Kristoffer Hermansen

Automating parts of the hardbanding process using computer vision and ROS

Master's thesis in Industrial Cybernetics
Supervisor: Ole Morten Aamo
June 2019

Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Kunnskap for en bedre verden

Kristoffer Hermansen

Automating parts of the hardbanding process using computer vision and ROS

Master's thesis in Industrial Cybernetics
Supervisor: Ole Morten Aamo
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Acknowledgements

In all the work I have done with my master thesis, it would not have been possible to achieve the same quality of result without the help of my supervisors from NTNU, Ole Morten Aamo and Lars Tingelstad, as well as my supervisor from WellConnection Mongstad, Henrik Meland Madsen. I would like to thank Heine Pedersen at InCon, Tom Trones at WellConnection and Joakim Linge at Goodtech for assisting in the research regarding the PipeFlow database. In addition I would like to direct a special thanks to Tor Andre Kjørlaug for the initial idea for this thesis. As with most kinds of work, the opportunity to get constructive feedback and discuss the challenges at hand with the right people is an important key for success.

I would also like to take the opportunity to give a special thanks everyone at WellConnection Mongstad, formerly known as Frank Mohn Mongstad. For all the support you have given me during the last nine years on my journey from apprenticeship, to industrial mechanic, to bachelor's degree in Subsea Technology and now finally to the end of my master's degree in Industrial Cybernetics at NTNU. I am truly grateful.

Finally I would like to thank my family and loved ones for all the support and encouragement I have received during my time as a student.

Kristoffer Hermansen
10.06.2019

Preface

This report is my master's thesis within Engineering Cybernetics (TTK4900) and represent what I have learned as a student at the Norwegian University of Science and Technology. The work done in this report is a combination of independent development and research. Where the research consists of both online research and a lot of conversations and discussions with colleagues at WellConnection Mongstad. From WellConnection Mongstad, managing director Henrik Meland Madsen, has been my internal supervisor and has provided important feedback throughout the report. The initial idea for the report came from workshop supervisor and technician Tor Andre Kjørlaug, a lot of the fundamentals for the system I have designed around, has been based on discussion with him. WellConnection Mongstad has also supplied me with an office, from which I have spent the majority of the semester working from. Ole Morten Aamo and Lars Tingelstad has been my supervisors from NTNU and have been available for guidance with both technical and formal matters.

During the report, I have created an environment in which the validation of the computer vision code could be done. This was done using Autodesk Inventor Professional 2019. The environment is based on the work performed in the specializing project (Hermansen, 2018), with some modifications, such as using exact measurements from the drill-pipe data-sheet to create the new drill-pipes for the computer vision. The vast majority of code in this report has been written in the programming language python 2.7 and parts of the feature detection program is based on prior work in python 3.7 from the specialization project (Hermansen, 2018). The scripts are developed from scratch with inspiration from the large OpenCV and ROS communities as well as the standard documentation available. OpenCV has been the library which has been used for all computer vision and image processing, and ROS kinetic has been used to structure the different scripts to work together. During the semester, the operating system was switched from Windows 10 to Ubuntu 16.04 LTS to achieve desired compatibility between Python, OpenCV and ROS. A thermal camera, FLIR One Pro was also purchased to test the abilities of using thermal images together with computer vision.

Summary

This master thesis takes a dive into some of the engineering needed to create a new automated production line for repair welding on worn drill-pipes at WellConnection Mongstad. The main motive for this thesis is related to a current bottleneck in the existing semi automated production line. Some of the drill-pipes which is to be repaired require a larger amount of welding to replace impurities removed during the machining process. This creates a challenge because the amount of welding needed quickly heats the pipes to temperatures where the inner coating is exposed to temperatures capable of destroying the inner coating. Therefore a lot of waiting is required between the strings of weld to avoid damaging to the coating. This causes an entire welding station to be set on hold, and not produce while the pipe cools, this is the bottleneck.

The new production line is planned to have a welding robot which is able to move from pipe to pipe and weld depending on the temperatures of the pipes to avoid damage to the pipe and ensure a more continuous production.

The work done in this report is based on the report (Hermansen, 2018) where it was investigated if the use of computer vision could be used to control a welding robot autonomous. Some of the work done in this report is upgrading the former feature detection program through improved search algorithms and added the functionality to detect two grooves in the drill-pipes instead of just the start of one groove. Computer vision has been used to estimate the different diameters needed to then create a welding planner which calculates the appropriate weld build structure and the needed rotational speed to achieve the desired weld height. A thermal camera was tested and showed promising results as a thermal sensor aided by computer vision. Finally parts of the code where implemented to a ROS network to create a better interface between the programs and the future hardware needed to realize the new production line.

