Semi-automation of a spray painting robot

Daniel Jovanovski

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6064
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

Spray painting by hand is a strenuous task for the operator and his shoulder joints. Today this is how the procedure is done at the Metso Sweden AB factory in Trelleborg. This thesis will analyze the spray painting process and investigate and implement a semi-automated robot which can be remote controlled from a safe environment as well as automated. The implementation has a focus on using vision feedback with AR-technology together with Oculus Rift touch controllers for controlling the robot.

The thesis covers the entire process of investigating possibilities, design, construction and implementation of a Cartesian robot. Due to a large amount of custom-made parts with small batches the company sees no reason to implement a fully automated robot solution which is why a semi-automated solution is desirable. The vision is to be able to automate the standard products while the small batches can be painted using tele-operation from outside the spray box in a safe environment.

The thesis resulted in a proof-of-concept for a semi-automated solution which worked as intended, although for a fully functional work cell, a few modifications must be done as well as implementing a GUI which was not covered in this thesis.

Acknowledgments

I would like to express my appreciation to Metso Sweden AB and their personnel, especially the technical department which has assisted me with technical questions. A special thanks to Peter Ahlberg, Magnus Johansson, Alfred Carlsson and Per Andersson for assisting me whenever I was in need of help.

I am most grateful to my supervisors Prof. Anders Robertsson at Department of Automatic Control as well as Gunnar Lindstedt at IEA for helping me and dedicating their valuable time. Their input have been of great importance in order for me to complete this thesis.

Further on I would like to thank all the suppliers of parts who gave me student discounts on the parts for the robot. Without discount the construction would break the budget and a full scale robot would not have been developed.

Contents

1 Introduction

1.1 Background

Metso is a Finnish industrial group which focuses on technological solutions and products for the mining, oil, gas and recycling industry. They have several factories throughout the world but their Trelleborg site is focused on rubber and plastic products for their mining solutions.[1] Today a lot of their production is done manually without any automation solutions. In order to keep up with the technological developments this branch of industry stands in front of, they need to implement automation solutions with the spray painting robot being one of the first projects.

The main reason for this project is that the shoulder position while painting causes a lot of stress to the operator's shoulder joint.[2] In order to relieve the stress the company wants to implement some kind of robot to do the spray painting.

The company implemented a spray painting robot during the late '90s. This robot did not fulfill the requirements on the the paint job which led to a lot of extra work for the operator to spray paint the unpainted parts by hand. The robot sprayed unevenly and did not cover the products as it should. This led to a dissatisfaction with the robot which eventually left the robot unused.

Multiple attempts to find an automated solution have been made but due to the enormous assortment of products (+50 000) and low volumes on certain products the conclusion was that it was a too large investment. With an almost infinite return on investment time this investment could not be motivated. Surveys have been made to investigate the possibility of using a vision feedback system to identify products but this involves a lot of difficulties due to bad contrasting between products and the background.

1.2 Problem formulation

The purpose of this thesis is to analyze the current spray painting process and develop a way to semi-automate the spray painting process and to implement this. A semi-automatic solution in this case means having the possibility to let the operator control the robot inside the spray room without being in the room, while still having the possibility of autonomous painting. The focus in this thesis is to see if VR/AR-technology is a good vision feedback for the operator to see what he/she is doing and if the robot can be controlled as close to the manual process as possible.

Developing a robot for this problem is very specific for this environment and requires a lot of custom solutions with off-the-shelf products as well as some self-developed products. The problems to be solved will depend a lot on the chosen solution and construction, but there are a few main goals to be achieved which are listed below.

[1] Wikipedia, *Metso,* retrieved 2018-04-12
[2]Australian government, *Spray painting,* retrieved 2018-04-12

1. Implementing VR/AR as a visual feedback system.
2. Keeping the cost low (maximum of 100 000 SEK).
3. Developing an easy-to-use and intuitive robot.
4. Show a proof of concept for autonomous painting

1.3 Method

In order to solve the problems in section 1.2 a theoretical study is required which involves a few topics which are the following.

1. Which is the most suitable construction for this problem?
2. Which parts are optimal for the construction?
3. Which programming language should be chosen and how will the robot be programmed?

The order which the theoretical study will follow is 1 – 3 due to the reason that they depend on each other.

Following the theoretical study is the actual construction of the robot and development and implementation of software. This process will follow the order down below.

1. Building the construction.
2. Implementing motion control of the robot.
3. Implementing vision feedback system with VR/AR.
4. Implementing a proof of concept for autonomous painting

The robot will initially be developed, built and optimized in an office. After an evaluation and discussion with the company it will be decided if the robot should be tested in an industrial environment.

2 Designing the robot

2.1 Current spray painting process

Today the current spray painting is done by hand and the process follows five steps. The first step is loading the products on to a table attached to a chain which pulls the table and circulate these around the machine. The table can be seen in Figure 1. There are 10 tables circulating around the machine and the loading process is also done by hand. After loading the table, it will enter an oven to be heated to 70°C which is required for the paint to stick to the material. After a couple of minutes, the products are ready to be painted and the operator will therefore go inside the spray painting room and put on a safety mask. The room can be seen in Figure 2. When the operator is ready to paint he/she will then press a button and the table will enter the room. The operator sprays one layer of paint from one side and can then turn the table 90 degrees to spray from a different angle. The amount of layers and angles he/she needs to spray from depends heavily on the products as they all are different sizes and materials.

When the operator is satisfied with the result he/she will press a button and the table will be pulled out of the room for cool down. Some products then circulate this process once again for a second layer of paint. The products can be seen in Figure 3.

Since spray painting spreads a lot of particles in the air they have to be either removed or absorbed which is why the room is covered with three water walls and has a well-built ventilation system. The table rests over a bath with water which absorbs a lot of the paint particles which does not stick to the products or table. This water is filtered outside the room to remove particles for recycling.

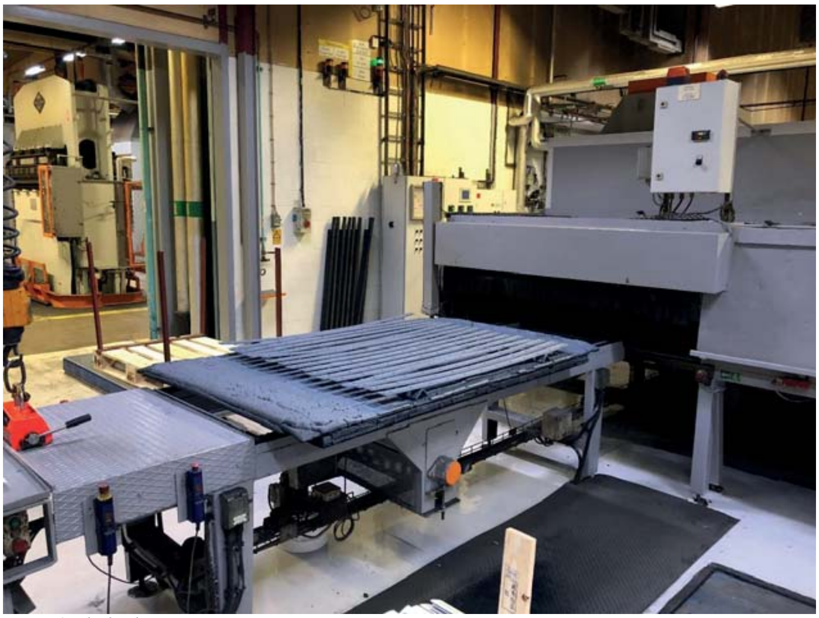


Figure 1: The loading area

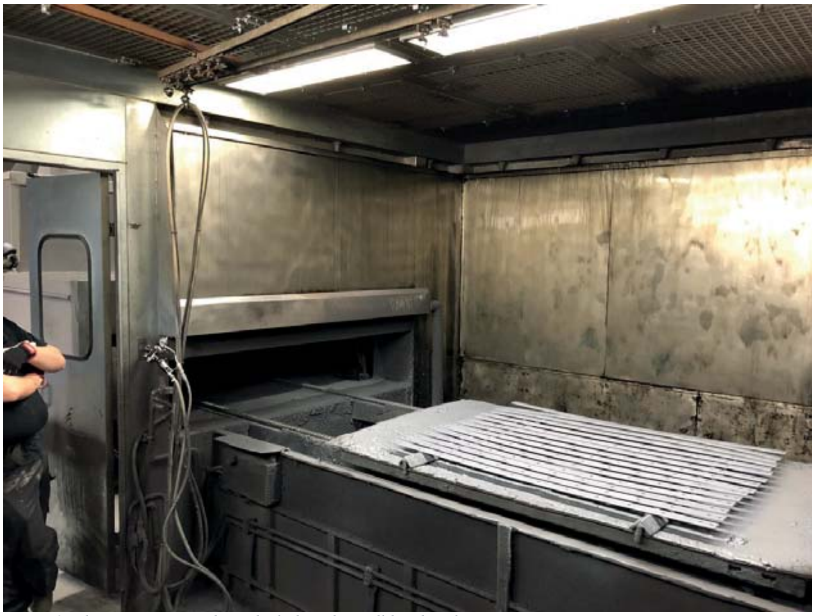


Figure 2: The spray painting box which the robot will be placed in

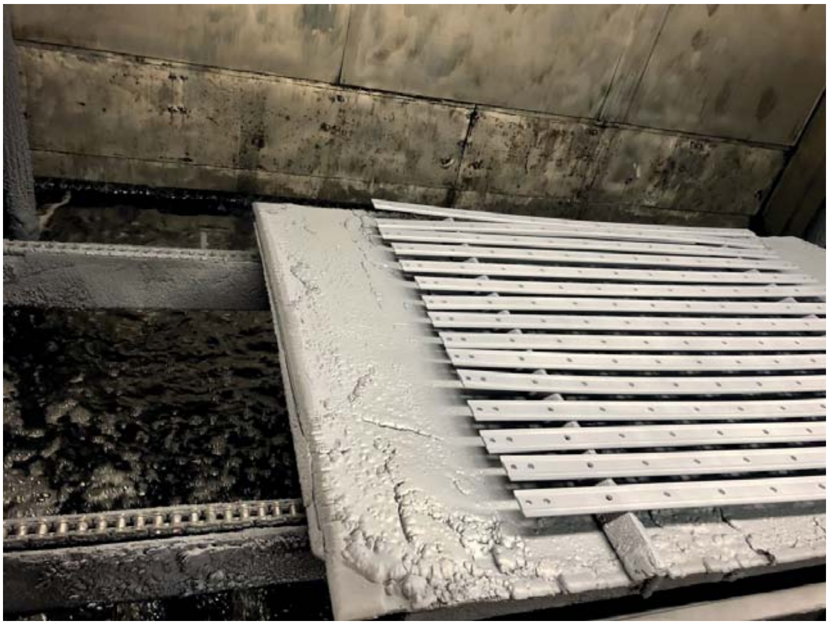


Figure 3: The one type of metals to be painted

2.2 Requirements for the semi-automated robot

Replacing a human with a robot comes with a few advantages as well as disadvantages. The robot will therefore need to fulfill a few requirements to be accepted by the operator as a replacement of manual painting but also needs to fulfill safety and economical requirements.

2.2.1 Requirements for spray painting

An interview was made with operator Robin Becker and he was asked to answer the following questions.

- What is the most important factor when spray painting?
- If implementing a robot, what do you think will be difficult?
- After hearing the basic idea, what is your input?
- From which angles and heights do you spray?

The answers can be found in Appendix 1.

By studying the operator while painting, a specification of the robot was made which is illustrated in Table 1.

Speed	1 m/s
Ramp-up time	0.1 s
Ramp-down time	0.1 s
Spray nozzle angle	± 45 ° around X and Y axis
Spray height/distance	20-40 cm
Spray nozzle angular speed	57 degrees/s

Table 1: Specification of performance and capabilities of the robot

By measuring the room and space where the robot is to be placed in Figure 2, dimensions for the product-table in Table 2 are illustrated.

Table height from floor	93 cm
Table width + extra spacing	130 cm
Table length + extra spacing	200 cm
Maximum width of room	500 cm
Maximum length of room	330 cm

Table 2: Table dimensions

Table 3 show the largest items to be painted. This is necessary to know for the gantry Z-axis dimensioning.

Largest object	300x148 cm
Highest object	20 cm

Table 3: Largest dimensions of items to be painted

2.2.2 Requirements for safety

Due to the fact that this is a prototype and will only be used in experimental purposes, safety will not be a large concern in this thesis. However, it is very important to assess the risks and find countermeasures for these.

The risks which are identified are the following.

1. Electrical devices in a room with water, such as servos and cameras.
2. Flammable particles from the paint
3. Moving parts which can injure operators in the room

The following countermeasures can be applied.

1. Using IP54 capsuling which isolates electrical devices from liquids[3]
2. Using IP54 capsuling which isolates electrical devices from dust/particles[4]
3. Placing a pressure mat to identify people inside the room as well as a sensor to stop the robot if the door is opened.

2.2.2 Economical requirements

After discussing with the company this thesis has been granted a budget of 100 000 SEK which is estimated to be well enough for a prototype with testing purpose. For a final implementation this would probably be higher due to the need of more industrial-grade hardware.

2.3 Choosing and designing a construction

Two concepts were developed based on own experience from projects at Lund University as well as the interview with the operator. Both concepts are based on a Cartesian robot (XYZ gantry) which is a good choice for this application.

2.3.1 Different concepts and ideas

<u>The basic idea</u>

The motion for the robot is based on linear units which allows for linear movement of the spray nozzle along the X, Y and Z axis. The spray nozzle is attached to the Z axis, in a construction which allows the spray nozzle to be tilted around the X and Y-axis. This allows for spray painting from different angles. The spray nozzle will be controlled by the operator with a hand controller with a gyroscope and trigger for the spray paint. A principle sketch of the gantry with table is illustrated in Figure 4.

The operator gets visual feedback from a camera which is visualized in a VR-headset or on a display. There are two ideas behind how the camera should be used and these are described below in subtopics *concept 1* and *concept 2*.

[3] Wikipedia, *IP Code,* retrieved 2018-04-16

[4] Ibid

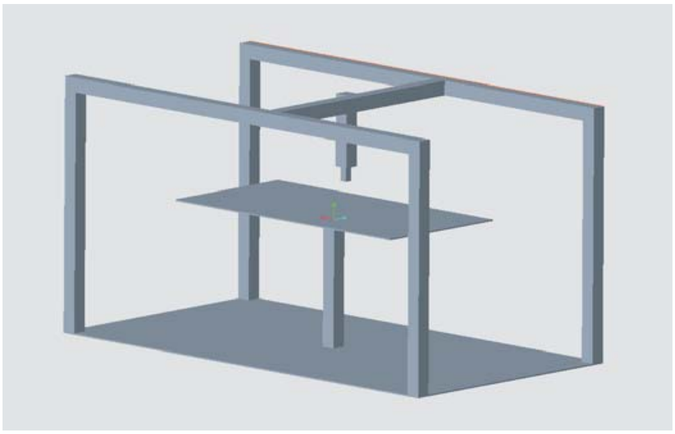


Figure 4: Principle sketch of the idea

<u>Concept 1: Spray nozzle with camera attached next to the nozzle</u>

The first idea is to attach a small, wide-angle camera directly next to the spray nozzle. The operator will see a picture of the table in his VR-headset. By moving the nozzle with the gyroscope, he/she also changes the area which the camera is watching as well as the area which the spray nozzle aims at.

The operator will begin by setting the Z-distance (height) and then calibrating by spraying onto the table. By visual feedback he/she can see where the spray ends up and he/she will mark this area with a touchscreen or similar. If the nozzle sprays a rectangular pattern, he/she can draw a rectangle in the picture which lets him know exactly where the paint will be sprayed.

This idea is inspired from first person video games, where you have a cross-hair which is used to aim the shot.

<u>Concept 2: Spray nozzle with camera separated from nozzle</u>
The second idea is to have the camera separated from the nozzle, attached to a similar construction as the spray nozzle which allows for movement around the X and Y axes. The camera should be placed at some distance over the spray nozzle so that the operator can see both the spray nozzle and the products on the table.

By reading the angles from the gyroscope in the VR-headset, the operator can tilt his head in different angles and the camera attached to the gantry will also tilt with the same angle. This allows for the operator to see the products from different angles, which is a necessity for some of the products.

<u>Chosen concept</u>
After discussions with the company it was decided that concept 1 was the concept to design and implement due to the fact that a movable camera did not add any real value. Concept 2 would also need the VR-headset to be able to display a picture which is not needed in concept 1 since a regular display would be enough.

2.3.3 System chart

The system chart with overview and interconnection of the hardware components is illustrated in Figure 5.

The central hub is the industrial PC which is responsible for the communication between all the different parts of the robot such as the VR-headset, the servos for the gantry, the servos for the nozzle control as well as the camera.

