. Three different products were used for testing of workstation 2. During one trial, the robot was tasked with placing tighteners at 15 different locations on the product. In Workstation 2, even a small measurement inaccuracy can result in errors during tightener insertion, as the robot is unable to accommodate a deviation as small as 2 mm. The accuracy of the process was assessed through visual inspection, the robot's error-handling mechanism and correct placement of the tighteners. The error rate for Workstation 2 was 0, indicating that localization yielded correct results in 100% of cases.

6.3 Assessment of the testing results

Both workstation testing processes gave 100% of the time correct results. Based on this, the author can claim that the way laser measurement localization process was done during thesis is highly accurate. Testing process results are shown in Table 5.

Table 5. Testing results based on error rate where products are marked as *P*.

	Workstation 1								**Workstation 2**					
	Normal conditions				Tilted product				Normal conditions			Tilted product		
	P1	*P2*	*P3*	*P4*	*P1*	*P2*	*P3*	*P4*	*P1*	*P2*	*P3*	*P1*	*P2*	*P3*
Error rate	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	0				0				0			0		
Total error rate	0													

Summary

The goal of this thesis was to make a localization process based on laser sensor that works without constant errors happening in localization process. The localization process based on laser measurement had to give accurate product measuring results so that robots can provide a good end-product quality. Accurate localization process will enable the company to produce more units in the specific time limit and provide higher quality products.

Thesis work was based on the author's proposed solution. Thesis work consisted of choosing and evaluating the laser sensors and choosing acceptable sensors based on test results. After the laser sensor was selected, the author made programs for two different robot workstations that had different conditions for robot tool rotation. Workstation 1 robot had to have constant tool rotational position regarding the robot base. Workstation 2 robot had to have dynamic tool rotational position that was depending and changing based on products rotation.

After creating the programs, the author evaluated the accuracy of implemented localization processes. Test results gave remarkably high accuracy results with an error rate of zero. Based on these results author can conclude that using laser sensors for localization is suitable and highly accurate way of product localization in industrial process using collaborative robots.

Next step for author is to start implementing laser-based localization systems in other robot lines in the company that are based on UR robots. The author will keep improving the laser-based localization system to improve the cycle time of localization and to make it possible to use laser measurement localization with products that have different shapes.

References

[1] B. Thormundsson, "Mobile Coboss Market," Industry Verticals, 2023.

[2] universal-robots, 15 04 2024. [Online]. Available: https://www.universal-robots.com/download/software-ur20ur30/update/latest-polyscope-software-update-sw-5152-e-series-and-ur20ur30/.

[3] universal-robots, "PolyScope," 15 04 2024. [Online]. Available: https://www.universal-robots.com/products/polyscope/.

[4] Panasonic, "HG-C1200-P | CMOS type Micro Laser Distance Sensor HG-C," [Online]. Available: https://www3.panasonic.biz/ac/e/search_num/index.jsp?c=detail&part_no=HG-C1200-P. [Accessed 16 04 2024].

[5] Panasonic, "Micro Laser Distance Sensor for IO-Link [CMOS]," 06 2023. [Online]. Available: https://mediap.industry.panasonic.eu/assets/download-files/import/mn_hgc1000l_instruction_pid_en.pdf. [Accessed 16 04 2024].

[6] Panasonic, "MICRO LASER DISTANCE SENSOR HG-C SERIES," [Online]. Available: https://media.distrelec.com/Web/Downloads/18/31/panasonic_hg_c_2014_eng_tds.pdf. [Accessed 01 05 2024].

[7] Universal Robots, "EXPLANATION ON ROBOT ORIENTATION," 11 04 2024. [Online]. Available: https://www.universal-robots.com/articles/ur/application-installation/explanation-on-robot-orientation/#:~:text=For%20calculation%20of%20kinematics%2C%20a,on%20the%20transformation%20matrix%20multiplication.. [Accessed 04 30 2024].

[8] Universal Robots, "URSCRIPT: MOVE WITH RESPECT TO A CUSTOM FEATURE/FRAME," 30 04 2024. [Online]. Available: https://www.universal-robots.com/articles/ur/programming/urscript-move-with-respect-to-a-custom-featureframe/. [Accessed 30 04 2024].