Sammendrag

Denne masteroppgaven tar ett dypdykk inn i noe av ingeniørarbeidet som trengs for å skape en ny autonom produksjonslinje for reparasjonssveis av slitte borerør hos Well-Connection Mongstad. Hovedmotivasjonen for denne masteroppgaven er knyttet til en eksisterende flaskehals i den eksisterende halvautomatiske produksjonslinjen. Noen av borerørene som må repareres trenger større mengder sveis for å erstatte urenheter som er fjernet under maskineringsprosessen. Dette skaper en utfordring fordi den store mengden sveis vil raskt tilføre varme til røret, som medfører at beskyttelsesbelegget inne i røret kan bli ødelagt. Derfor er det idag mye venting mellom sveisestrengene for å unngå å skade beskyttelsesbelegget innvendig. Dette medfører at en hel sveisestasjon blir satt på vent, og vil ikke kunne produsere noe mens røret kjøles, dette er flaskehalsen.

I den nye produksjonslinjen planlegges det å ha en sveiserobot som kan bevege seg fra rør til rør og sveise avhengig av rørtemperatur for å unngå å skade rørene samt sikre en kontinuerlig produksjon

Arbeidet gjort i denne rapporten er basert på rapporten (Hermansen, 2018) hvor det ble undersøkt om bruk av datasyn kunne bli brukt til å kontrollere en sveiserobot autonomt. Noe av arbeidet som er gjort i denne rapporten er å oppgradere det tidligere gjenkjennelses programmet ved å forbedre søkealgoritmen og legge til funksjonalitet som å kunne oppdage to maskinerte groper istedenfor en. Datasyn har blitt brukt til å estimere de ulike diametre som trengs for å lage en sveiseplanlegger i python som regner ut passende sveiseoppbygning samt nødvendig rørrotasjon for oppnå en ønsket sveisehøyde. Et termisk kamera ble testet og viste lovende resultater innen bruk som termisk sensor assistert av datasyn. Til slutt ble deler av koden implementert til et ROS nettverk for å lage en bedre overgang fra programmene laget til fremtidig utstyr som trengs for å realisere den nye produksjonslinjen.

Contents

List of Figures

Chapter 1

Introduction

This paper is being written on behalf of WellConnection Mongstad. WellConnection Mongstad is a supplier of inspection, maintenance and repair (*IMR*) services on equipment for the oil and gas industry. A substantial part of WellConnection's work is related to *IMR* of drill pipes. The *IMR* work on the drill pipes consists of several parts, and vary a lot depending on the state of the pipes received. The work initially consists of cleaning the pipes before inspection, then the amount of work required on maintenance and repair is decided on behalf of the inspection results. After this the pipes are sent through the automatic production line where all of the maintenance and repair work is handled (Hermansen, 2018).

For this report, the main interest is the repair of the soft-legging layer under the hard-banding welds. When the pipes are being mounted together, while drilling and when taken apart, a lot of wear and tear happens on the tool joints. The tool joint is the section on the drill pipe with the enlarged diameter and where iron roughneck attaches when screwing them together during operations. In order to extend the lifetime of the drill pipe, repairing the hard-banding weld is essential. The pipes are first sent to a lathe for machining to remove the old hard-banding. Sometimes it is enough to just remove the old layer and then weld a new hardbanding layer. Other times the pipe has deeper pores under the weld, which require a larger diameter to be removed until the surface is smooth and free of pores (Hermansen, 2018).

When the deeper pores are being removed, the diameter of the tool joint will decrease. Therefore the drill pipe needs to have another layer of weld to get the diameter back up before the hard-banding is welded. This layer of weld is called soft-lagging, which is a softer weld more similar to the quality of the pipe. Welding the soft-lagging layer is time consuming due to the large amount of welding needed, and a temperature restriction in order to not burn the coating inside the drill pipes (Hermansen, 2018).

This temperature restriction causes great time-delays as the drill-pipe need several breaks to cool down between the welding, effectively preventing a welding station to produce as much as it otherwise could. This is the fundamental reason for WellConnection Mongstad to create a new production line, which uses a different setup where a welding robot is mobile and can move between several drill-pipes to do work while the former drill-pipes are cooling down. Additional cooling is not considered in this report due to the risk of the weld cracking as a result of too rapid cooling.