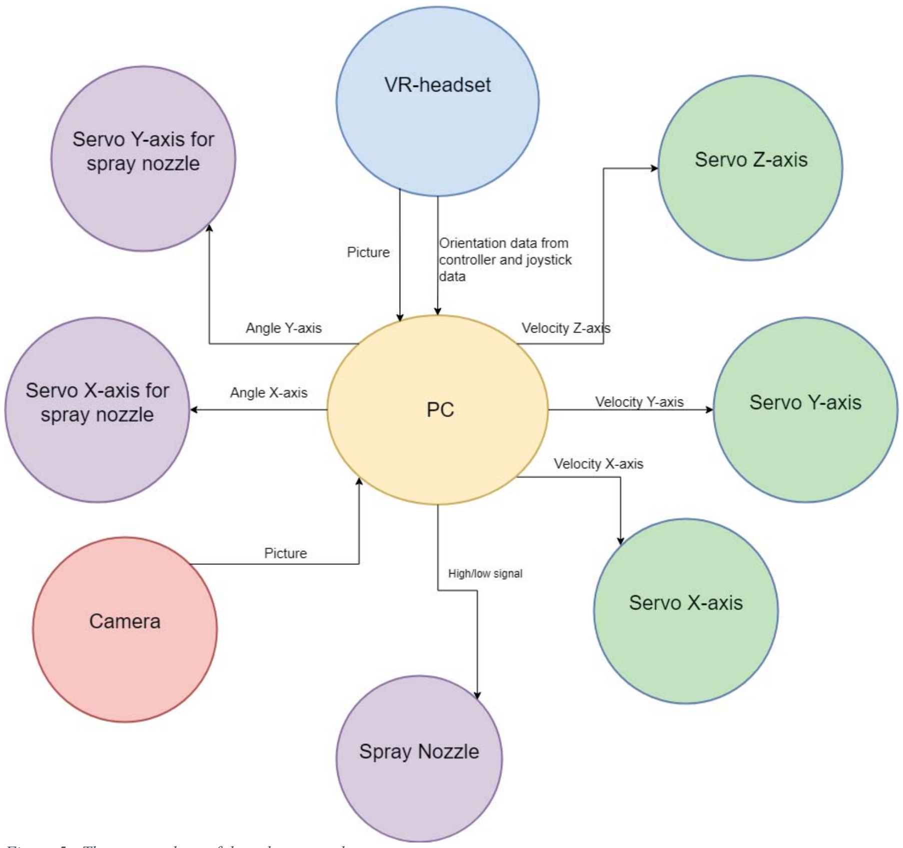


Figure 5: The system chart of the robot control

3 Hardware

Based on the system chart in Figure 5 and the data in Table 1, Table 2 and Table 3 a few essential parts were dimensioned and chosen. The essential parts are illustrated by the headlines in this chapter.

Some less important parts that for this thesis does not require any special dimensioning or specification are not mentioned in this report. This can be items such as roller bearings or a power adapter.

3.1 Gantry

The robot is an X-Y-Z gantry, or in other words a Cartesian coordinate robot.[5] This will allow movement of the spray nozzle in all three dimensions and is also very easy to dimension due to its long range of motion and simple mechanics. The supplier for this X-Y-Z gantry was Rollco AB, a Helsingborg established company specialized in linear motion products.[6]

Rollco AB recommended the use of their linear unit RHL80 after giving them the specifications in Table 1 and Table 2 as well as the estimated weight on the Z-axis which is listed in Table 4. The force from the spray nozzle is estimated to be around 1 kg which in this case is negligible.

Weight from servos and gearboxes	5 kg
Weight from spray nozzle and construction	2 kg

Table 4: Estimated load on Z-axis, maximum values

RHL80 is a modular system which can be dimensioned up to 6000 mm.[7] Based on the dimensions in Table 2 and Table 3 the following dimensions in Table 5 were chosen.

Stroke X-axis	2400 mm
Stroke Y-axis	1500 mm
Stroke Z-axis	400 mm

Table 5: Strokes for the different axes

The weight of the RHL80 modules are listed in table 6.

Weight stroke 0mm	5.2 kg
Weight/100mm stroke	0.8 kg
Weight for the slider	0.8 kg

Table 6: Weight for RHL80 [8]

For dimensioning the size of the servos and gearboxes, weight of the Y-axis and Z-axis gantry modules were calculated according to Equation 1 and is illustrated in Table 7.

$$m = 5.2 + 0.8 \cdot L + 0.8 \qquad (1)$$

[5] M. W. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*, s.16-17, John Wiley & Sons INC, New Jersey

[6] Rollco, *Välkommen till Rollco*, retrieved 2018-03-09

[7] Rollco, *Linear unit RHL,* retrieved 2018-03-09

[8] Ibid

Total weight of RHL80 on Y-axis	17.2 kg
Total weight of RHL80 on Z-axis	10 kg

Table 7: Total weights of Y and X-axis module

3.2 Servos and gearboxes

3.2.1 Gantry servos and gearboxes

For gantry servos the manufacturer Teknic was chosen due to their unique all-in-one solution with an integrated controller, integrated PID tuner and C++ compatibility for speed and position control. For the spray nozzle servos the manufacturer Robotis was chosen due to their design and usability within robot applications. They are small enough with built-in gearbox which is necessary for this construction to be able to be small and as simple as possible.

In order to be able to calculate the needed torque the specification in Table 1 it is necessary to calculate the total weight the servos will need to accelerate as well as the forces acting on the gantry. The weights for the servos on the Y and Z axes are estimated, as well as the weight of the nozzle and nozzle servos. The reason for this is the uncertainty in how the exact construction will look like as well as material choices. Therefore, maximum estimated values have been used. The estimation comes from approximating a worst case scenario and dimensioning the servos according to that specific load.

The first approximation made was to determine the worst case scenario load on the servos aiming the spray nozzle. By placing the center of mass 20 cm away from the pivot point and approximating the spray nozzle weight to 1 kg, the torque on the servo can be calculated with equation 2 assuming the working angle to reach is 90˚.

$$M = F \cdot L \tag{2}$$

The worst case scenario for the spray nozzle servo gives a servo torque of ~2 Nm. Servos which can handle these loads weighs around 0.1 kg each.[9] In this case, this equals to 0.2 kg in total.

After approximating the weight attached to the Z-axis, which basically is the construction holding the spray nozzle servos as well as the actual spray nozzle, the Z-axis servo can then be dimensioned.

To dimension the servos used for the gantry, a tool called “engineering calculator” from Servotak was used.[10] The equations used in this toolbox can be found in Appendix 3.

By inserting the data in Table 7 for the Z-axis values into the toolbox, the result shows a required gearbox output torque of 0.782 Nm and gearbox ratio of 24:1. The initial values (marked with * in table) comes from the servo CPM-SCHP-2310S-ELNB.[11]

[9] Robotis, *Dynamixel X,* retrieved 2018-03-13
[10] Servotak, *Engineering calculator,* retrieved 2018-03-16
[11] Teknic,*ClearPath integrated servo system CPM-SCHP-2310S-ELNB,* retrieved 2018-03-16

With 24:1 gearbox ratio and the rated nominal torque of the servo at 0.3 Nm it gives that the output torque from the gearbox will be 7.2 Nm which is well over the 0.782 Nm required. This servo therefore fulfills the requirements and is chosen as the Z-axis servo.

System inclination	90 degrees
Pulley diameter	57.3 mm
Load mass	1.2 kg + 0.8 kg (the sleigh)
Friction coefficient	0.1 (estimated)
Ramp-up time	0.1 s
Speed	0.5 m/s
Safety factor	1.25
Motor rated speed*	4000 rpm(taken from the specific servo)
Motor inertia*	0.1kg/cm2 (taken from the specific servo)

Table 8: Data used for calculating servo torque required for Z-axis servo

For the Y-axis servo values in Table 9 were used to calculate the required torque. The motor specific values are chosen from the servo CPM-SCHP-3411S-ELNB.[12] The result was 4.758 Nm of torque and a gearbox ratio of 6:1. With the rated nominal torque of 1.1 Nm the torque after the gearbox is 6.6 Nm, which is more than the required torque of 4.758 Nm. This means that this specific motor is well suited.

System inclination	0 degrees
Pulley diameter	57.3 mm
Load mass	12.1 kg
Friction coefficient	0.1 (estimated)
Ramp-up time	0.1 s
Speed	1 m/s
Safety factor	1.25
Motor rated speed*	2030 rpm(taken from the specific servo)
Motor inertia*	0.7 kg/cm^2 (taken from the specific servo)

Table 9: Data used for calculating servo torque required for Y-axis servo

Lastly, for the X-axis servo values in Table 10 were used to calculate the required torque. The servo specific values come from the servo CPM-SCHP-3432P-ELNB which gave the required torque 12.1 Nm at the gearbox as well as the ratio 7:1.[13] With a nominal torque of 1.5 Nm this gives the torque a torque of 10.5 Nm at the gearbox. This is lower than the required torque of 12.1 Nm, but the servo has a maximum torque of 4.9 Nm instead of the nominal 1.5 Nm which means that this will not be an issue. In worst case it will lead to a slightly slower ramp-up time of 0.1 s. This servo is therefore chosen.

[12] Teknic,*ClearPath integrated servo system CPM-SCHP-3411S-ELNB,* retrieved 2018-03-16
[13] Teknic,*ClearPath integrated servo system CPM-SCHP-3432P-ELNB,* retrieved 2018-03-16

System inclination	0 degrees
Pulley diameter	57.3 mm
Load mass	30.9 kg
Friction coefficient	0.1 (estimated)
Ramp-up time	0.1 s
Speed	1 m/s
Safety factor	1.25
Motor rated speed*	2330 rpm(taken from the specific servo)
Motor inertia*	2.1 kg/cm^2 (taken from the specific servo)

Table 10: Data used for calculating servo torque required for X-axis servo

The total maximum weights seen by the servos are listed below in Table 11

Weight seen by X-servo	
Weight of Y + Z gantry modules	27.2 kg
Weight of Y-servo	1.6 kg
Weight of Z-servo	0.9 kg
Weight of nozzle and nozzle servos	1.2 kg (approximated)
Total weight seen	30.9 kg
Weight seen by Y-servo	
Weight of Z gantry module	10 kg
Weight of Z-servo	0.9 kg
Weight of nozzle and nozzle servos	1.2 kg (approximated)
Total weight seen	12.1 kg
Weight seen by Z-servo	
Weight of nozzle and nozzle servos	1.2 kg (approximated)
Total weight seen	1.2 kg

Table 11: Total weights seen by the servos in all axes.

The servos chosen for the gantry are all Teknic ClearPath brushless DC servos with built in controller and encoder. They can be controlled through C++ with a SDK provided by Teknic and the PID can be tuned for the specific mechanical system with a separate USB cable and program on the PC. They draw power from a separate power supply, IPC-5, and this is also supplied by Teknic.[14]

The communication between the servos and the computer is done by USB to a Teknic SC-hub which controls the servos with Teknics own communication protocol over RS485.

Clearpath servos also gives the possibility to define a maximum torque and acceleration. The SC-hub has an input pin for emergency stop which stops all the servos connected to the hub.[15]

Another function of the Teknic ClearPath servos is the patented G-stop anti-vibration technology. This solution is based on different motion profiles within the controller which dampens down resonances and vibration caused by the mechanics.[16]

[14] Teknic, *How does ClearPath compare?,* retrieved 2018-03-16
[15] Teknic, *ClearPath-SC,* retrieved 2018-03-16
[16] Teknic, *ClearPath-SC,* retrieved 2018-03-16

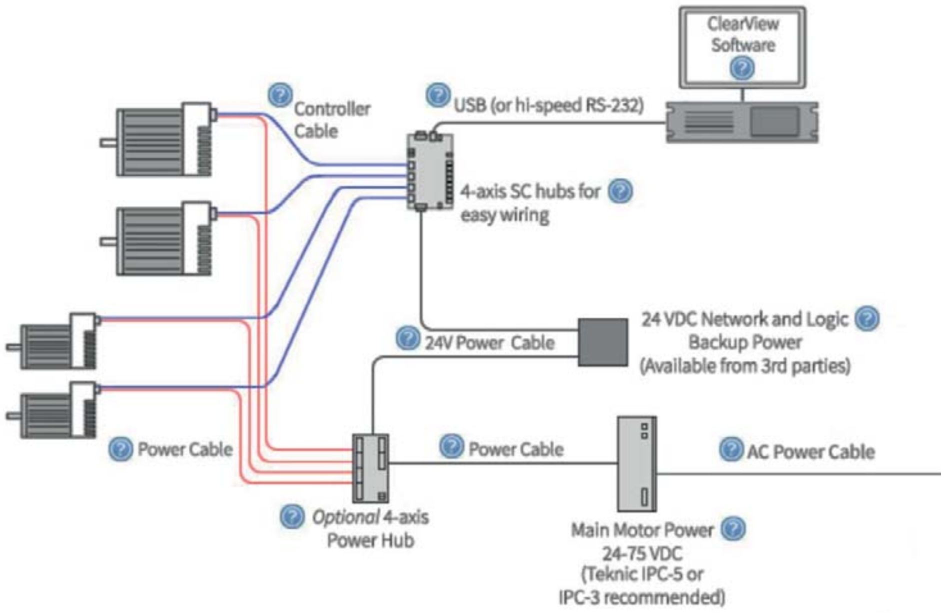

ClearView Software
USB (or hi-speed RS-232)
Controller Cable
4-axis SC hubs for easy wiring
24 VDC Network and Logic Backup Power (Available from 3rd parties)
24V Power Cable
Power Cable
Power Cable
AC Power Cable
Optional 4-axis Power Hub
Main Motor Power 24-75 VDC (Teknic IPC-5 or IPC-3 recommended)

Figure 6: Generic figure of the connections for Teknic servos.
Source: Teknic, retrieved 2018-03-16 [Courtesy Teknic]

3.2.2 Spray nozzle servos

For the spray nozzle servos, Robotis Dynamixel X series servos were chosen due to their small size and good precision but mainly due to Robotis SDK which allows the servos to be controlled in C++ with a simple USB-to-serial device developed by Robotis. The servos range between 57 g and 157 g. For estimating the torque needed the midrange servo, XH430-W350-R was chosen with its 82 g. The brackets for these are in aluminum and their weight is assumed to be 25 g.[17]

In order to choose the right servo for the aiming mechanism the required torque needs to be calculated. The torque needed depends on three factors, the distance from the first servo to the center of mass, the weight and the angle. The construction can be seen in Figure 9.

As specified in Table 1, the working angle is ±45°. The center of gravity with respect to the first servo is obtained in CAD. The mass is estimated to be around 1.5 kg and the center of gravity lies 110 mm from the servo gear according to CAD and the estimated weights.

With equation 3 the torque needed can then be obtained.

$$M = m \cdot g \cdot sin\, 45 \cdot 0.11 = 1.15 \text{ Nm} \tag{3}$$

[17] Robotis, *Dynamixel X,* retrieved 2018-03-13

With 12V power, the XH430-W350-R stall torque is 3.4 Nm, which gives us a safety factor of around three. This servo is therefore a good choice and is chosen for the final construction.

The Dynamixel X series allows for daisy chaining the servos which means that only one servo is connected to the USB-to-serial device as well as the power supply.
With a resolution of 0.088 degrees the precision will be approximately 0.6 mm at 400 mm working distance according to equation 4.[18]

$$d = tan\ 0.0088 \cdot 400 \text{ mm} \approx 0.6 \text{ mm} \tag{4}$$

The Dynamixel servos are using RS485 serial communication which requires terminating the cables when either extending the cables or increasing the baud rate. Terminating the cable means that two resistors are placed between the transmission and receiving cables. It is necessary due to oscillations/reflections that happens in the cable when the signal is passed through the cables, which can cause data corruption. To solve this issue, two terminating resistor on each end of the cable need to be placed. The resistance should match the characteristic impedance of the cable, which is 120 Ω for a twisted pair Ethernet cable.[19]

3.3 VR headset and controllers

The VR headset Oculus Rift is chosen mainly due to its well developed and supported SDK (software development kit). The Oculus Rift comes with two hand controllers which have integrated sensors to keep track of their orientation. The headset itself has a range of sensors used to keep track of the positioning of the head which is essential for the VR to work in this application. The SDK provides easy handling of grabbing and using this data.[20]

The hand controllers have multiple buttons, one joystick each and stepless triggers which can be used for the operator to adjust the flow of paint. [21] The buttons are programmed for calibrating, dead man’s switch and adjusting the speed while the joysticks are used for positioning the spray nozzle over the table.

[18] Robotis, *Dynamixel X,* retrieved 2018-03-13
[19] Maxim integrated, *Guidelines for proper wiring of an RS-485 (TIA/EIA-485-A) Network,* retrieved 2018-04-20
[20] Oculus, *Why Oculus?,* retrieved 2018-03-18
[21] Oculus, *Oculus rift,* retrieved 2018-03-18

Figure 7: The oculus rift VR headset
Source: Wikipedia, retrieved 2018-09-26

Figure 8: The oculus rift touch controllers
Source: Wikipedia, retrieved 2018-09-26

3.4 Spray nozzle

The spray nozzle was not specified since it depends a lot on which spray picture the company wants. Metso uses *Spraying Systems CO* as a supplier for spray nozzles. They recommended the use of their AAB10000JAU nozzle, which is controlled by a control signal between 0 and 24 Volts, where 24 Volts opens the nozzle. The nozzle is either closed or open. The exact setup was not investigated any further since the company made the decision to wait with buying the spray nozzle until the robot was tested due to the high price of the nozzle.