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis

I Patrick Laansalu

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis “Development of A Camera controller prototype for drones”, supervised by Andres Rähni and Kadir Mert Unlu

   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that author also retains the rights specified in clause 1 of non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation

01.05.2024

Appendix 2 – *RPC* Server origin calculation

```
def Find_Cp_Dist(X1,X2,Y1,Y2):
    t = sympy.Symbol('t')
    s = sympy.Symbol('s')
    VX= [(X2[0]-X1[0]),(X2[1]-X1[1]),(X2[2]-X1[2])] ## Vector of
X Line
    VY= [(Y2[0]-Y1[0]),(Y2[1]-Y1[1]),(Y2[2]-Y1[2])] ## Vector of
Y Line
    CPX=[X1[0]+VX[0]*t,X1[1]+VX[1]*t,X1[2]+VX[2]*t] ## Closest
Point on X Line
    CPY=[Y1[0]+VY[0]*s,Y1[1]+VY[1]*s,Y1[2]+VY[2]*s] ## Closest
Point on Y Line
    if Y1!=Y2 :
        CPY_CPX=[(CPY[0]-CPX[0]),(CPY[1]-CPX[1]),(CPY[2]-
CPX[2])] ## Closest Points Vectors ( If This Value Equal=0 This
2 Lines Cross)
    else :
        CPY_CPX=[(Y1[0]-CPX[0]),(Y1[1]-CPX[1]),(Y1[2]-CPX[2])]


    CPY_CPX_VX= np.dot(CPY_CPX, VX) ## Using Dot Product .
Multiply 2 Vectors .
    ## CPY_CPX_VX= result = np.multiply(CPY_CPX, VX) ## This
should Equal=0 . Multiply with (PQ*VX) ( Because It should Be
Perpendicular)
    ## CPY_CPX_VX=(CPY_CPX_VX[0]+CPY_CPX_VX[1]+CPY_CPX_VX[2]) ##
This should Equal=0 . Multiply with (PQ*VX) ( Because It should
Be Perpendicular)
    CPY_CPX_VY= np.dot(CPY_CPX, VY) ## Using Dot Product .
Multiply 2 Vectors .
    ## CPY_CPX_VY= result = np.multiply(CPY_CPX, VY) ## This
should Equal=0 . Multiply with (PQ*VX) ( Because It should Be
Perpendicular)
    ## CPY_CPX_VY=(CPY_CPX_VY[0]+CPY_CPX_VY[1]+CPY_CPX_VY[2]) ##
This should Equal=0 . Multiply with (PQ*VX) ( Because It should
Be Perpendicular)


    equation1 = sympy.Eq(CPY_CPX_VX,0) ## Dot product of 2
vectors Should equal=0
    equation2 = sympy.Eq(CPY_CPX_VY,0) ## Dot product of 2
vectors Should equal=0



    solution_t1 = sympy.solve(equation1,s) ## To Find t value
leave s alone
```

```
    solution_t2 = sympy.solve(equation2,s) ## To Find t value
leave s alone


    solution_s1 = sympy.solve(equation1,t)
    solution_s2 = sympy.solve(equation2,t)


    if Y1!=Y2 :
        equation_s = sympy.Eq(solution_s1[0],solution_s2[0])
        equation_t = sympy.Eq(solution_t1[0],solution_t2[0])


        solution_s = sympy.solve(equation_s)
        solution_t= sympy.solve(equation_t)

        t=solution_t[0]
        s=solution_s[0]
    else :
        t=solution_s1[0]


    CPX = [expr.subs('t', t) for expr in CPX]
    CPY = [expr.subs('s', s) for expr in CPY]
    CPY_CPX = [expr.subs('t', t).subs('s', s) for expr in
CPY_CPX] ## Write New Value inside sub methot


    norm = math.sqrt(sum(x ** 2 for x in CPY_CPX)) ## Calculate
Vectors Length

    CPX=[float(CPX[0]),float(CPX[1]),float(CPX[2])]
    CPY=[float(CPY[0]),float(CPY[1]),float(CPY[2])]
    if Y1!=Y2 :
        Center=[(CPX[0]+CPY[0])/2,(CPX[1]+CPY[1])/2,(CPX[2]+CPY[
2])/2]
    else :
        Center=CPX


    return Center
```