In the autumn of 2018 the author wrote the paper "Using computer vision to control a robotic welder" (Hermansen, 2018) as the TTK4551 Specialization Project at NTNU. Based on that work, the problem definition for this master thesis was created by the author together with WellConnection Mongstad.

Problem definition
In this thesis, the task at hand will be to discretize the soft-lagging welding process into sub tasks which will be solved individually in python programs before they are combined to work together as one. The sub tasks are as follows:

- Investigate the possibilities for retrieving existing information on the pipes from existing PipeFlow database
- Upgrade the python program created with the paper (Hermansen, 2018) to locate two grooves instead of only one groove, and increase the robustness of the feature detection
- Use computer vision to estimate the pipe diameter at the machined groove as well as the tool joint diameter.
- Create a program which uses the geometry of the machined groove to adjust the needed pipe rotational speed to control desired weld height.
- Investigate whether thermal cameras can be used as a method for measuring pipe temperatures for drill-pipes between welding.
- Use ROS to divide the upgraded python program into nodes and connect these nodes.

Chapter 2

Theoretical Background

2.1 Computer vision

This is a clarification to specify that the whole computer vision theory chapter is copied from the previous paper from the same author (Hermansen, 2018)

2.1.1 OpenCV

In this paper OpenCV is used as the main library for computer vision. OpenCV is, in a more general term; "an open source computer vision and machine learning software library". OpenCV has a python interface and supports all the major operating systems. In this report the library is used to save a lot of time by using well established functions. Since OpenCV is open source, all the code can be further improved and tuned later without the need for any licenses.(Hermansen, 2018)

2.1.2 Noise reduction

Prior to the actual blurring, the image is read as a gray scale image. In order to detect edges in an image, noise reduction (smoothing or blurring) the image is essential to get good results. Without the initial blurring the edge detector will struggle to see the difference between textures and edges, making the result less than helpful. Blurring an image can be viewed as using a low pass filter on the image, since it reduces the noise in the image. There are several ways to blur an image, after looking at the mean filter method, median filter method and the Gaussian filter method, the latter gave the better results. (Hermansen, 2018)

The Gaussian filter looks at each pixel in the original image, and then the kernel used takes the surrounding pixels and tries to make a weighted average which becomes the new pixel value for the blurred image. It is the ability to easily prioritize the closest pixels more than the pixels further away that makes the Gaussian blur so applicable for use before edge detection. This Gaussian weight in the kernel does a better job preserving potential edges, which is why this method ended up being the preferred method for image blurring in this report. (Hermansen, 2018)

2.1.3 Edge detection

To detect edges, the general concept is to apply a high pass filter to detect changes in neighbouring pixels, which will be interpreted as edges. This can be done with a custom kernel which enlarges pixel values with large gradients, indicating the presence of an edge. However, tuning this kernel will not create a robust edge detector, and several steps are required to create an usable edge detector algorithm. There are several algorithms and techniques to achieving edge detection in an image, the one used in this paper is the Canny Edge Detection. The canny edge detection algorithm contains four steps:

1. Noise reduction to blur the image
2. Find the intensity gradients to locate edges
3. Non-maximum suppression in order to thin the edges.
4. Hysteresis thresholding in order to delete the unqualified edges.

(Hermansen, 2018)

2.1.4 Optics

The camera parameter matrix is a matrix containing important parameters which describe the intrinsic parameters of the camera. To obtain the intrinsic parameters, a camera calibration can be done. The result after the calibration is the camera parameter matrix, denoted K.(Hermansen, 2018)

$$K = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \qquad f_x = \frac{f}{\rho_w} \qquad f_y = \frac{f}{\rho_h} \tag{2.1}$$

Where f is the focal length, ρ_w is the width of a pixel and ρ_h is the height of a pixel. Further u_0 and v_0 are the pixel coordinates at the center of the image plane. The convention also gives the origin of the image plane is the top left corner, and moving horizontally to the right is the positive u-direction, while moving vertically down is the positive v-direction. ((Egeland, 2018b))

$$p = \begin{bmatrix} u \\ v \end{bmatrix}, \qquad \tilde{p} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.2}$$

2.2 Transformations

In order to express distances from one frame, to another frame with a different orientation, the different frames need to be linked by the use of rotation matrix. From equation 2.3 we see the rotation matrices which each describe a rotation around one axis, where R_x, R_y and R_z is the rotation around the x-axis, y-axis and z-axis respectively. (Hermansen, 2018)