3.5 Spray nozzle construction

One of the most important factors according to the operator Robin Becker is the spray nozzle's ability to replicate the hand movement of the operator for a natural operating of the spray painting according to the interview in Appendix 1.

To be able to replicate the movement the spray nozzle needs to have two degrees of freedom, one rotation around the X axis and one rotation around the Y axis. No rotation around the Z axis is necessary according to Robin Becker, but can be added in the future if the need arises.

The final construction is illustrated in Figure 9 and as seen in the picture the blue parts are the two Dynamixel X430 servos with two FR-12 brackets from Robotis. The brackets are attached perpendicular to each other to be able to be rotated around both the X and Y-axis.

The FR-12 brackets are made out of aluminum while the bracket holding the X-servo and the bracket holding the spray nozzle is made out of polyamide plastic which has been laser cut and glued together.[22]

A simple FEM analysis of the spray nozzle construction in solid polyamide showed a displacement of 0.2 mm when a force corresponding to 1 kg is applied at 45° angle to simulate the pressure force from the spray nozzle. This small displacement is negligible and will not affect the spray painting. The real displacement will most likely be larger than 0.2 mm due to backlash in the servo components.

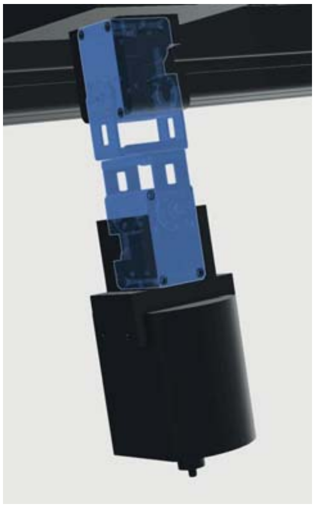


Figure 9: The spray nozzle construction with two servos and brackets

[22] Robotis, *FR-12-H101K Set,* retrieved 2018-03-16

3.6 Camera and lens

In today's market there are a lot of camera and lens manufacturers, as well as many different standards and communication protocols. The typical machine vision camera uses USB, FireWire or GigE communication standard and the lens mounts are typically of type C, CS or S standard.[23]

In order to find a suitable camera and lens, a few assumptions and requirements were made.

1. Distance from the robot to operator placement is at least 20 m.
2. The operating distance from the spray nozzle to table is 200-400 mm.
3. The field of view should be at least 500x350 mm when camera is positioned at 400 mm from the products.
4. The camera needs a frame rate close to the Oculus Rift frame rate of 90 FPS.[24]
5. Good lighting in the room which means no requirement on aperture value for the lens.
6. OpenCV will be used as vision application.
7. A small and lightweight camera.
8. Maximum cost of 700€ in total.

Assumption #1 excludes the use of USB standard cameras due to their maximum length of 5m, or 10 meters with an active USB cable. It also excludes the possibility of FireWire cable due to its maximum length of 4.5 m according to the IEEE-1394 standard. Left is the GigE standard which allows for cable distances up to 100 m.[25, 26, 27]

Assumptions #2 and #3 together with the sensors size of the camera gives the required focal length according to the lens calculator supplied by Flir.[28]

Assumptions #6 and #7 is fulfilled by the Flir Blackfly S models which are very small, around 30x30x30 mm, and has support for converting the image from Flir standard to OpenCV matrix.

One suitable camera in the Blackfly S series is the BFS-PGE-16S2C-CS which has the specification listed in Table 11.[29]

Resolution	1440x1080
Frame rate	78 FPS
Communication	GigE
Price	305€
Sensor size	1/2.9"

Table 11: Specification of the BFS-PGE-16S2C-CS camera

With a fairly low cost, good resolution and high frame rate, this camera was chosen.

[23] Quality Magazine, *How to choose a machine vision camera,* retrieved 2018-03-24
[24] Oculus, *Guidelines for VR performance optimization,* retrieved 2018-03-24
[25] Flir, *Maximum length of FireWire cable,* retrieved 2018-03-25
[26] Flir, *Understanding USB 3.1 and 3.2,* retrieved 2018-03-25
[27] Flir, *GigE Vision,* retrieved 2018-03-25
[28] Flir, *Lens calculator,* retrieved 2018-03-25
[29] Flir, *Blackfly S Color 1.6 MP GigE vision,* retrieved 2018-03-25

In order to choose the right sensor it is necessary to calculate the needed focal length to achieve the right field of view.
As mentioned earlier in this chapter in Assumptions #2 and #3, a field of view of at least 500x350 mm when being 400 mm over the products is required. With a sensor size of 1/2.9” according to Table 11, the Flir lens calculator can be used.[30]

The result of the lens calculator gives a required focal length of 4.16 mm or less. With a CS mount on the camera and a focal length of below 4 mm, the 1.67 mm wide-angle lens from Edmund Optics is a good choice.[31]

This lens features no need for refocusing from 100 mm working distance to infinity. [32] This is essential for this application due to the inability to refocus the camera while operating the robot. According to the Flir lens calculator, this lens will give a field of view of 1240x930 mm, which is well above the requirements.

[30] Flir, *Lens calculator,* retrieved 2018-03-25
[31] Edmund Optics, *1.67mm FL CS-Mount Wide Angle Lens,* retrieved 2018-03-25
[32] Ibid

3.7 To-buy list and economical calculations

Part	Quantity	Total price w/o VAT	Description
Oculus Rift	1	3592 sek	VR-headset with controllers
Teknic CPM-SCHP-3432P	1	5672 sek	X-axis servo motor
Teknic CPM-SCHP-3411S	1	5672 sek	Y-axis servo motor
Teknic CPM-SCHP-2310S	1	4448 sek	Z-axis servo motor
Teknic SC hub	1	308 sek	USB controller hub
Teknic CPM –CTRL-MM660	3	2714 sek	16 m controller cable for gantry servos
Teknic CPM-USB-120-AB	1	80 sek	Usb cable from pc to SC hub
Teknic CPM-PWR-MS120	3	462 sek	Power cable from PSU to servos
Teknic IPC-5	2	3889 sek	Power unit for gantry servos
Wittenstein CP060-MO1-7-111-000	2	5745 sek	Gearbox for X and Y axis
Wittenstein CP060-MO2-25-111-000	1	2768 sek	Gearbox for Z-axis
XYZ Gantry	1	51948 sek	Complete XYZ gantry with accessories
Computer	1	7387 sek	Computer with necessary hardware
Dynamixel X430-W350R	2	3864 sek	Servos for spray nozzle
Robotis FR12-H101k	2	451 sek	Aluminum frame for attaching dynamixel servos
Dynamixel U2D2	1	387 sek	Dynamixel USB to serial communication hub
Robotis 12V PSU	1	251 sek	PSU for the dynamixel servos
SMPS2Dynamixel Power adapter	1	85 sek	Power adapter for dynamixel servos
Flir BFS-PGE-16S2C-CS	1	2562 sek	Machine vision camera
1.67mm wide angle lens	1	3066 sek	Wide angle lens for the machine vision camera
Cat 6 cable 50m	1	400 sek	Cat 6 ethernet cable for extending serial communication and for the machine vision camera
TP-link POE injector	1	239 sek	POE injector for the machine vision camera
Total price excluding VAT		105 990 sek	

Table 12: List of the needed components and the price

As seen in Table 12, the price exceeded the initial budget of 100 000 SEK. This is due to the need of a computer with necessary hardware which initially was thought to be provided by Metso.

3.8 CAD sketches of the finished robot

With pre-made CAD sketches of the chosen hardware, a complete CAD assembly was made in order to get a picture of the final construction before products were ordered and produced.

The CAD model is illustrated in Figure 10 - Figure 12.

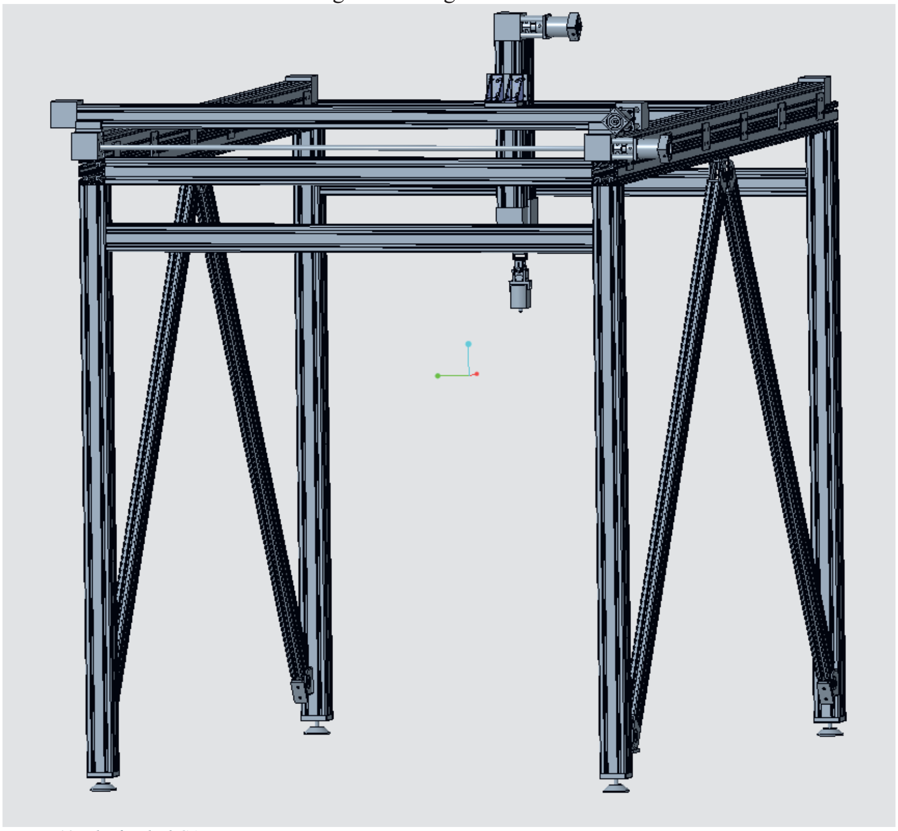


Figure 10: The finished CAD construction

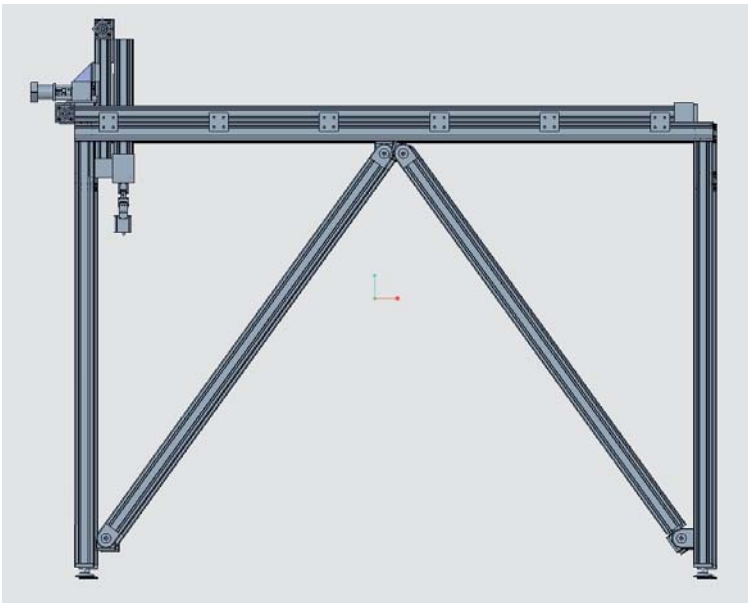


Figure 11: The finished CAD construction

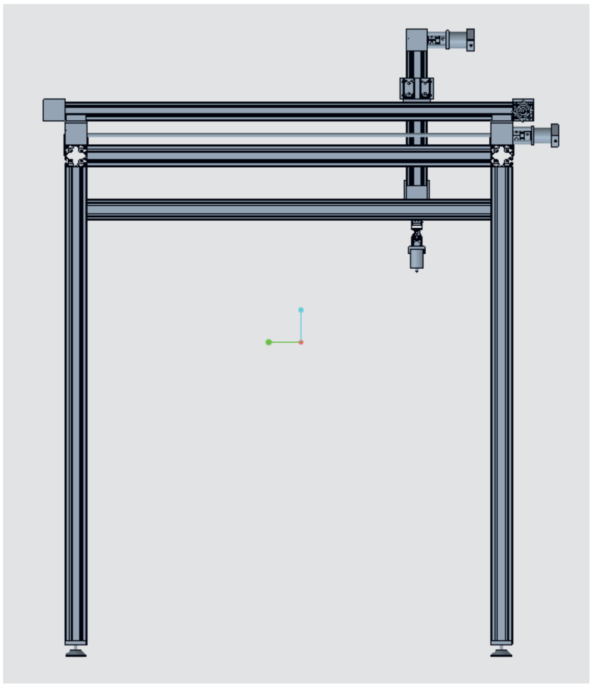


Figure 12: The finished CAD construction

4 Software

This section contains the main segments of the software part of this thesis.

When choosing hardware it was essential that the controlling of the hardware had a SDK supplied with the product.[33] This leads to a lot less development time for the software since a lot of code is prewritten. For example, the Teknic SDK allows the programmer to program for position and velocity control with C++ code while the SDK handles the serial communication between the PC and the servos.[34]

The SDKs used in the application are listed down below and they are all described more thoroughly in *4.1 Programming language and libraries.*

1. Teknic SDK for the gantry servos
2. Dynamixel SDK for the spray nozzle servos
3. OculusSDK for the Oculus VR headset and touch controllers

4.1 Programming language and libraries

Due to the C++ compatibility for all the SDKs supplied with the hardware, C++ is chosen as programming language. C++ is a very popular choice for industrial automation and robotic applications and gives real time performance depending on the operating system.[35] This gives a good foundation for this application and possible extensions.

For this thesis, Microsoft Visual Studio 2017 has been used as an integrated development environment. This IDE includes a lot of standard libraries for C++, such as math libraries for calculating mathematical operations.[36] To be able to access the independent SDKs their libraries, headers and binary files have to be added to the project properties and correct folders.

4.1.1 Teknic SDK

The Teknic SDK is a software development kit which is used to access all the functionalities in the Teknic Clearpath SC models. The SDK includes methods and prewritten code to allow for communication between the computer and the SC controller hub over USB. The SC controller hub then communicates over RS 485 serial communication with the servo motor's internal controller.

The Teknic SDK is not platform dependent but the USB to serial communication drivers are only provided for Windows and Linux, therefore the SDK can be used on any platform but the final application can only be run on Windows or Linux.[37]

In this application the Teknic SDK functions for velocity control were used since it is controlled by joystick inputs from the Oculus touch controller.

[33] Wikipedia, *Software development kit,* retrieved 2018-04-12
[34] Teknic, *ClearPath SC Motors user manual,* retrieved 2018-04-12
[35] Robotiq, *What is the best programming language for robotics?,* retrieved 2018-04-14
[36] Wikipedia, *Microsoft Visual C++,* retrieved 2018-04-14
[37] Teknic, *Downloads,* retrieved 2018-04-12

4.1.2 Dynamixel SDK

In the same manner as Teknics SDK, the Dynamixel SDK lets the Dynamixel servos be controlled with C++ code while the U2D2 adapter handles the RS485 communication between the servos and the computer. The SDK includes all functions and declarations needed to initiate serial communication with the servos as well as setting the registries and commanded position or velocity.[38]

The Dynamixel SDK is platform independent which means it is provided in source code and can be built on any C++ capable platform, such as Windows or Linux.[39]

In this application the servos are used with position control since it is necessary to set a certain position for each servo to replicate the angle of the hand gesture of the Oculus touch controller.

4.2.3 Oculus SDK

The Oculus SDK contains the necessary code for developing own applications with the Oculus Rift VR headset together with the Oculus touch controllers. Oculus SDK supports developing in for example C++ and Unreal Engine.[40] In this application C++ is used. The Oculus SDK is platform dependent and is provided with Visual Studio prebuilt libraries.

For the Oculus SDK to work, the Oculus runtime service has to be installed. This contains the drivers for the Oculus headset which allows for the PC to communicate with the headset over USB. The runtime service has minimum requirements on the PC hardware to be able to operate, and these are listed in Table 13.[41]

Nvidia 970 or better graphics card
Intel i5-4590 or better CPU
8 Gb of RAM or more
HDMI 1.3 output
USB 3.0 for sensors
Windows 7 or newer

Table 13: Minimum requirements of the Oculus Rift headset

The Oculus touch controllers has built in gyroscope, accelerometer and magnetometer. These are used for the SDK to calculate the position and orientation of the touch controller. The orientation is of importance in this thesis to be able to calculate the pitch and yaw of the controller. With a method call, the SDK returns a quaternion with the rotation of the controller.[42] A quaternion is a mathematical notation to represent orientations and rotations in three dimensions.