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & Cos(\theta) & -Sin(\theta) \\ 0 & Sin(\theta) & Cos(\theta) \end{bmatrix} \quad R_y = \begin{bmatrix} Cos(\phi) & 0 & Sin(\phi) \\ 0 & 0 & 1 \\ -Sin(\phi) & 0 & Cos(\phi) \end{bmatrix} \quad R_z = \begin{bmatrix} Cos(\phi) & -Sin(\phi) & 0 \\ Sin(\phi) & Cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

In addition, the different frames, might be located in different places. When this is the case, one also has to take the distance between them into account when describing points from one frame to another. The translation from a point A, to a point B seen from frame A, can then be described using a translation vector $t^A_{AB} = [x, y, z]^T$. By the combination of translation and rotation, one can easily go from one frame to another and still be able to describe points and vectors from the other frame. The combination of translation and rotation can be described in one single matrix, called the Transformation matrix. (Egeland (2018a))

$$T^A_{AB} = \begin{bmatrix} R^A_B & t^A_{AB} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x \\ 0 & Cos(\theta) & -Sin(\theta) & y \\ 0 & Sin(\theta) & Cos(\theta) & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

In 2.4 the rotation happens to be a rotation around the x-axis in this case. The notation used for the transformation matrices is explained in equation 2.5 (Hermansen, 2018):

$$T^A_{AB} = T^{\substack{Seen\ from\ frame\ A \\ from\ point\ A\ to\ point\ B}} \tag{2.5}$$

2.3 Databases

A database, in its most simple terms, is a collection of data. Therefore a database can be anything from your shopping list to the phone books to customer data for large companies. (Robert Latek, 2004) There are a lot of RDBMS - Relational DataBase Management Systems available, the one relevant for this paper is the Microsoft SQL server (Microsoft, 2019). SQL is a data manipulation language used to interact with relational database systems.(Maier, 1983)

2.4 ROS

ROS, which stands for Robot Operative System is licensed under an open source, BSD license. It is used to make the creation of robot applications easier by providing essential libraries and tools.(Open Source Robotics Foundation, 2019)

Nodes are specialized programs used to execute some desired task. A node could operate by itself, but is most useful when it is set to operate with other nodes, effectively creating a network of nodes working together. Topics is essential for large networks of nodes to communicate with each other. A topic is a line of communication which is published by one node, this communication line can be subscribed to by any other node(TullyFoote, 2019). Each node can both subscribe and publish to several other nodes. ROS-master is the master which connects all the nodes so that they are publishing and subscribing to the correct node/nodes.(YanqingWu, 2019)

A simple example of how a topic works is shown in figure 2.1 All software in ROS is organized in packages, each package contains nodes, a data-set, configuration files and third-party piece of software among other things. (IsaacSaito, 2019b) For topics to be able to transport messages it is essential that the message type used is of the correct type. This is specified by a .msg file, where several message types can be added, such as a string, int8 or int8[] which define the type of data for ROS to handle. The different parts of the message also needs to have an individual variable for the specific parts of the .msg file to be extracted from the topics. (AustinHendrix, 2019)

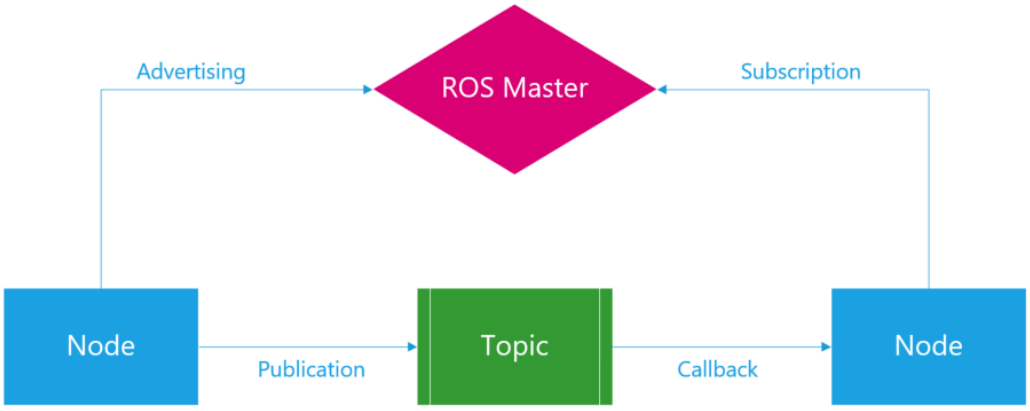


Figure 2.1: ROS Master, node and topic connection, self made with inspiration from (Noel Martignoni, 2019)