[38] Github, *Robotis Dynamixel SDK,* retrieved 2018-04-20
[39] Robotis, *Dynamixel SDK,* retrieved 2018-04-20
[40] Oculus, *Oculus rift,* retrieved 2018-03-18
[41] Oculus, *Oculus rift driver setup,* retrieved 2018-03-18
[42] Oculus, *LibOVR 1.17 Reference guide,* retrieved 2018-03-23

They are, compared to Euler angles, simpler to compose and avoid the problem of gimbal lock.[43] With the function getYawPitchRoll() the yaw, pitch and roll are extracted from the quaternion.[44]

4.2.4 Flir SpinView and Spinnaker SDK

Flir SpinView is an application used to control the Flir cameras and also view the camera feed.[45] The Spinnaker SDK also provides this functionality but is used for developing own applications with Flir machine vision cameras.[46]

Both the SpinView and Spinnaker SDK allow for functionalities like changing exposure rate, frame rate, resolution, trigger mode etc. In this application, the exposure rate is set to automatic and the frame rate as well as the resolution are modified in the SpinView application.

[43] Wikipedia, *Quaternions and spatial rotation,* retrieved 2018-04-17
[44] Oculus, *Initialization and sensor enumeration,* retrieved 2018-04-23
[45] Flir, *Getting started with SpinView,* retrieved 2018-04-20
[46] Flir, *Spinnaker SDK,* retrieved 2018-04-20

4.3 Application structure for remote controlled robot

The final program code is found in Appendix 4 but a simplified pseudocode is found in Section *4.3.1 Pseudocode.*

4.3.1 Pseudocode

```
#Include all header files

Initialize Teknic servos
Initialize Dynamixel servos
Initialize Oculus Rift headset

main{
        Declare parameters //Necessary parameters such as maximum speed, acceleration,
        angles as well as used variables in the application

        Create necessary object //Creating objects for the teknic servos, port handlers for serial
        communication

        Calibrate controller  //Calibrating touch controller to set the current orientation to 0.

while(true){

        Get coordinates from Oculus rift touch controller //Grabbing current orientation of the
        Oculus rift controller

        Calculate angle (X,Y) //Calculating which angle the servos should have depending on
        the Touch controller orientation

        Thread set servo angle  (X,Y) //Setting servo angle, threaded to not let the slow
        mechanical operation stall the while loop

        Get coordinates from joystick //Grabbing the current joystick input

        Calculate servo speed(X, Y, Z)

        Thread Set servo speed(speed, servo object) //Setting the speed in the specific servo
        passed to the function, for example 2000 rpm to servoX. Threaded to not let the slow
        mechanical operation stall the while loop

        Checking button input from operator //checking if the operator presses any of the
        buttons

        Do what operator wants //Checking which buttons or triggers are pressed, example
        spray.

        }
        Close ports and deinitialize everything
        return 0; //Quit the program
}
```

4.3.2 Multithreading
Multithreading is the ability for the processor to execute multiple threads on the same processor unit.[47] In this specific application multithreading is essential to not stall any of the operations done in the main thread.

For example, it is essential to be able to fetch pictures from the camera while reading sensor data from the Oculus touch sensors and controlling the servos, all at the same time. Without multithreading everything would be stalled by the slowest process. This would lead to slow responsiveness for the servos and delay in the picture which most likely would cause so much latency that the robot would be impractical.

The multithreading is accomplished by the C++11 standard class std::thread. This class creates an object of the passed function and its parameters and runs it in its own thread.[48]

In software systems there is a phenomenon called race conditions or race hazard which arises when a system depends on a sequence or timing of processes and threads. This can become critical and is therefore called critical race conditions.[49] It occurs most often while processes or threads use the same shared resource, therefore this must be handled by mutual exclusion which prevents threads of accessing the same shared resource and changing its value.[50] In multithreading this must always be taken into account to not let the threads change value of the same resource at the same time since this would lead to wrong data being read.

4.4 Application for autonomous painting
As of request from the company, a simple sequential program was written to autonomously paint the products which cover the entire table. The code for this application is basically the same as in *4.3.1 Pseudocode*, but with an additional section of code which lets the operator start an autonomous painting mode.

The autonomous painting mode lets the operator define a start and end point of the table and the spray angle of the nozzle. The application then calculates the required offset in X and Y axis, moves the robot to the offset position and then starts autonomously painting back and forth over the table with the set angle α, see Figure 13.

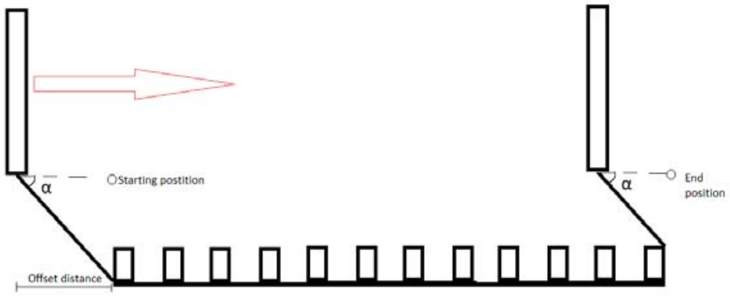


Figure 13: The procedure for autonomous painting

[47] Wikipedia, *Multithreading,* retrieved 2018-04-18
[48] CPPReference, *STD::Thread,* retrieved 2018-04-18
[49] Wikipedia, *Race condition,* retrieved 2018-04-18
[50] Wikipedia, *Mutual exclusion,* retrieved 2018-04-18

The operator can jog the robot to the start and end position with the joystick and then press button A for saving the position. The spray angle can be adjusted with input from the operator. For example, using 0 as spray angle would have 0 offset distance which means the robot sprays from the top and starts exactly at the starting position and stops at the end position.

This application is only suitable for the parts which are closely placed on the table due to the fact that it sprays paint the entire stroke. For parts which do not cover much area of the table, this autonomous painting would lead to a lot of waste of paint.

5 Optimization

One of the most important aspects of the semi-automated spray painting robot is how the robot is behaving and the operator situation awareness while controlling the robot. The robot must be tuned to have the perfect speed and low jerkiness while still feeling responsive and predictable for the operator. The optimization process was therefore an iterative process together with the operator to find the perfect balance. The following topics describe the parameters tuned.

5.1 Optimization of the gantry servos

In Table 1 the maximum required speed was set to 1 m/s after studying the operator while painting. The joystick is continuous which means it can output any value between 0 and the maximum, here 1 m/s. However it was found that the operator had difficulties running at very low speeds due to the small joystick. Therefore, a button on the controller was used to let the operator quickly adjust the max speed to a lower value.

Furthermore, the operator had difficulties running the robot in just one direction, for example if he wanted to spray along a straight line. This was due to that the joystick was difficult to hold exactly at zero in one direction. To resolve this issue, a dead zone was implemented to interpret all inputs below 0.1 m/s as 0 m/s.

Another issue found was the vibrations caused by backlash in the Z-axis linear unit. This caused the entire Z-axis unit to vibrate back and forth for a small period of time when the operator for example went from full speed to a complete stop. To resolve this issue, two things were adjusted.

First, the acceleration and retardation were limited to 4000 rpm/s for the X and Y axes. The acceleration could be lowered even more but was kept at this value due to the responsiveness going down the lower the acceleration is. At very low acceleration, the operator felt the responsiveness being so bad that the robot was completely unpredictable and was not behaving as the operator wanted. Secondly, the patented anti-vibration function G-stop was activated. After testing a few motion profiles, a certain profile was chosen. These two adjustments lowered the vibrations a lot while the robot still remained controllable and predictable.

To ensure that the Z-axis linear unit never bumped in to the gantry frame, software limits were implemented. This made sure that the robot slowed down to a complete stop when closing in to the software limits. The software limits were 0-560 000 encoder counts on the X-axis and 0-330 000 counts on the Y-axis corresponding to a working space of 2200x1350 mm.

5.2 Optimization of the spray nozzle servos

The spray nozzle servos were behaving quite well out of the box and the only optimization done was adjusting the PID parameters and increasing the baud rate.

During the first test run it was discovered that the movement of the nozzle had a delay of approximately 0.3 seconds and the movements were quite jerky. This was measured with a slow motion camera.

This was unwanted and therefore the transmission speed between the computer and the servos was increased to 3 MBps. This cancelled out almost the entire delay and the spray nozzle moved really accurately and quickly.

After this change, the jerkiness was still an issue. To resolve this the PID parameters were tuned, especially the P-part which was lowered from a value of 900 to 200. This eliminated the jerkiness while keeping the responsiveness. The PID parameters were chosen together with the operator, to find the optimal "feeling" according to him.

5.3 Optimization of the camera

Initially when testing the camera, it worked very well with a refresh rate of 68 Hz and measured latency below 200 ms. When the Oculus runtime drivers started communication with the Oculus VR headset, massive packet drops were detected which led to stutter in the picture and an update rate jumping between 10 Hz and 68 Hz. The cause of this is still unknown but probably happens because of the heavy load and massive data transfers between the CPU and GPU. The VR headset is very GPU demanding while the Flir camera transfers a large amount of data between the computer and the camera. When running at full resolution, 1440x1080 pixels, it reaches about 800-900 Mbit per second.

To solve this issue, the resolution had to be lowered to 1000x750 pixels which lowered the transfer rates to 550-650 Mbit per second and the packet losses came to a stop. This probably happens due to the large amount of interrupts while transferring picture streams which stalls the CPU. The Oculus runtime also introduces interrupts and when the camera is running at 900 Mbit per second, the amount of total interrupts cannot be handled by the CPU and it therefore drops packets from the camera.

6 Results

The results in this thesis are based on the experience of how the operator controls and handles the robot as well as whether implemented functions worked or did not work as intended. In the results section the results are described while the optimization section describes the cause of the result as well as the solution.

The idea of using the VR headset as a vision feedback system was not implemented due to difficulties in showing a picture from the camera in the headset. Instead a computer monitor was used with AR-technology implemented.

In Figure 14 - Figure 16, the final construction can be seen. Due to the small space in the room where the prototyping was done, the entire construction could not be fit into one picture.

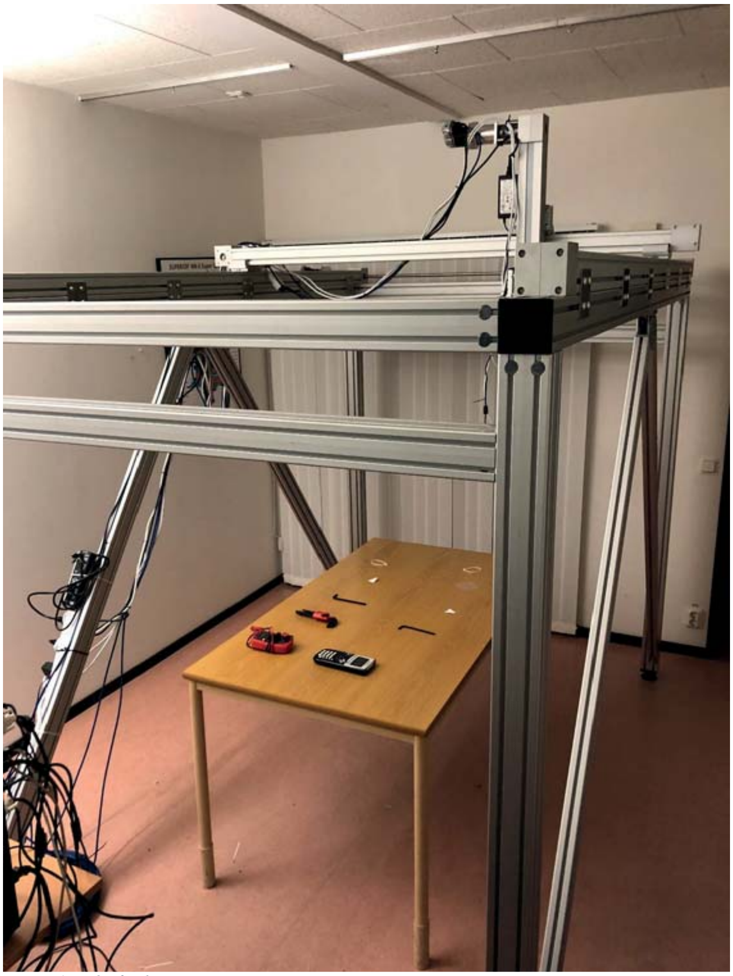


Figure 14: The final construction

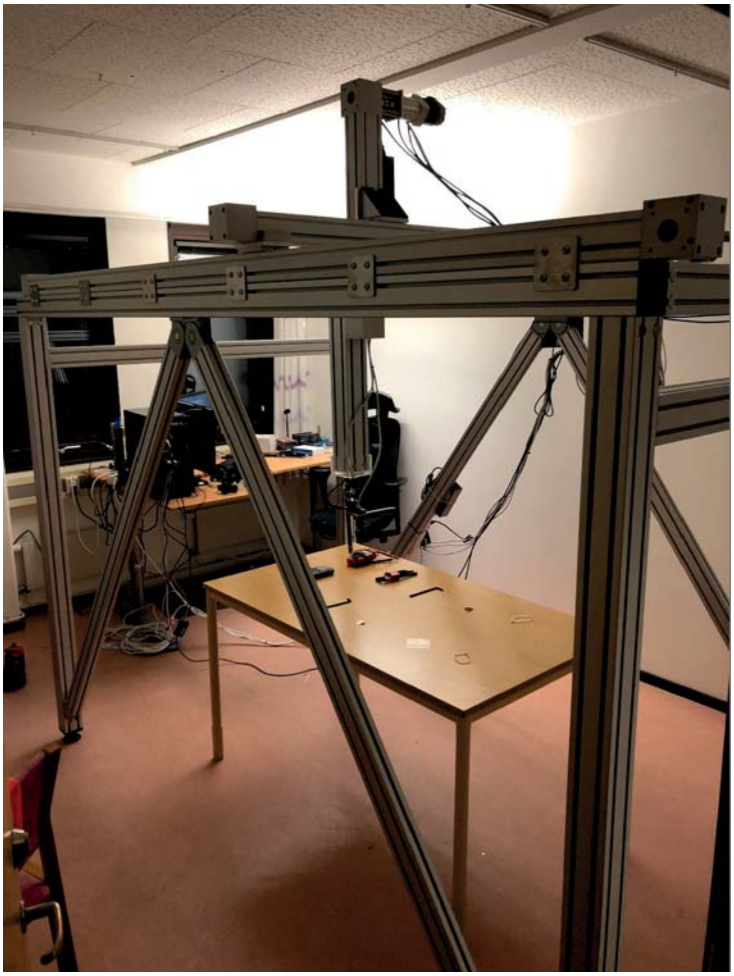


Figure 15: The final construction

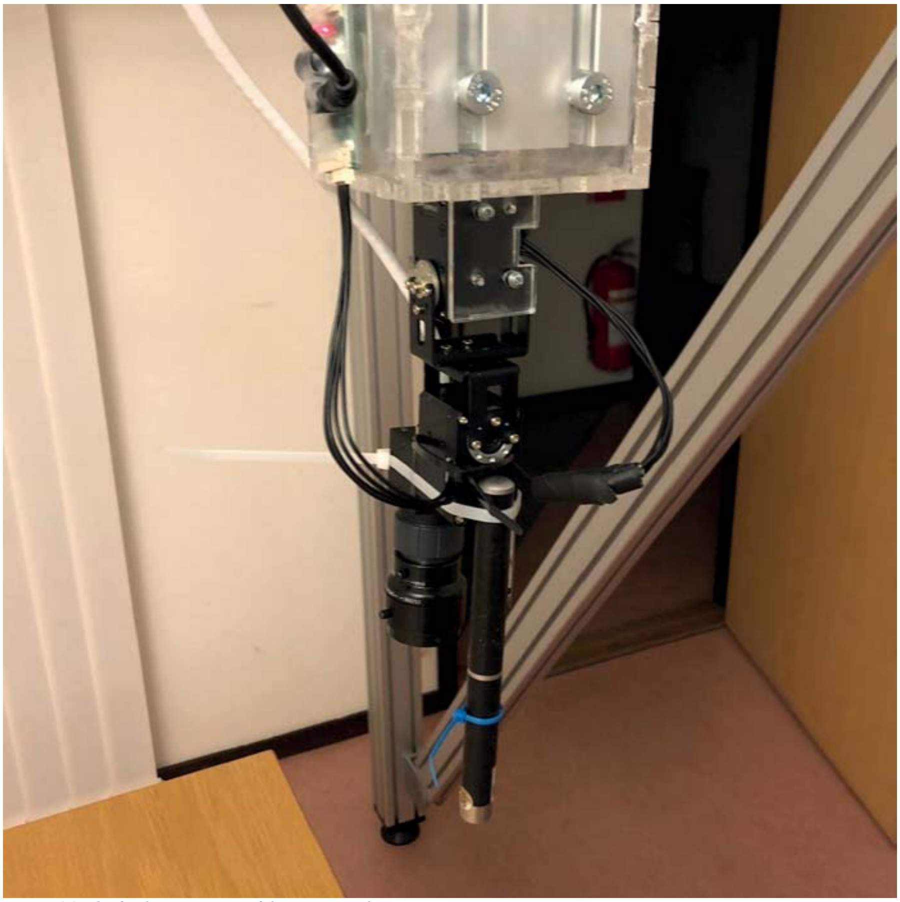


Figure 16: The final construction of the spray nozzle setup

6.1 Pre-optimization results

The results from the initial test run is the basis of this section, before any optimization was done.

The positioning of the spray nozzle could be adjusted with the joystick. The positioning was accurate and predictable, but a bit jerky. The height was adjusted with two buttons named Z and X, on the touch controller. The buttons can be seen in Figure 8. Difficulties running along one axis were found. The maximum speed was found to be optimal but lower speeds were difficult to handle. It was also found that bumping in to the sides of the gantry frame was an issue due to the inability to see the frame in the camera.