In addition to topics, there are two additional ways for nodes to communicate. The first one is ROS services. ROS services consists of a node containing the actual service, which is like a regular node by the fact that it often contains specialized programs. In order to actually get a response from the service, it needs to be called either by a client or directly from the terminal. ROS services allows nodes to respond only when the service is called, unlike topics which constantly publish if they have something to publish (AnisKoubaa, 2019). For services to be able to communicate, there has to be a clearly defined message that is to be sent, like for topics. For ROS services a .srv file needs to be created. A .srv file is different from the .msg file by the fact that all .srv files are divided into two

parts, a request and a response. This means that all ROS services require an input called a request in order to call on the service. When the ROS service has been called with the request it does whatever it is assigned to do, and then responds with a message defined in the response part of the .srv file. (DirkThomas, 2019)

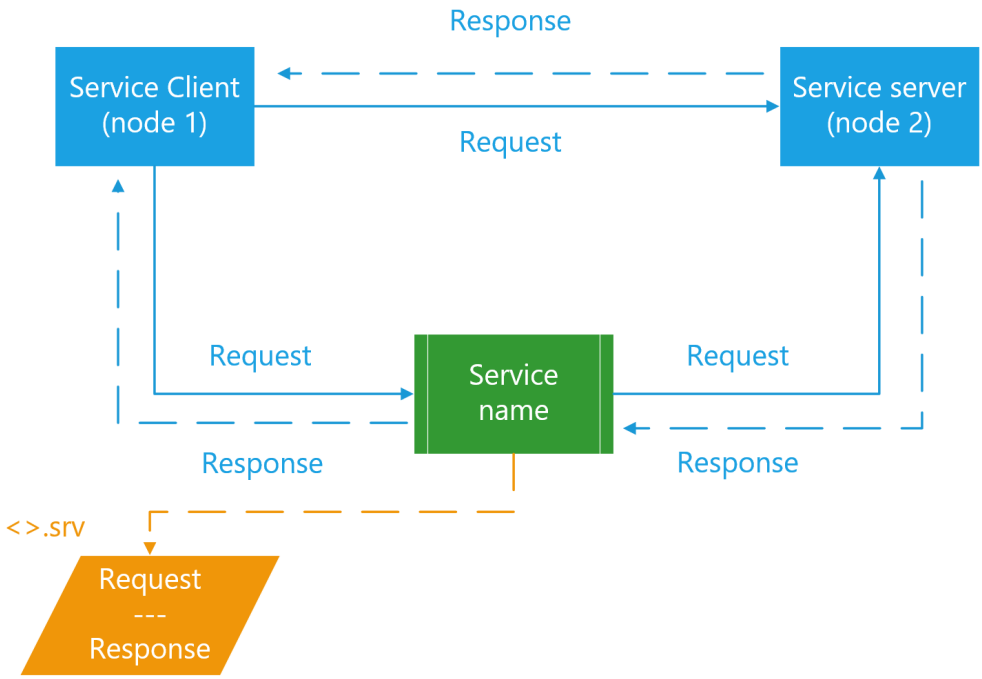


Figure 2.2: ROS service client and server connection, self made with inspiration from (Raymond Chen, 2019)

The second addition to topics is the ROS actions, which are quite similar to ROS services in some matters, but different in the fact that ROS actions are asynchronous, meaning that they can do several sub tasks at once. While ROS services are synchronous and have to wait until a service has finished before it continues. One important difference between the ROS service and the ROS actions would be that the ROS action could cancel an operation before it has been finished. The message used by actions are .action file. This is a more complex type than the two former message types. The .action files are divided into three parts; goal, result and feedback. The goal message describes what the client expects the server to do, this could be coordinates for a robot arm to move to. The feedback message is used to keep the action client updated on how the action server is doing on the way to the goal sent earlier, this might be the current robot pose. The result message is sent once from the action server to the action client when the goal has been completed, and might contain the final robot pose. (IsaacSaito, 2019a)