The spray nozzle imitated the hand of the operator in a good way and it was accurate. Some delay was noticed which was described as annoying. The nozzle movement was too jerky. When twisting the hand controller (roll), both servos got an input signal and changed the angle which is unwanted.

The visual feedback from the camera was good and the picture covered a larger field of view than needed. Some problems were noticed with packet losses from the camera which caused stutters in the picture from time to time. This only occurred while running the Oculus application together with the camera. The AR solution with a cross hair in the picture to illustrate where the operator is pointing was working well.

6.2 Post-optimization results

In this section the result after the iterative optimization test are described.

The positioning could be adjusted with the joystick and the movement was responsive and predictable and very little jerkiness was detected. It was easy to adjust the speed with the "menu button" on the touch controller as well as the height with the Z and X buttons on the controller. Crashing in to the sides of the gantry was impossible.

As before the optimization, the spray nozzle imitated the hand of the operator in a good way and it was also accurate. There was almost no delay and was therefore very difficult to measure. The nozzle movement was smooth and followed the hand very well. The servos still changed angle when twisting (roll) the touch controller.

Lastly, the visual feedback in the monitor was working well and just as intended. The area covered by the field of view was very good and no stutter or annoying latency was discovered.

6.3 Autonomous painting results

The robot followed the planned trajectory perfectly and calculated the correct distances depending on the spray angle and spray width just as intended. It passed the correct amount of strokes back and forth to cover the entire table. Due to that the robot never was implemented in the real world scenario, it is difficult to assess how the paint actually covered the products and if the painting would be sufficient.

6.4 Videos of the robot

In the following links four videos are uploaded to Youtube to show the basic functionality of the robot in a simulated environment. The videos show that the robot fulfill the specification of requirements, such as responsiveness and speed. However, no error analysis has been made.

Remote Control
https://youtu.be/si3gf4YhhuA
Video of the complete painting process, such as the visual feedback from the camera, how he/she aims with the nozzle and how he/she repositions the robot.

Responsiveness of gantry
https://youtu.be/Mtxdxrp0TLY
Video of the speed and responsiveness when repositioning the robot with the joystick.

Responsiveness of the camera and nozzle servos
https://youtu.be/1hp7jxCmMkk
Video of how speed and responsiveness when aiming the spray nozzle depending on the orientation of the joystick

Automatic program
https://youtu.be/5--IvVYTdVk
Video of the automatic program.

7 Discussion

7.1 Pre-optimization results

Before optimizing and iteratively testing different settings the robot did not behave well enough for using in the real spray painting process. The movements were too jerky and unpredictable which made it too difficult for the operator to use the robot. This was expected before implementing the robot since it is very difficult to actually predict how the robot actually would behave due to a lot of unknown factors.

Another big question mark before implementing the robot was how the visual feedback system would work. Initially the idea was to use the VR headset for displaying the picture which was proven to be a difficult problem to solve due to the complexity in how the VR headset works graphically. Using a screen to show pictures instead worked very well and was approved by the operator according to the interview in Appendix 2, but the camera needed a few optimizations.

7.2 Post-optimization results

After tuning all the parameters such as setting the correct speed, acceleration and PID parameters for the spray nozzle servos, the robot behaved very well and was well accepted from the operator as well as the other engineers at the company. The movement and "feeling" was very good according to the interview with Robin in Appendix 2.

The issue with the touch controller and how it outputs orientation related to the world coordinate system instead of the local coordinate system still needs to be adjusted for optimal functionality. This requires more understanding in how the coordinate systems are defined as well as understanding how to rotate the quaternion correctly.

Unfortunately, the robot was not yet tested in the real world scenario due to decision-taking aspects at the company. The robot was well received by the operator and engineers and the company now has to take a decision whether to try this or not. Testing the robot would lead to downtime in production and other problems to solve such as buying new equipment for the spray nozzle.

7.3 Autonomous painting

The focus on this thesis was to investigate the possibility of having a remote controlled robot which also could do some autonomous painting. Therefore, not much focus was put into making the autonomous part perfect, instead it was developed to show a proof of concept for a semi-autonomous painting robot.

As described in the result section, the autonomous written application behaved well and seemed to paint the entire table based on the spray angle and spray width just as intended. But since the robot is not implemented in the real world scenario, it is difficult to draw any conclusion whether this works as intended or not. Theoretically, it would work.

The autonomous painting can be implemented in many ways, for example, instead of jogging the robot to a start and end position, the camera could be used to identify where objects are and then calculate how to paint these.
The possibilities are endless and it is up to the company to discuss how much time and money shall be put in to development of an autonomous application.

8 Conclusion and further possibilities

Based on the results presented in Chapter 7, the robot is worth testing in the real world scenario to be able to identify other issues that might arise as well as fine tuning other parameters which might show up in the real world. For prototyping and proof of concept the construction worked well and fulfilled the requirements of usability as well as costing less than 100 000 SEK.

If the company wishes to implement this robot, a few things needs to be taken into account. First, all parts in the current construction are not IP classified (ingress protection marking). This means that they are not protected against intrusion of fluids and dust. Since the robot will be placed in a harsh environment, the company would need to look into other linear units which are able to withstand the environment. The spray nozzle construction would also need to be re-developed to be IP classified. The other parts for the robot are IP classified or are not placed inside the spray box and does not need to be changed or redeveloped.

Furthermore, the camera lens would probably need compressed air to be sprayed in front of it to avoid particles to stick to the camera lens.
For the programming part, a GUI with necessary functions would need to be implemented. The current application is controlled in a command prompt which most likely is not very intuitive for the operator. The application needs to be very simple and quick to use and would probably need a HMI panel, next to the loading station, for the operator to make necessary adjustments.

Lastly, for the autonomous painting, there are endless possibilities in developing an autonomous painting process. Exactly how to implement this depends highly on the requirements from the company, such as budget and requirements on waste of paint and other factors. The easiest and cheapest option would probably be to write pre-written programs for the sheet metals and ceramics, and let the operator specify on the HMI panel which type of product he/she is loading the table with. For the other products, such as the frames, the operator could use the remote controlled function and paint the products through the tele-operational functionality.

Implementing this robot would provide a better ergonomic working environment which would lead to reduced stress and strain for the operator. The robot shows capabilities of both remote controlled painting for non-standard products as well as autonomous painting for standard products.

References

[1] Australian government, *Spray painting* https://www.comcare.gov.au/Forms_and_Publications/publications/services/fact_sheets/fact_sheets/spray_painting/spray_painting2, retrieved 2018-04-12
[2] CPPReference, *STD::Thread,* http://en.cppreference.com/w/cpp/thread/thread, retrieved 2018-04-18
[3] Edmund Optics, *1.67mm FL CS-Mount Wide Angle Lens,* https://www.edmundoptics.eu/imaging-lenses/fixed-focal-length-lenses/1.67mm-fl-cs-mount-manual-iris-wide-angle-lens/, retrieved 2018-03-25
[4] Flir, *Maximum length of FireWire cable,* http://flir.custhelp.com/app/answers/detail/a_id/67/~/maximum-length-of-the-firewire-cable%2C-ieee-1394, retrieved 2018-03-25
[5] Flir, *Understanding USB 3.1 and 3.2,* https://www.ptgrey.com/understanding-usb-31, retrieved 2018-03-25
[6] Flir, *Lens calculator,* https://www.ptgrey.com/lens-calculator, retrieved 2018-03-25
[7] Flir, *GigE Vision,* https://www.ptgrey.com/gige-vision-cameras, retrieved 2018-03-25
[8] Flir, *Blackfly S Color 1.6 MP GigE vision, https://www.ptgrey.com/blackfly-s-color-16-mp-gige-vision-sony-imx273,* retrieved 2018-03-25
[9] Flir, *Getting started with SpinView,* https://www.ptgrey.com/support/downloads/10613, retrieved 2018-04-20
[10] Flir, *Spinnaker SDK,* https://www.ptgrey.com/spinnaker-sdk, retrieved 2018-04-20
[11] Github, *Robotis Dynamixel SDK,* https://github.com/ROBOTIS-GIT/DynamixelSDK/wiki, retrieved 2018-04-20
[12] M.W. Spong, S. Hutchinson, M. Vidyasagar, *Robot Modeling and Control*, 2005, New Jersey: John Wiley & Sons Inc
[13] Maxim integrated, *Guidelines for proper wiring of an RS-485(TIA/EIA-485-A) Network,* https://www.maximintegrated.com/en/app-notes/index.mvp/id/763, retrieved 2018-04-20
[14] Oculus, *Guidelines for VR performance optimization,* https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-performance-guidelines/, retrieved 2018-03-24
[15] Oculus, *Initialization and sensor enumeration,* https://developer.oculus.com/documentation/pcsdk/latest/concepts/dg-sensor/, retrieved 2018-04-23
[16] Oculus, *LibOVR 1.17 Reference guide,* https://developer.oculus.com/reference/libovr/1.17/structovr_quatf/, retrieved 2018-03-23
[17] Oculus, *Oculus rift,* https://www.oculus.com/rift/, retrieved 2018-03-18
[18] Oculus, *Oculus rift driver setup,* https://developer.oculus.com/documentation/pcsdk/0.7/concepts/dg-software-setup/, retrieved 2018-03-18
[19] Oculus, *Why Oculus?,* https://developer.oculus.com/, retrieved 2018-03-18
[20] Quality magazine, *How to choose a machine vision camera,* https://www.qualitymag.com/articles/93861-how-to-choose-a-machine-vision-camera, retrieved 2018-03-24

[21] Robotiq, *What is the best programming language for robotics?,*
https://blog.robotiq.com/what-is-the-best-programming-language-for-robotics,
retrieved 2018-04-14
[22] Robotis, *Dynamixel X,* http://www.robotis.us/x-series/, retrieved 2018-03-13
[23] Robotis, *Dynamixel SDK,* http://support.robotis.com/en/software/dynamixelsdk.htm,
retrieved 2018-04-20
[24] Robotis, *FR-12-H101K Set,* http://www.robotis.us/fr12-h101k-set/, retrieved 2018-03-16
[25] Rollco, *Välkommen till Rollco,* http://www.rollco.se/om-oss/valkommen-till-rollco/,
retrieved 2018-03-09
[26] Rollco, *Linear unit RHL,*
http://www.rollco.se/fileadmin/user_upload/Rollco/Brochures/Linear_Unit_RHL.pdf,
retrieved 2018-03-09
[27] Servotak, *Engineering calculator,* https://www.servotak.eu/tools/engineering_calculator,
retrieved 2018-03-16
[28] Teknic, *ClearPath integrated servo system CPM-SCHP-2310S-ELNB,*
https://www.teknic.com/model-info/CPM-SCHP-2310S-ELNB/, retrieved 2018-03-16
[29] Teknic, *ClearPath integrated servo system CPM-SCHP-3411S-ELNB,*
https://www.teknic.com/model-info/CPM-SCHP-3411S-ELNB retrieved 2018-03-16
[30] Teknic, *ClearPath integrated servo system CPM-SCHP-3432P-ELNB,*
https://www.teknic.com/model-info/CPM-SCHP-3432P-ELNB/, retrieved 2018-03-16
[31] Teknic, *How does Teknic ClearPath compare?,*
https://www.teknic.com/products/clearpath-brushless-dc-servo-motors/, retrieved 2018-03-16
[32] Teknic, *ClearPath-SC,* https://www.teknic.com/products/clearpath-brushless-dc-servo-motors/clearpath-sc/, retrieved 2018-03-16
[33] Teknic, *sc_drawing_arch_2.png,*
https://www.teknic.com/images/sc_drawing_arch_2.png, retrieved 2018-03-16
[34] Teknic, *Downloads*, https://www.teknic.com/downloads/*,* retrieved 2018-04-12
[35] Teknic, *ClearPath SC Motors user manual,*
https://www.teknic.com/files/downloads/Clearpath-SC%20User%20Manual.pdf,
retrieved 2018-04-12
[36] Wikipedia, *IP Code,* https://en.wikipedia.org/wiki/IP_Code, retrieved 2018-04-16
[37] Wikipedia, *Metso,* https://en.wikipedia.org/wiki/Metso, retrieved 2018-04-12.
[38] Wikipedia, *Microsoft Visual C++,*
https://en.wikipedia.org/wiki/Microsoft_Visual_C%2B%2B, retrieved 2018-04-14
[39] Wikipedia, *Mutual exclusion,* https://en.wikipedia.org/wiki/Mutual_exclusion,
retrieved 2018-04-18
[40] Wikipedia, *Multithreading,*
https://en.wikipedia.org/wiki/Multithreading_(computer_architecture), retrieved 2018-04-18
[41] Wikipedia, *Quaternions and spatial rotation,*
https://en.wikipedia.org/wiki/Quaternions_and_spatial_rotation, *retrieved* 2018-04-17
[42] Wikipedia, *Race condition,* https://en.wikipedia.org/wiki/Race_condition,
retrieved 2018-04-18
[43]Wikipedia, *Software development kit,*
https://en.wikipedia.org/wiki/Software_development_kit, retrieved 2018-04-12

Appendix

Appendix 1: Interview #1 with Robin Becker, machine operator
Question 1: What is the most important factor when spray painting?
It is necessary to be able to reach from different angles due to be able to paint the products on every side, the paint must be evenly spread and maintenance must be quite easy for low down time. It is very important that the robot follows my wrist movement exactly.

Question 2: If implementing a robot, what do you think will be difficult?
Daily maintenance would probably be quite difficult.

Question 3: After hearing the basic idea, what is your input?
It seems to work in theory. Based on my experience it seems like the robot fulfills the requirements I have. It will most definitely relieve the stress on my body.

Question 4: From which angles and heights do you spray?
It depends a lot on the product, experience taught me to do it with feeling. I approximate the angle to be 45-50 ° at max and 50 cm from the table at max. The speed would be probably 1 m/s.

Question 5: Is it necessary to rotate the spraying nozzle?
I don't believe so. In the current manual process we almost never rotate the spray nozzle.

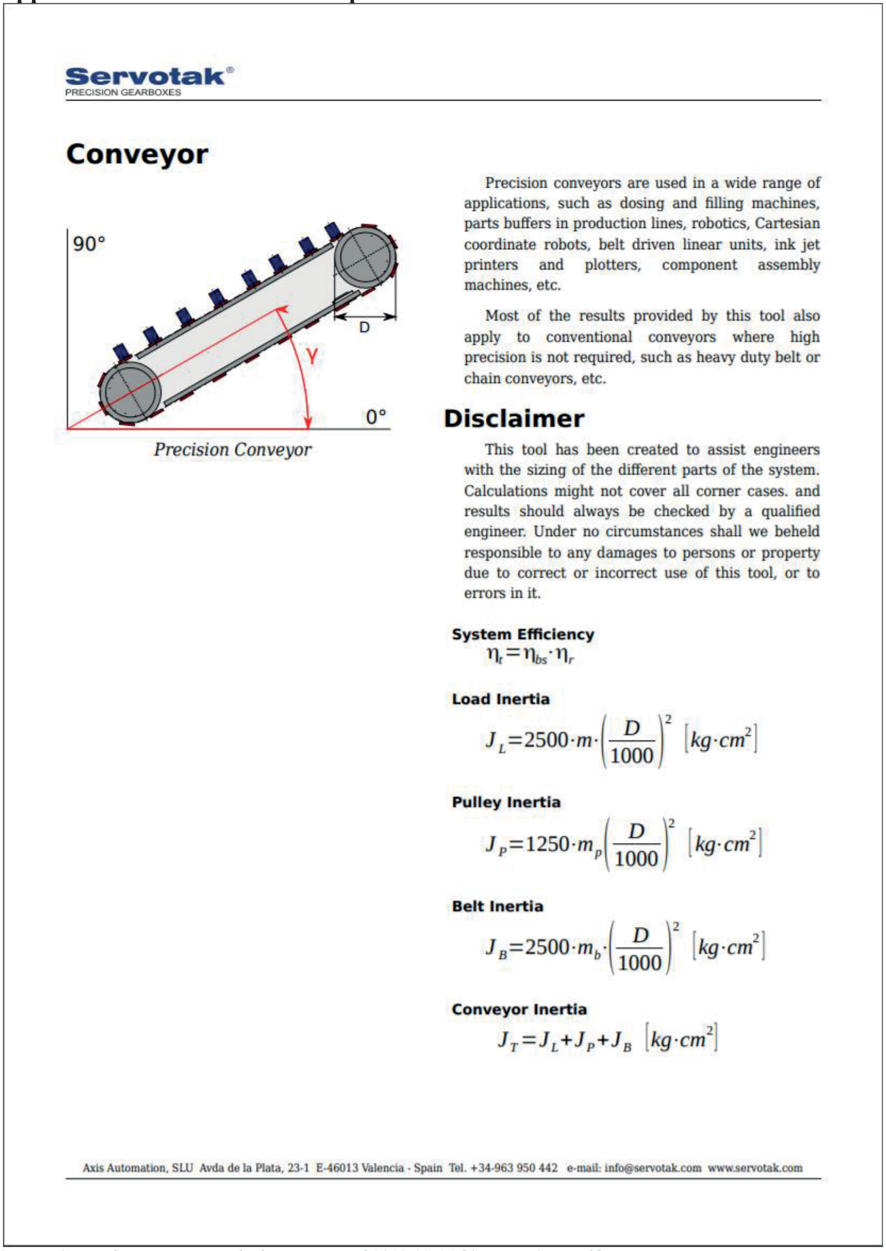
Question 6: Do you feel that the VR headset would lead to any value or would a screen work?
I believe a screen would work fine. I think that a VR headset would lead to dizziness and you would need to have it on your head the entire time.