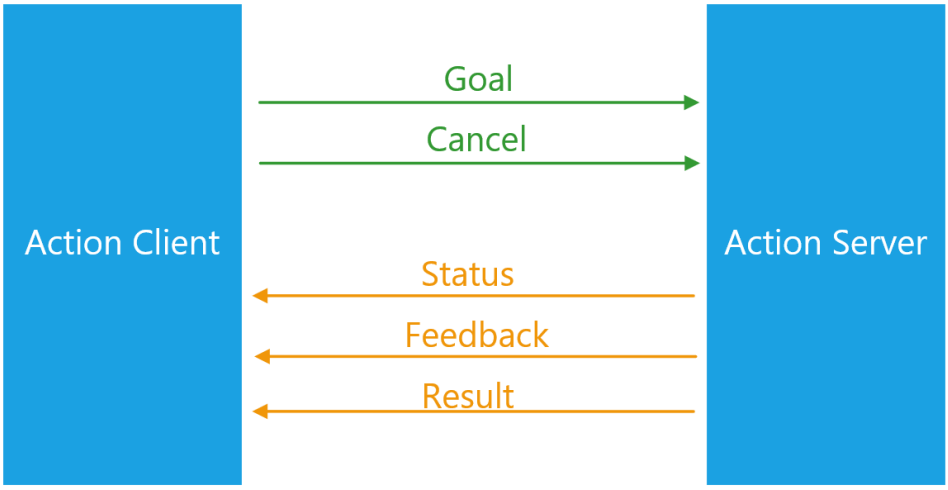


Figure 2.3: ROS action client and action server connected by standard topics, self made with inspiration from (Alberto Ezquerro, 2019)

2.5 MIG welding

MIG (metal inert gas) is a sub type of Gas metal arc welding which uses an inert shielding gas, most common for welding steel is a semi-inert gas, often a mixture between argon and carbon dioxide. The electrode used in MIG welding, often referred to as the MIG wire is a metallic alloy wire. The MIG wire feeder feeds the MIG wire through the weld gun and closes the circuit when it contacts the grounded work piece.(Wikipedia, 2019) The MIG wire is referred to as the welding rod later in the report.

2.6 Drill-pipe

The drill-pipe is an essential part of the oil and gas industry. When drilling for oil and gas, the drill bit is connected to the rotor by several drill-pipes. When drilling the drill-pipes are connected together as the drilling depth is increased. Each drill-pipe is one continuous pipe but it can be categorized into three section: The Pin end has the male threaded connection, and is connected to a tool joint. The tool joint has an enlarged outer diameter which is where the Iron Roughneck grips the pipes when torquing up the pipes, see figure 2.4.

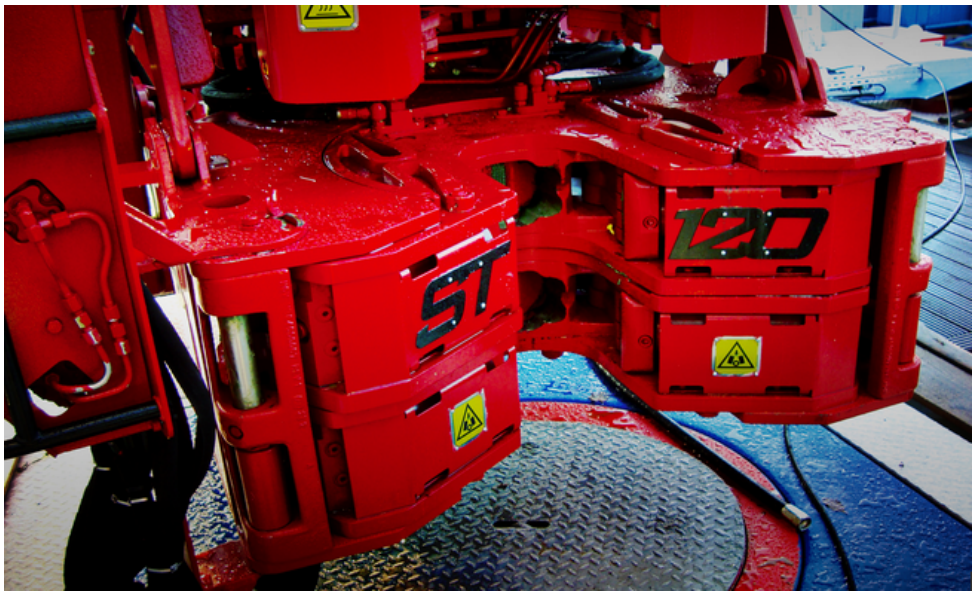


Figure 2.4: Iron roughneck from NOV, collected from (NOV, 2019)

The tool joint on the Pin side sometimes also includes hardbanding, a hardened weld which is to prevent the tool joint from wearing down during drilling operations. The Box end has the female thread connection and connects to the Pin threads of the next drill-pipe. The Box is also connected to a tool joint like the Pin end, however this end always has the hardbanding weld. The main body is the mid-section of each drill-pipe, this section has a smaller outer diameter than the tool joint and is the convention for specifying the size of the drill-pipe. For instance the 6-5/8" drill-pipe has a main body outer diameter of 6-5/8".