Appendix 2: Interview #2 with Robin Becker, machine operator
Question 1: How does the robot operate?
It feels well, however it feels a bit jerky and the speed is probably too high. It is a bit difficult to paint in only one direction.

Question 2: How does the robot operate after optimization?
It feels much better and the speed is good and jerkiness is gone. However, I believe there would be a lot of training to be able to use this in a good way.

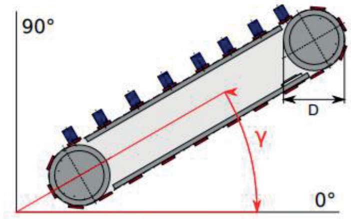
Question 3: Do you see yourself using this robot?
Most definitely yes, it would lead to a lot less stress on my body.

Appendix 3: Servotak toolbox equations



Servotak®
PRECISION GEARBOXES

Conveyor



Precision Conveyor

Precision conveyors are used in a wide range of applications, such as dosing and filling machines, parts buffers in production lines, robotics, Cartesian coordinate robots, belt driven linear units, ink jet printers and plotters, component assembly machines, etc.

Most of the results provided by this tool also apply to conventional conveyors where high precision is not required, such as heavy duty belt or chain conveyors, etc.

Disclaimer

This tool has been created to assist engineers with the sizing of the different parts of the system. Calculations might not cover all corner cases. and results should always be checked by a qualified engineer. Under no circumstances shall we beheld responsible to any damages to persons or property due to correct or incorrect use of this tool, or to errors in it.

System Efficiency

$$\eta_t = \eta_{bs} \cdot \eta_r$$

Load Inertia

$$J_L = 2500 \cdot m \cdot \left(\frac{D}{1000}\right)^2 \quad [kg \cdot cm^2]$$

Pulley Inertia

$$J_P = 1250 \cdot m_p \left(\frac{D}{1000}\right)^2 \quad [kg \cdot cm^2]$$

Belt Inertia

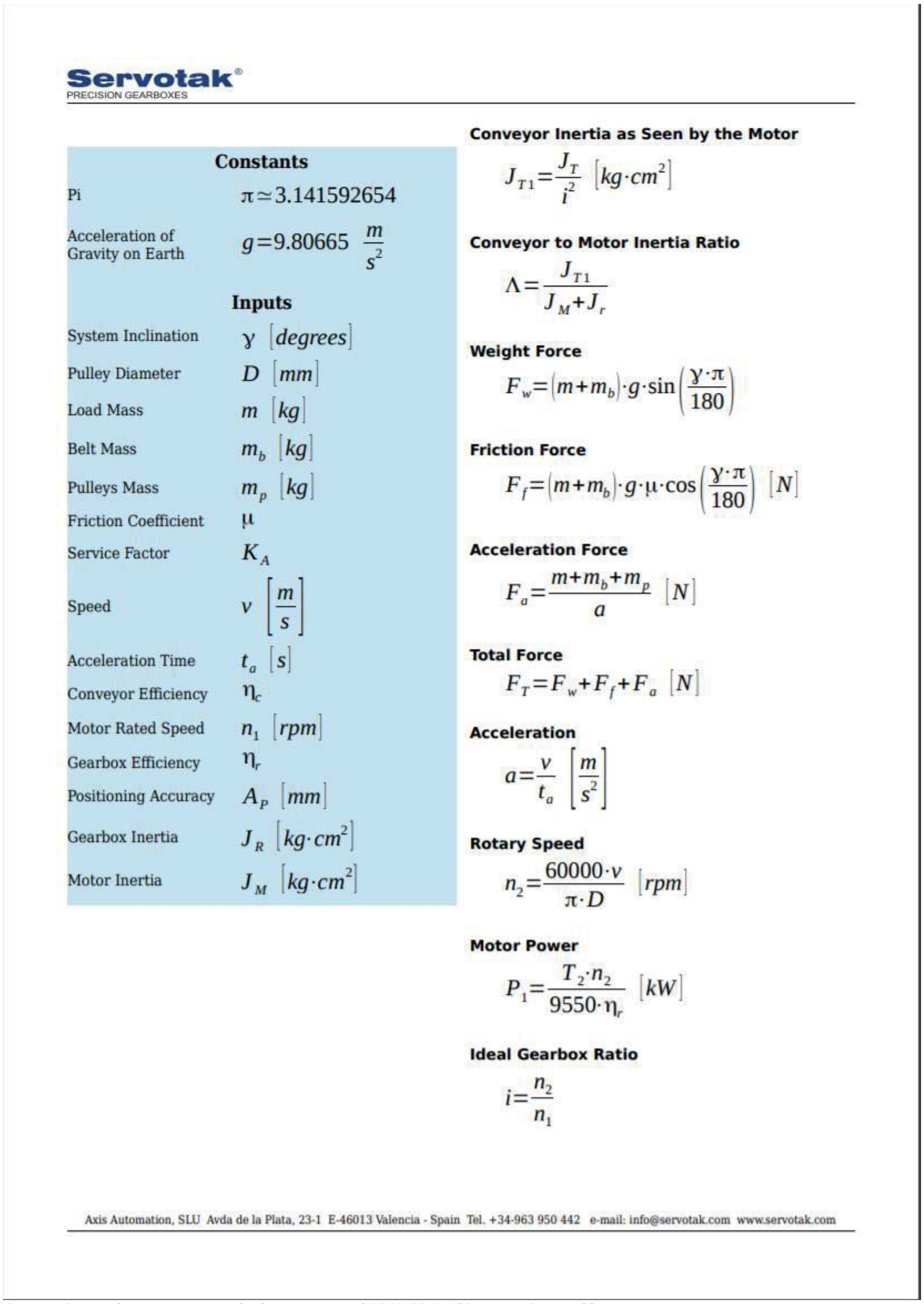
$$J_B = 2500 \cdot m_b \cdot \left(\frac{D}{1000}\right)^2 \quad [kg \cdot cm^2]$$

Conveyor Inertia

$$J_T = J_L + J_P + J_B \quad [kg \cdot cm^2]$$

Axis Automation, SLU Avda de la Plata, 23-1 E-46013 Valencia - Spain Tel. +34-963 950 442 e-mail: info@servotak.com www.servotak.com

Source: Servotak, Engineering calculator, retrieved 2018-03-16 [Courtesy Servotak]



Servotak®
PRECISION GEARBOXES

Constants	
Pi	$\pi \simeq 3.141592654$
Acceleration of Gravity on Earth	$g = 9.80665 \ \frac{m}{s^2}$
Inputs	
System Inclination	$\gamma \ [degrees]$
Pulley Diameter	$D \ [mm]$
Load Mass	$m \ [kg]$
Belt Mass	$m_b \ [kg]$
Pulleys Mass	$m_p \ [kg]$
Friction Coefficient	μ
Service Factor	K_A
Speed	$v \ \left[\frac{m}{s}\right]$
Acceleration Time	$t_a \ [s]$
Conveyor Efficiency	η_c
Motor Rated Speed	$n_1 \ [rpm]$
Gearbox Efficiency	η_r
Positioning Accuracy	$A_P \ [mm]$
Gearbox Inertia	$J_R \ [kg \cdot cm^2]$
Motor Inertia	$J_M \ [kg \cdot cm^2]$

Conveyor Inertia as Seen by the Motor

$$J_{T1} = \frac{J_T}{i^2} \ [kg \cdot cm^2]$$

Conveyor to Motor Inertia Ratio

$$\Lambda = \frac{J_{T1}}{J_M + J_r}$$

Weight Force

$$F_w = (m + m_b) \cdot g \cdot \sin\left(\frac{\gamma \cdot \pi}{180}\right)$$

Friction Force

$$F_f = (m + m_b) \cdot g \cdot \mu \cdot \cos\left(\frac{\gamma \cdot \pi}{180}\right) \ [N]$$

Acceleration Force

$$F_a = \frac{m + m_b + m_p}{a} \ [N]$$

Total Force

$$F_T = F_w + F_f + F_a \ [N]$$

Acceleration

$$a = \frac{v}{t_a} \ \left[\frac{m}{s^2}\right]$$

Rotary Speed

$$n_2 = \frac{60000 \cdot v}{\pi \cdot D} \ [rpm]$$

Motor Power

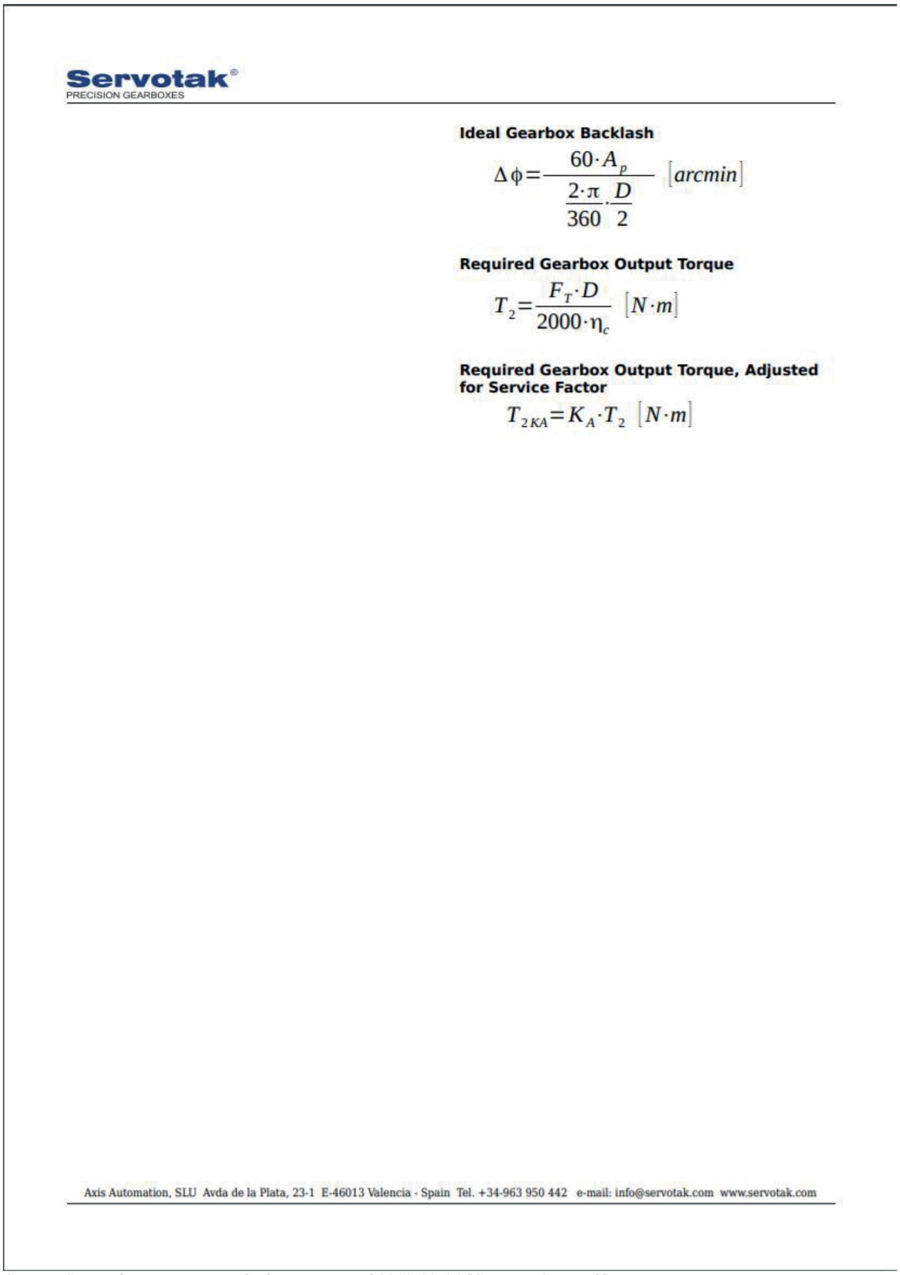
$$P_1 = \frac{T_2 \cdot n_2}{9550 \cdot \eta_r} \ [kW]$$

Ideal Gearbox Ratio

$$i = \frac{n_2}{n_1}$$

Axis Automation, SLU Avda de la Plata, 23-1 E-46013 Valencia - Spain Tel. +34-963 950 442 e-mail: info@servotak.com www.servotak.com

Source: Servotak, Engineering calculator, retrieved 2018-03-16 [Courtesy Servotak]



Servotak®
PRECISION GEARBOXES

Ideal Gearbox Backlash

$$\Delta\phi = \frac{60 \cdot A_p}{\frac{2\cdot\pi}{360}\cdot\frac{D}{2}} \quad [arcmin]$$

Required Gearbox Output Torque

$$T_2 = \frac{F_T \cdot D}{2000 \cdot \eta_c} \quad [N \cdot m]$$

Required Gearbox Output Torque, Adjusted for Service Factor

$$T_{2KA} = K_A \cdot T_2 \quad [N \cdot m]$$

Axis Automation, SLU Avda de la Plata, 23-1 E-46013 Valencia - Spain Tel. +34-963 950 442 e-mail: info@servotak.com www.servotak.com

Source: Servotak, Engineering calculator, retrieved 2018-03-16 [Courtesy Servotak]

Appendix 4: Programming code

```
//Required include files
#include <stdio.h>
#include <string>
#include <iostream>
#include "pubSysCls.h"
#include <iomanip>
#include <OVR_CAPI.h>
#include <Extras\OVR_Math.h>
#include <thread>
#include <windows.h>
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>

#include "dynamixel_sdk.h"

//Define Teknic Servo settings
#define ACC_LIM_RPM_PER_SEC 1500
#define VEL_LIM_RPM         700
#define NUM_MOVES           5
#define TIME_TILL_TIMEOUT   10000

//Define Robotis servos settings
// Control table address
#define ADDR_MX_TORQUE_ENABLE           64
#define ADDR_MX_GOAL_POSITION           116
#define ADDR_MX_PRESENT_POSITION        132
#define PROTOCOL_VERSION                2.0
#define DXL_ID0                         0
#define DXL_ID1                         1
#define BAUDRATE                        3000000
#define DEVICENAME                      "COM5"  //COM port used by U2D2 usb
to serial converter
#define TORQUE_ENABLE                   1  //Value for enabling the torque
#define TORQUE_DISABLE                  0  //Value for disabling the torque
#define DXL_MOVING_STATUS_THRESHOLD     10
#define DXL_P                           300
#define
DXL_I                           0
#define DXL_D                           0
#define ESC_ASCII_VALUE                 0x1b


//Writing one byte data to Dynamixel servo to specific adress
void writeToDynamixelOneByte(dynamixel::PortHandler &portHandler,
dynamixel::PacketHandler &packetHandler, int adress, int value, int id) {

    int dxl_comm_result = COMM_TX_FAIL;
    dxl_comm_result = packetHandler.write1ByteTxOnly(&portHandler, id,
adress, value);
    if (dxl_comm_result == COMM_SUCCESS) {
        printf("Data is set to: %d \n", value);
    }
}

//Writing two byte data to Dynamixel servo to specific adress
void writeToDynamixelTwoByte(dynamixel::PortHandler &portHandler,
dynamixel::PacketHandler &packetHandler, int adress, int value, int id) {

    int dxl_comm_result = COMM_TX_FAIL;
    dxl_comm_result = packetHandler.write2ByteTxOnly(&portHandler, id,
adress, value);
    if (dxl_comm_result == COMM_SUCCESS) {
        printf("Data is set to: %d \n", value);
    }
}

//Writing four byte data to Dynamixel servo to specific adress
void writeToDynamixelFourByte(dynamixel::PortHandler &portHandler,
dynamixel::PacketHandler &packetHandler, int adress, int value, int id) {

    int dxl_comm_result = COMM_TX_FAIL;
    dxl_comm_result = packetHandler.write4ByteTxOnly(&portHandler, id,
adress, value);
    if (dxl_comm_result == COMM_SUCCESS) {
        printf("Data is set to: %d \n", value);
    }
}

//Seting an angle to the dynamixel servos, ID is the # of servo controlled.
void setAngle(double angle, dynamixel::PortHandler &portHandler,
dynamixel::PacketHandler &packetHandler, int id) {
    int dxl_comm_result = COMM_TX_FAIL;
    uint8_t dxl_error = 0;
    double angle2 = 0;

    //Checks which servo to set angle depending on ID that is passed
    if (id == 1) {
        dxl_comm_result = packetHandler.write4ByteTxRx(&portHandler, id,
ADDR_MX_GOAL_POSITION, angle, &dxl_error);
    }
    else {
        angle2 = -1.37226*angle + 4858;
        dxl_comm_result = packetHandler.write4ByteTxRx(&portHandler, id,
ADDR_MX_GOAL_POSITION, angle2, &dxl_error);
    }

    //Checks if write to register was succesfull, otherwise printing the
errorcode
    if (dxl_comm_result != COMM_SUCCESS)
    {
        printf("%s\n", packetHandler.getTxRxResult(dxl_comm_result));
    }
    else if (dxl_error != 0)
    {
        printf("%s\n", packetHandler.getRxPacketError(dxl_error));
    }
}

//Reading an angle from the dynamixel servo
int readAngle(dynamixel::PortHandler &portHandler, dynamixel::PacketHandler
&packetHandler, int id) {
    int dxl_comm_result = 0;
    uint8_t dxl_error = 0;
    uint32_t currentPosition = 0;
    dxl_comm_result = packetHandler.read4ByteTxRx(&portHandler, id,
ADDR_MX_PRESENT_POSITION, &currentPosition, &dxl_error);

    //Printing error if error occured
    if (dxl_error != 0)
    {
        printf("%s\n", packetHandler.getRxPacketError(dxl_error));
    }
    return currentPosition;
}

//Initializing dynamixel servo communication
int initDynamixel(dynamixel::PortHandler *portHandler,
dynamixel::PacketHandler *packetHandler) {

    int dxl_comm_result = COMM_TX_FAIL;             // Communication result
    uint8_t dxl_error = 0;                          // Dynamixel error

    //Opening COM port
    if (portHandler->openPort())
    {
        printf("Succeeded to open the port!\n");
    }
    else
    {
        printf("Failed to open the port!\n");
        printf("Press any key to terminate...\n");
        _getch();
        return 0;
    }

    //Set port baudrate
    if (portHandler->setBaudRate(BAUDRATE))
    {
        printf("Succeeded to change the baudrate!\n");
    }
    else
    {
        printf("Failed to change the baudrate!\n");
        printf("Press any key to terminate...\n");
        _getch();
        return 0;
    }

    //Enable Dynamixel Torque on servo 0
    dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID0,
ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
    if (dxl_comm_result != COMM_SUCCESS)
    {
        printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
    }
    else if (dxl_error != 0)
    {
        printf("%s\n", packetHandler->getRxPacketError(dxl_error));
    }
    else
    {
        printf("Dynamixel 0 has been successfully connected \n");
    }
    //Enable Dynamixel Torque on servo 1
    dxl_comm_result = packetHandler->write1ByteTxRx(portHandler, DXL_ID1,
ADDR_MX_TORQUE_ENABLE, TORQUE_ENABLE, &dxl_error);
    if (dxl_comm_result != COMM_SUCCESS)
    {
        printf("%s\n", packetHandler->getTxRxResult(dxl_comm_result));
    }
    else if (dxl_error != 0)
    {
        printf("%s\n", packetHandler->getRxPacketError(dxl_error));
    }
    else
    {
        printf("Dynamixel 1 has been successfully connected \n");
    }
}
//Calculating the value to send as angle to Dynamixel servos, using min and
max values from
//calibration process to calculate the steepnes of the linear function used
to calculate the servo input
double calcAngle(float& min, float& max, float& calibrated, int id, int
inverse) {


    float m = 2048;

    float dY = 0;
    float dX = 0;
    float k = 0;

    if (id == 0) {
        dY = 3100 - 1100;
        dX = inverse * (max - min);
        k = (dY / dX);
    }
    if (id == 1) {
        dY = 3050 - 1100;
        dX = inverse * (max - min);
        k = (dY / dX);
    }
    float y = k * calibrated + m;
    return static_cast<double>(y);
}

//Calibration method. Calculates the minimum and maximum values for X and Y
axis to be able to know which
//intervals the user wish to operate in.
std::vector<float> calibrate(ovrTrackingState& ts, ovrSession& session,
ovrInputState& inputState, OVR::Quatf handPoseOrient0) {
    printf("Calibration process activated\n");

    int state = 0;
    float minX = 0;
    float minY = 0;
    float maxX = 0;
    float maxY = 0;
    float minXX = 0;
    float minYY = 0;
    float maxXX = 0;
    float maxYY = 0;
    float a;
    float b;
    float c;

    ts = ovr_GetTrackingState(session, 0, true);
    ovrPosef rightHandOrientation = ts.HandPoses[ovrHand_Right].ThePose;
    OVR::Quatf handPoseOrient = rightHandOrientation.Orientation;
    OVR::Quatf calibratedHandposeOrient;
    Sleep(300);
    printf("Go to maximum up and press B when you are there \n");
//Running while loop until user presses B, then grabs the current
orientation and saves the value
    while (state == 0) {
        ts = ovr_GetTrackingState(session, 0, true);
        rightHandOrientation = ts.HandPoses[ovrHand_Right].ThePose;
        handPoseOrient = rightHandOrientation.Orientation;
        calibratedHandposeOrient = handPoseOrient * handPoseOrient0;
        calibratedHandposeOrient.GetYawPitchRoll(&a, &b, &c);
        ovr_GetInputState(session, ovrControllerType_Touch, &inputState);
        if (inputState.Buttons & ovrButton_B) {
            minXX = b;
            state = 1;
        }
    }

    Sleep(500);
    state = 0;
    printf("Go to maximum down and press B when you are there \n");
//Running while loop until user presses B, then grabs the current
orientation and saves the value
    while (state == 0) {
        ts = ovr_GetTrackingState(session, 0, true);
        rightHandOrientation = ts.HandPoses[ovrHand_Right].ThePose;
        handPoseOrient = rightHandOrientation.Orientation;
        calibratedHandposeOrient = handPoseOrient * handPoseOrient0;
        calibratedHandposeOrient.GetYawPitchRoll(&a, &b, &c);

        ovr_GetInputState(session, ovrControllerType_Touch, &inputState);
        if (inputState.Buttons & ovrButton_B) {
            maxXX = b;
            state = 1;
        }
        //printf("max X %f \n", minX * 100);
    }
    Sleep(500);
    state = 0;
    printf("Go to maximum left and press B when you are there \n");
//Running while loop until user presses B, then grabs the current
orientation and saves the value
    while (state == 0) {
        ts = ovr_GetTrackingState(session, 0, true);
        ovrPosef rightHandOrientation =
ts.HandPoses[ovrHand_Right].ThePose;
        handPoseOrient = rightHandOrientation.Orientation;
        calibratedHandposeOrient = handPoseOrient * handPoseOrient0;
        //printf("Calibrated handposeorient y %f \n", handPoseOrient.y);
        calibratedHandposeOrient.GetYawPitchRoll(&a, &b, &c);

        ovr_GetInputState(session, ovrControllerType_Touch, &inputState);
        if (inputState.Buttons & ovrButton_B) {
            minYY = a;
            state = 1;
        }
    }

    Sleep(500);
    state = 0;
    printf("Go to maximum right and press B when you are there \n");
//Running while loop until user presses B, then grabs the current
orientation and saves the value
    while (state == 0) {
        ts = ovr_GetTrackingState(session, 0, true);
        ovrPosef rightHandOrientation =
ts.HandPoses[ovrHand_Right].ThePose;
        handPoseOrient = rightHandOrientation.Orientation;
        calibratedHandposeOrient = handPoseOrient * handPoseOrient0;

        calibratedHandposeOrient.GetYawPitchRoll(&a, &b, &c);

        ovr_GetInputState(session, ovrControllerType_Touch, &inputState);
        if (inputState.Buttons & ovrButton_B) {
            maxYY = a;
            state = 1;
        }
    }
    Sleep(500);
    //Checking maximum and minimum values
    minX = min(minXX, maxXX);
    maxX = max(maxXX, minXX);
    minY = min(minYY, maxYY);
    maxY = max(maxYY, minYY);
    //Prints out maximum and minimum values
    printf("Minimum X = %f \n", minX);
    printf("Maximum X = %f \n", maxX);
    printf("Minimum Y = %f \n", minY);
    printf("Maximum Y = %f \n", maxY);
    float movableDistanceX = abs(maxX - minX);
    float movableDistanceY = abs(maxY - minY);
    printf(" X distance = %f \n", movableDistanceX);
    printf(" Y distance= %f \n", movableDistanceY);
    printf("Calibration complete");
    //Returns vector with maximum and minimum values for further
calculations.
    return { minX, maxX, minY, maxY };
}

//Setting speed of teknic gantry servos
int setSpeed(double speed, sFnd::INode &servo, double previousSpeed) {
    servo.Status.RT.Refresh(); //Refreshing real-time register

    if (servo.Status.RT.operator mnStatusReg().cpm.MoveBufAvail == 1 &&
previousSpeed != speed) {    //Checking if RT register "moveBufAvail" is
true,
        try {
//otherwise does not send commanded speed. This happends
            servo.Motion.MoveVelStart(speed);
//when lots of values are sent and servo has not finished
        }
// the other speed goals
        //Checking errors and printing if there is any
        catch (sFnd::mnErr theErr) {
            printf("Movement failed \n");

            //This statement will print the address of the error, the error
code (defined by the mnErr class),
            //as well as the corresponding error message.
            printf("Caught error: addr=%d, err=0x%08x\nmsg=%s\n",
theErr.TheAddr, theErr.ErrorCode, theErr.ErrorMsg);
```

```
            ::system("pause"); //pause so the user can see the error
message; waits for user to press a key

            return 0;  //This terminates the main program
        }
    }

}
//Initializing the three teknic servos
int servoInit(sFnd::INode &z, sFnd::INode &y, sFnd::INode &x, sFnd::IPort
&myPort, sFnd::SysManager* myMgr) {
    using namespace sFnd;

    printf(" Port[%d]: state=%d, nodes=%d\n",
        myPort.NetNumber(), myPort.OpenState(), myPort.NodeCount());
//Opening port

    //Init of Servo 0
    z.EnableReq(false);
    myMgr->Delay(200);
    printf("   Node[%d]: type=%d\n", int(0), z.Info.NodeType());
    printf("            userID: %s\n", z.Info.UserID.Value());
    printf("        FW version: %s\n", z.Info.FirmwareVersion.Value());
    printf("          Serial #: %d\n", z.Info.SerialNumber.Value());
    printf("             Model: %s\n", z.Info.Model.Value());

    z.Status.AlertsClear();                    //Clear Alerts on node
    z.Motion.NodeStopClear(); //Clear Nodestops on Node
    z.EnableReq(true);

    double timeout = myMgr->TimeStampMsec() + 3000;

    //This will loop checking on the Real time values of the node's Ready
status
    while (!z.Motion.IsReady()) {
        if (myMgr->TimeStampMsec() > timeout) {
            printf("Error: Timed out waiting for Node \t%i to enable\n",
0);
            return -2;
        }
    }

    //At this point the node is enabled
    printf("Node \t%i enabled\n", 0);
    printf("\n");

    //Init of Servo 1
    y.EnableReq(false);
    myMgr->Delay(200);
    printf("   Node[%d]: type=%d\n", int(1), y.Info.NodeType());
    printf("            userID: %s\n", y.Info.UserID.Value());
    printf("        FW version: %s\n", y.Info.FirmwareVersion.Value());
    printf("          Serial #: %d\n", y.Info.SerialNumber.Value());
    printf("             Model: %s\n", y.Info.Model.Value());

    y.Status.AlertsClear();                    //Clear Alerts on node
    y.Motion.NodeStopClear();                    //Clear Nodestops on Node
    y.EnableReq(true);
    timeout = myMgr->TimeStampMsec() + 3000;
    while (!y.Motion.IsReady()) {
        if (myMgr->TimeStampMsec() > timeout) {
            printf("Error: Timed out waiting for Node \t%i to enable\n",
1);
            return -2;
        }
    }
    //At this point the node is enabled
    printf("Node \t%i enabled\n", 1);
    printf("\n");

    //Init of Servo 2
    x.EnableReq(false);
    myMgr->Delay(200);
    printf("   Node[%d]: type=%d\n", int(2), x.Info.NodeType());
    printf("            userID: %s\n", x.Info.UserID.Value());
    printf("        FW version: %s\n", x.Info.FirmwareVersion.Value());
    printf("          Serial #: %d\n", x.Info.SerialNumber.Value());
    printf("             Model: %s\n", x.Info.Model.Value());

    x.Status.AlertsClear();                     //Clear Alerts on node
    x.Motion.NodeStopClear(); //Clear Nodestops on Node
    x.EnableReq(true);

    timeout = myMgr->TimeStampMsec() + 3000;

    //This will loop checking on the Real time values of the node's Ready
status
    while (!x.Motion.IsReady()) {
        if (myMgr->TimeStampMsec() > timeout) {
            printf("Error: Timed out waiting for Node \t%i to enable\n",
2);
            return -2;
        }
    }

    //At this point the node is enabled
    printf("Node \t%i enabled\n", 2);
    printf("\n");
    printf("\n");
    printf("\n");

    //Setting different settings
    z.AccUnit(INode::RPM_PER_SEC);                   //Set the units for
Acceleration to RPM/SEC
    z.VelUnit(INode::RPM);                           //Set the units for
Velocity to RPM
    z.Motion.AccLimit = ACC_LIM_RPM_PER_SEC;         //Set Acceleration Limit
(RPM/Sec)
    z.Motion.VelLimit = VEL_LIM_RPM;                 //Set Velocity Limit
(RPM)

    y.AccUnit(INode::RPM_PER_SEC);                   //Set the units for
Acceleration to RPM/SEC
    y.VelUnit(INode::RPM);                           //Set the units for
Velocity to RPM
    y.Motion.AccLimit = ACC_LIM_RPM_PER_SEC;         //Set Acceleration Limit
(RPM/Sec)
    y.Motion.VelLimit = VEL_LIM_RPM;                 //Set Velocity Limit
(RPM)
```

```
    x.AccUnit(INode::RPM_PER_SEC);                //Set the units for
Acceleration to RPM/SEC
    x.VelUnit(INode::RPM);                        //Set the units for
Velocity to RPM
    x.Motion.AccLimit = ACC_LIM_RPM_PER_SEC;      //Set Acceleration Limit
(RPM/Sec)
    x.Motion.VelLimit = VEL_LIM_RPM;
}

int main(int argc, char* argv[]) {
    //Declaration of used variables
    using namespace sFnd;
    double zSpeed = 0;
    double ySpeed = 0;
    double zInput = 0;
    double yInput = 0;
    double jX = 0;
    double jY = 0;
    double vX = 0;
    double vY = 0;
    double radius = 0;
    double angle = 0;
    double zPreviousSpeed = 0;
    double yPreviousSpeed = 0;
    double xPreviousSpeed = 0;

    float calibratedX = 0;
    float calibratedY = 0;
    float minX = 0;
    float minY = 0;
    float maxX = 0;
    float maxY = 0;
    float accelerationX = 0;
    float accelerationY = 0;
    float a = 0;
    float b = 0;
    float c = 0;

    int calibrationFlag = 0;
    int zBuffer = 1;
    int yBuffer = 1;
    int portnum = 4;
    int counter = 0;
    int inverse = -1;

    boolean calibrated = false;

    OVR::Quatf handPoseOrient0;
    OVR::Quatf calibratedHandPoseOrient;
    OVR::Quatf handPoseOrient;

    uint32_t currentPositionX = 0;
    uint32_t currentPositionY = 0;

    //Creating teknic SysManager
    SysManager* myMgr = SysManager::Instance();

    //Creating port and packethandler for the dynamixel servos
    dynamixel::PortHandler *portHandler =
dynamixel::PortHandler::getPortHandler(DEVICENAME);
```

```
    dynamixel::PacketHandler *packetHandler =
dynamixel::PacketHandler::getPacketHandler(PROTOCOL_VERSION);

    //Trying to open port to the Teknic SC hub
    printf("Hello World, I am SysManager\n");
    printf("\n I will now open port \t%i \n \n", portnum);
    try {
        myMgr->ComHubPort(0, portnum);
        myMgr->PortsOpen(1);
    }

    catch (mnErr theErr) {
        printf("Port Failed to open, Check to ensure correct Port number
and that ClearView is not using the Port\n");
        system("pause");
        return -1;
    }

    //Assign port
    IPort &myPort = myMgr->Ports(0);
    printf("Sysmanager managed to open the ports\n");

    //assign Nodes/servos
    INode &z = myPort.Nodes(0);
    INode &y = myPort.Nodes(2);
    INode &x = myPort.Nodes(1);

    //Initializing servos
    printf("Teknic servos initializing\n");
    servoInit(z, y, x, myPort, myMgr);
    printf("Teknic servos initialized\n");
    printf("Dynamixel servos initializing\n");
    initDynamixel(portHandler, packetHandler);
    printf("Setting PID parameters \n");
    writeToDynamixelTwoByte(*portHandler, *packetHandler, 84, DXL_P, 0);
//Setting P parameter
    writeToDynamixelTwoByte(*portHandler, *packetHandler, 84, DXL_P, 1);
//Setting P parameter
    printf("Dynamixel servos initialized\n");
    Sleep(100);
    printf("Oculus initializing \n\n\n", 0);
    Sleep(1000);


    //Homing of teknic servos to check the physical limits of the gantry
    printf("Starting homing \n\n", 0);
    if ((z.Motion.Homing.WasHomed() && x.Motion.Homing.WasHomed() &&
y.Motion.Homing.WasHomed()) == false) {

        z.Motion.Homing.Initiate();

        while (z.Motion.Homing.IsHoming()) {
            //wait
        }

        if (z.Motion.Homing.WasHomed()) {
            z.Motion.PosnMeasured.Refresh();      //Refresh our current
measured position
            printf("Z completed homing, current position: \t%8.0f \n",
z.Motion.PosnMeasured.Value());
            printf("Soft limits now active\n");
        }

        printf("Soft Limit 1 = %d \n", (int)z.Limits.SoftLimit1);

        printf("Soft Limit 2 = %d \n\n", (int)z.Limits.SoftLimit2);

        y.Motion.Homing.Initiate();

        while (y.Motion.Homing.IsHoming()) {
            //wait
        }

        if (y.Motion.Homing.WasHomed()) {
            y.Motion.PosnMeasured.Refresh();      //Refresh our current
measured position
            printf("Y completed homing, current position: \t%8.0f \n",
y.Motion.PosnMeasured.Value());
            printf("Soft limits now active\n");
        }

        //Printing soft limits
        printf("Soft Limit Y 1 = %d \n", (int)y.Limits.SoftLimit1);
        printf("Soft Limit Y 2 = %d \n\n", (int)y.Limits.SoftLimit2);

        x.Motion.Homing.Initiate();

        while (x.Motion.Homing.IsHoming()) {
            //wait
        }

        if (x.Motion.Homing.WasHomed()) {
            x.Motion.PosnMeasured.Refresh();      //Refresh our current
measured position
            printf("X completed homing, current position: \t%8.0f \n",
x.Motion.PosnMeasured.Value());
            printf("Soft limits now active\n");
        }

        //Printing soft limits
        printf("Soft Limit X 1 = %d \n", (int)x.Limits.SoftLimit1);
        printf("Soft Limit X 2 = %d \n\n", (int)x.Limits.SoftLimit2);

        if ((x.Limits.SoftLimit1.Exists() && x.Limits.SoftLimit2.Exists()
&& y.Limits.SoftLimit1.Exists()
        && y.Limits.SoftLimit2.Exists() && z.Limits.SoftLimit1.Exists() &&
z.Limits.SoftLimit2.Exists()) == false) {
            printf("Make sure to setup Soft limits\n");
            return 0;
        }
    }

    //Initializing oculus
    if (ovr_Initialize(nullptr) == ovrSuccess) {
        printf("Oculus initialized\n", 0);

        //Creating sessions and other Oculus related objectts
        ovrSession session = nullptr;
        ovrGraphicsLuid luid;
        ovrResult result = ovr_Create(&session, &luid);
        ovrInputState inputState;
        //Checking if session could be created, if true it will run the
while(true)
        if (result == ovrSuccess) {
            //Creating a empty trackingstate
            ovrTrackingState ts;
            while (true) {

                //Get tracking state, contains all sensordata for tracking
                ts = ovr_GetTrackingState(session, 0, true);

                //Get handpose
                ovrPosef rightHandOrientation =
ts.HandPoses[ovrHand_Right].ThePose;
                ovrPosef leftHandOrientation =
ts.HandPoses[ovrHand_Left].ThePose;
                handPoseOrient = rightHandOrientation.Orientation;

                //Multiplying orientation with the inverted
HandPoseOrient0. HandPoseOrient0 contains the values of the center point
which the user points to in the calibration
                calibratedHandPoseOrient = handPoseOrient *
handPoseOrient0;
                calibratedHandPoseOrient.GetYawPitchRoll(&a, &b, &c);
                calibratedX = b;
                calibratedY = a;

                //Get ThumbStick values for left and right thumb
                ovrVector2f rightStick =
inputState.Thumbstick[ovrHand_Right];
                ovrVector2f leftStick =
inputState.Thumbstick[ovrHand_Left];

                //If calibration is not done, do calibration process until
calibration is done and calibrationFlag == 1
                do {
                    ts = ovr_GetTrackingState(session, 0, true);
                    ovrPosef rightHandOrientation =
ts.HandPoses[ovrHand_Right].ThePose;
                    handPoseOrient = rightHandOrientation.Orientation;

                    if (counter == 1000000) {
                        printf("Please press B to calibrate\n");
                        counter = 0;
                    }

                    if (OVR_SUCCESS(ovr_GetInputState(session,
ovrControllerType_Touch, &inputState)) && inputState.Buttons & ovrButton_B)
{

                        handPoseOrient0 = handPoseOrient.Inverse();

                        setAngle(2048, *portHandler, *packetHandler, 0);
                        setAngle(2048, *portHandler, *packetHandler, 1);

                        calibrationVector = calibrate(ts, session,
inputState, handPoseOrient0);
                        minX = calibrationVector[0];
                        maxX = calibrationVector[1];
                        minY = calibrationVector[2];
                        maxY = calibrationVector[3];
```

```
                            calibrationFlag = 1;
                            calibrated = true;
                        }
                        counter++;
                    } while (calibrationFlag == 0);

                    //Checking if buttons is pressed
                    if (OVR_SUCCESS(ovr_GetInputState(session, 
ovrControllerType_Touch, &inputState))) {
                        //Deadmans switch. If handtrigger is pressed, run 
servos. If handtrigger is released Calibrated == fals
                        if (inputState.HandTrigger[ovrHand_Left] > 0.5f) {

                            //calibration,  if handtrigger is released 
Calibrated == false which stores the new origin.
                            if (calibrated == false) {
                                handPoseOrient0 = handPoseOrient.Inverse();
                                calibrated = true;
                            }

                            //If B is pressed, recalibrate max and min values
                            if (OVR_SUCCESS(ovr_GetInputState(session, 
ovrControllerType_Touch, &inputState)) && inputState.Buttons & ovrButton_B) 
{
                                setAngle(2048, *portHandler, *packetHandler, 
0);

                                setAngle(2048, *portHandler, *packetHandler, 
1);

                                calibrationVector = calibrate(ts, session, 
inputState, handPoseOrient0);
                                minX = calibrationVector[0];
                                maxX = calibrationVector[1];
                                minY = calibrationVector[2];
                                maxY = calibrationVector[3];
                            }



                            //Calculating which angle/servo input to send to 
servo

                            double angleX = calcAngle(minX, maxX, calibratedX, 
0, inverse);

                            double angleY = calcAngle(minY, maxY, calibratedY, 
1, inverse);


                            //Check if values are in range, otherwise do 
nothing

                            if (angleX > 1100 && angleX < 2900) {
                                setAngle(angleX, *portHandler, *packetHandler, 
0);

                            }
                            if (angleY > 1100 && angleY < 2900) {
                                setAngle(angleY, *portHandler, *packetHandler, 
1);

                            }

                            //Get speed from stick
                            jX = static_cast<double>((leftStick.x));
                            jY = static_cast<double>((leftStick.y));
                            radius = sqrt(pow(jX, 2) + pow(jY, 2))*VEL_LIM_RPM;
                            double m = max(abs(jX), abs(jY));
                            //If stick is centered, set speed to 0, else set
speed to vY and vX. setSpeed is threaded to not stall
                            if (m == 0) {
                                std::thread t1(setSpeed, 0, std::ref(y), 5000);
                                std::thread t2(setSpeed, 0, std::ref(x), 5000);

                                t1.join();
                                t2.join();

                                xPreviousSpeed = 0;
                                yPreviousSpeed = 0;
                            }
                            else {
                                vY = (radius * jY) / -m;
                                vX = (radius * jX) / m;

                                if (abs(vY) < 100) {        //If values are
really small, dont set speed. This is to be able to run along a single
axis.
                                vY = 0;
                                }
                                if (abs(vX) < 100) {
                                vX = 0;
                                }
                                std::thread t1(setSpeed, vX, std::ref(x),
xPreviousSpeed);
                                std::thread t2(setSpeed, vY, std::ref(y),
yPreviousSpeed);
                                t1.join();
                                t2.join();
                                xPreviousSpeed = vX;
                                yPreviousSpeed = vY;
                            }

                            //Checks if X or Y is pressed to set Z height
                            if (inputState.Buttons & ovrButton_X) {
                                while (inputState.Buttons & ovrButton_X &&
OVR_SUCCESS(ovr_GetInputState(session, ovrControllerType_Touch,
&inputState))) {
                                std::thread t1(setSpeed, 1000, std::ref(z),
0);
                                t1.join();
                                }
                                Sleep(10);
                                std::thread t1(setSpeed, 0, std::ref(z), 1000);
                                t1.join();
                            }

                            if (inputState.Buttons & ovrButton_Y) {
                                while (inputState.Buttons & ovrButton_Y &&
OVR_SUCCESS(ovr_GetInputState(session, ovrControllerType_Touch,
&inputState))) {
                                std::thread t1(setSpeed, -1000,
std::ref(z), 0);
                                t1.join();
                                }
                                Sleep(10);
                                std::thread t1(setSpeed, 0, std::ref(z), -
1000);
                                t1.join();
                            }
                            //Break while loop if button A is pressed
                            if (inputState.Buttons & ovrButton_A) {
                                break;
                            }

                            if (OVR_SUCCESS(ovr_GetInputState(session,
ovrControllerType_Touch, &inputState)) && inputState.Buttons &
ovrButton_Enter) {
                                if (inverse == 1) {
                                inverse = -1;
                                }
                                else {
                                inverse = 1;
                                }
                            }
                        }
                        //If deadmans switch/the right hand trigger is
released, set speed to zero, angle to zero for dynamixel servos and
calibrated ==
                        //false for new origin placement when trigger is
pressed.
                        else {
                            std::thread t1(setSpeed, 0, std::ref(x), 3000);
                            std::thread t2(setSpeed, 0, std::ref(y), 3000);
                            t1.join();
                            t2.join();
                            setAngle(2048, *portHandler, *packetHandler, 0);
                            setAngle(2048, *portHandler, *packetHandler, 1);

                            calibrated = false;
                        }
                    }
                }
            }
            printf("Oculus closing \n", 0);
            ovr_Destroy(session);
        }
        printf("Ports closing \n", 0);
        myMgr->PortsClose();
        z.EnableReq(false);
        y.EnableReq(false);
        return 0;           //End program
}
```