2.7 Heat transfer

The transformation of heat from an object can be divided into three main types of heat transfer, conductive heat transfer, convection heat transfer and radiation heat transfer. Further the transfer of heat energy require a temperature difference, where the energy always flow from the medium with higher temperature to the medium with the lower temperature. (the, 2015)

Conductive heat transfer is the transfer of energy from more energetic particles to the less energetic ones within a substance. The rate of heat conduction through a layer of constant thickness is proportional to the difference in temperature across the layer. And the cross-section area is normal to the direction of heat transfer and inversely proportional to the layer thickness, this yields the equation:

$$\dot{Q}_{conductive} = -k_t A \frac{(T_2 - T_1)}{\Delta x} \tag{2.6}$$

Where k_t is the thermal conductivity of the material, A is the area of the cross-section, T_1 and T_2 are the different temperatures and Δx is the layer thickness

Convection heat transfer is the transfer of energy between a solid surface and the liquid or gas in contact with the surface. Convection is the combined effect of convection and fluid motion.

$$\dot{Q}_{convective} = -hA(T_s - T_f) \tag{2.7}$$

Where h is the *convection heat transfer coefficient*, A is the surface area, T_s is the surface temperature while T_f is the fluid temperature in the surrounding fluid. The *convection heat transfer coefficient* is an experimental value and needs to be determined either by experiments or by comparing the situation to earlier experiments conducted.

Radiation heat transfer if the transfer of heat energy emitted as a result of electromagnetic radiation as waves (or photons). The equation for the radiation heat transfer rate is given below

$$\dot{Q}_{radiation} = -\epsilon\sigma A(T_{surf}^4 - T_{surr}^4) \tag{2.8}$$

Where ϵ is the emissivity of the surface, and lies between 0 and 1, where $\epsilon = 1$ is considered to be black-body. σ is the *Stefan-Boltzmann constant* where $\sigma = 5.67 \cdot 10^{-8}$. T_{surf} and T_{surr} is the surface and surrounding temperature respectively. Note that these temperatures needs to be given in Kelvin for this equation since the temperatures are in fourth power.

2.8 Thermal camera specifications

The thermal camera used in the Flir one pro with the USB-c connector which connects directly to the charging port of a suitable android phone. The camera is operated via an app developed by Flir.

FLIR ONE Pro	Specifications
Scene Dynamic Range	-20°C - 400°C
Visual resolution	1440x1080
Thermal resolution	160x120
Focus	Fixed 15cm - Infinity
Frame rate	8.7 Hz

Table 2.1: FLIR ONE Pro thermal camera (FLIR, 2019) specifications

Chapter 3

Method

3.1 PipeFlow database

To get a better idea of what kind of data is available through the inspection, the foreman for the inspection company InCon Mongstad, Heine Pedersen was contacted. From conversations with Heine Pedersen it became clear which kind of measurements was being conducted on the drill-pipes as well as how the inspection was performed. A list of the relevant measurements taken will be given in the results.

After gaining the information about the types of measurements, Heine referred me to Tom Trones, which further referred me to Joakim Linge at Goodtech whom is the maintainer of the database. The findings of this conversation can be found in the results.

3.2 Improvements of the feature detection code

The first objective was to improve the reliability of the feature detection, this was done in several steps, where the first was to improve the edge detected image. This was done by using several edge detected images at different settings, and then summing all images together and dividing on the number of images used in the setup. The edge detected images generated from using the canny edge detection algorithm has two parameters used to tune what is considered an edge and what is not. The first parameter was placed in a for loop and the second parameter was then placed in a nested for loop. At the end of the nested for loop a canny edge detector generated an image and added together with any former images generated. In order to sum together gray-scale images it is necessary to change the data type to int32, thus allowing values higher than 255 to be stored. after the average image was generated, the data type was returned to int8. The averaged image was then filtered through a threshold filter, where any pixel values under a user specified value was replaced with the pixel value of zero, also known as black.