Appendix 5: User manual

This appendix contains a user manual for the robot as well as guidelines of using the different applications associated with the servos and camera. The manual is written in Swedish as requested by the company.

Teknic ClearView
Installerat på datorn finns programmet Teknic ClearView. Detta är ett program som innehåller diagnostiska funktioner till motorerna som styr X,Y och Z linjärenheterna. Dessutom finns det möjlighet för vissa inställningar som sparas i servonas hårdvara. Exempelvis är detta konfiguration för hemkörning, så som hastighet och maximalt moment som skall uppnås innan en punkt anses vara ändlägespunkten.

Hemkörning innebär att motorn går till ändläget, alltså i slutet av linjärenhetens rörelseområde. Hemkörning måste alltid genomföras så fort strömmen har varit avstängd. Detta gör automatiskt i programmet som är skrivet i detta examensarbete, men vid exempelvis vidareutveckling måste man ta hänsyn till detta. I inställningarna för hemkörning finns där fyra parametrar relevanta för denna roboten. Det är parameterna ”homing speed” vilket anger hastigheten, ”homing acceleration” som anger accelerationen, ”torque limit” som är momentet motorn måste uppnå för att anses var i ändläge och slutligen en kompensering som ser till att nollpunkten är innanför linjärenhetens ändläge. Hemkörning är väldigt viktigt att genomföra då motorerna inte kommer att utföra någon rörelse om det inte är utfört.

”Soft limits” är en viktig funktion att ta hänsyn till. Det är värdet från den inkrementella pulgivaren som servot håller sig innanför för att se till att linjärenheterna inte kan krocka med varandra eller med ställningen. Soft limit sätts till ett specifikt intervall, för X-servot är det exempelvis 0 – 560 000 pulser. Skulle man skicka ett kommando som innebär att servot skulle åka utanför detta område kommer servot automatiskt bromsa in för att stanna på gränsen.

Servomotorerna har många andra funktioner inbyggda i ClearView, så som grafer som i realtid visar hastigheter, accelerationer, moment och annan data som går att läsa av från servot. Dessutom kan man utföra en kalibreringsprocess där motorerna finjusterar PID parameterna för att matcha det mekaniska system det är kopplat till.

Flir SpinView
Installerat på datorn finns även programmet Flir SpinView. Detta programmet används för att visa bilden från kameran. Programmet används genom att det startas upp från en genväg på skrivbordet. I listan till höger kommer kameran att listas. Genom att klicka på kameran kommer flertalet inställningar upp. Under inställningarna måste upplösningen ändras till 800x600 pixlar. Därefter startar man strömmingen av bilder genom att klicka på den gröna ”play” knappen.

R+ Manager
Installerat på datorn finns programmet R+ manager. Det används för kommunikation med Dynamixel servomotorerna, dvs de som styr riktningen av spraydysan. Genom att starta R+ manager och klicka på ”diagnostics” kommer man åt servomotorernas register. I listan till vänster väljer man vilken motor som man kommunicerar med, och därefter kan alla värden i registret ändras, så som exempelvis PID parametrar, hastighetsprofiler, accelerationsprofiler, vinkeln och hastigheten. Programmet används i princip bara för diagnostik.

Användarmanual för programmet till fjärrstyrning,
Steg 1: Först måste Oculus sensorerna kalibreras. Detta görs i Oculusprogrammet ”Oculus Rift” som är installerat på datorn. Genom att gå in under ”settings” i programmet finner man ”reset tracking data”. Operatören får därefter följa guiden i programmet. Detta måste endast göras de gånger sensorerna har flyttats fysiskt.

Steg 2: Därefter startas Flir SpinView och kameralänken startas genom att klicka på den gröna ”play-knappen”.

Steg 3: På skrivbordet ligger en exe-fil vid namn ”Metso målningsrobot”. Genom att dubbelklicka på denna startar programmet och en svart kommandotolkruta öppnas. Initiering sker automatiskt av alla motorer och andra funktioner. Hela processen skrivs ut i kommandotolken och eventuella felkoder kommer även att skrivas ut, exempelvis att seriekommunikation inte går att starta vilket är ett vanligt problem om t.ex R+ manager eller Teknik ClearView är startartade samtidigt.

Steg 4: När programmet har initierat klart kommer en text upp ”Please press B to calibrate”. Operatören tar då upp handkontrollerna och placerar handen i det område han vill röra handleden. Därefter trycker operatören på B-knappen på touch controllern och en ny utskrift kommer upp, ”Go maximum up and press B”. Operatören vinklar då handen upp så långt han känner är bekvämt och klickar därefter B. Nästa steg är att definiera ”maximum down” vilket görs genom att ta handen så långt ner han känner är bekvämt och knappen B trycks ner. Därefter repeteras proceduren för vänster och höger.

När kalibreringsprocessen är klar har operatören kalibrerat det arbetsområde som handkontrollen kommer att läsa av data från. Ju mindre området är, desto mindre rörelser krävs för att handleden ska kopiera rörelsen. Med andra ord kalibreras känsligheten.

Steg 5: För att utföra målning trycker operatören in knappen vid pekfingret på den vänstra kontrollern. Denna knapp är dödmansgrepp och så fort knappen släpps stannar roboten. Medan dödsmansgrepp är intryckt styrs placeringen av X och Y med hjälp av den vänstra joysticken och höjden (Z-axeln) höjs med Y-knappen och sänks med X-knappen på den vänstra kontrollern.

Steg 6: För att spraya trycker operatören på den högra kontrollerns knapp vid pekfingret, vilket även kräver att dödmansgreppet är intryckt.

Användarmanual för automatiserat program
För det automatiserade programmet utförs först steg 1-4 enligt ovan. Därefter trycker operatören på A på den högra kontrollern. Följt av detta kommer en utskrift som ber operatören styra roboten med den vänstra joysticken till startpunkten, exempelvis det vänstra nedre hörnet på bordet med produkter. När roboten är styrd till startpunkten klickar operatören på A igen och går därefter till det hörnet diagonalt över på andra sidan, i detta fallet det övre högra hörnet och klickar på A igen. Nu startar roboten automatiskt målning över det markerade området. Operatören kan släppa kontrollerna och dödmansgrepp och vänta på att roboten återgår till sin startposition, därefter är målningen klar.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER'S THESIS *Date of issue* December 2018 *Document Number* TFRT-6064
Author(s) Daniel Jovanovski	*Supervisor* Gunnar Lindstedt, Dept. of Industrial Electrical Engineering and Automation, Lund University, Sweden Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle

Semi-automation of a spray painting robot

Abstract

Spray painting by hand is a strenuous task for the operator and his shoulder joints. Today this is how the procedure is done at the Metso Sweden AB factory in Trelleborg. This thesis will analyze the spray painting process and investigate and implement a semi-automated robot which can be remote controlled from a safe environment as well as automated. The implementation has a focus on using vision feedback with AR-technology together with Oculus Rift touch controllers for controlling the robot.

The thesis covers the entire process of investigating possibilities, design, construction and implementation of a Cartesian robot. Due to a large amount of custom-made parts with small batches the company sees no reason to implement a fully automated robot solution which is why a semi-automated solution is desirable. The vision is to be able to automate the standard products while the small batches can be painted using tele-operation from outside the spray box in a safe environment.

The thesis resulted in a proof-of-concept for a semi-automated solution which worked as intended, although for a fully functional work cell, a few modifications must be done as well as implementing a GUI which was not covered in this thesis.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-62	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/