To determine whether an object in the image is the feature which is being searched for, the program from the report (Hermansen, 2018) used the cv2.matchTemplate function. The function takes three arguments, the image to be searched, also referred to as the source image, the image to be searched for, also referred to as the template image and

finally the matching method. The value returned from the function is a number from 0 to 1 describing how close the match between location in the source image and the template image. In this report the searching algorithm has been altered by having a dynamic threshold. By dynamic threshold the search algorithm is running through a for loop within the desired user specified threshold, starting at the highest threshold value and running the loop until the first detection has been made.

After detecting the weld start, the cv2.matchTemplate receives a new template image to locate the weld end. A new constraint was inserted to pass any matches with more then 10 pixels of difference in the v image coordinate. Any matches found to the right side of the weld start was also ignored by the algorithm. The search will go on until a match that approves the new constraint is found, or no matches is found.

The next improvement to be done was allowing the cv2.matchTemplate function to have access to more template images. The first step was done by creating a folder for weld start template images and a folder for weld stop template images. Then a listdir was imported from the python "os" package, which enabled a function to list all the filenames in a directory, such as the new folders just created and then the save the result as a list of several strings. This list of filenames is then inserted via a for loop which runs the feature detection algorithm described above. This was done for both the weld start template images and for the weld stop template images. If a template does not get approved by the threshold and the constraints, the template image is skipped and the next template image is tested.

To allow for several groves to be located, the feature detection algorithm can run again, this time with additional input on the former groove located and constraints are given to avoid findings to close to the former grove. Additional constraints also gives that if the new groove located is with a higher image coordinate in the v-direction, that is lower on the image, then the groove must be within the former grove. Likewise if the new groove is located at a higher diameter, it needs to have the weld start and end located outside the former groove.

The images used to test the code are generated in Autodesk Inventor Professional 2019, and in order to get consistent images a JPEG image is exported from a camera inside inventor. This image is of size 3017x1704 pixels. To reduce the time spent searching the image for the different templates, a region of interest was derived. The region of interest was determined from the tool joint neck, which is a recognizable part of all drill-pipes considered in this report. A functionality searching for angled lines through the cv2.HoughLinesP function is used in combination with constraints ignoring all lines which is outside the user specified angle, for this report the angle was set to be between 15 and 25 degrees. In combination with this another cv2.HoughLinesP function was created to detect the tool joint diameter and generate a line following this diameter. A final function was created to calculate the intersection between the tool joint line and the neck line, which then gives a point for the neck location. This image coordinate is then used to generate the relevant region of interest by drawing a rectangle which will later become the region of interest.

With the known neck position and the desired size of the region of interest, the image is then cropped by creating a new image which consists of only the data inside the rectangle described above. This image is the introduced as the new source image for the feature detection algorithm and the pixel offset is saved to be able to translate the location back

to the original image. To transform the image coordinates to 3D coordinates the same function that was used in (Hermansen, 2018) was used.

3.3 Diameter estimation

To estimate the diameter of the different grooves, the cv2.HoughLinesP function is used in combination with constraints to filter out the non-relevant lines. The first constraints used is to ignore all lines with an angle greater then a user specified vale, for instance +- 1 °, this was done using the expression given in equation 3.1. Further the positional data from the weld start and stop is also known at this part of the code and is used to decide how long lines which are to be considered. As well as the location of the line start and end point additionally reducing the allowed height locations of located lines depending of the weld start and stop location. Finally the image coordinates of the located groove diameter is transformed to 3D coordinates using the same function mentioned in the former paragraph.

$$\alpha = \arctan\left(\frac{|v_2 - v_1|}{|u_2 - u_1|}\right) \cdot \frac{180}{\pi} \tag{3.1}$$

Where u_1 and u_2 is the horizontal start and stop pixel coordinate respectively, while v_1 and v_2 is the vertical start and stop pixel coordinate respectively. The tool joint diameter has already been detected in the detection software, further the horizontal lines are found by searching with the cv2.HoughLinesP within the different weld start and weld stop locations.

3.4 Weld planning

The weld program uses several parameters to calculate the relevant information which is to be given to the hardware. From the feature detection software the start and stop locations are given. Depending on the number of features in the feature detection program, the weld planner decides how many grooves which is actually present on the drill-pipe. This will split the program in two, where one part is designed to handle pipes with only one groove and the other is designed to handle pipes with two grooves. In the respective programs the feature detection results is also used to calculate the Groove width, which is further used combined with the welding string width to get the needed number of weld strings to fill the groove.

$$Strings = \frac{Groove\ \ width}{String\ \ width} \tag{3.2}$$

The diameter estimation is used to calculate the needed weld height to fill each groove. The program has a maximum allowed weld height, and the needed number of layers to

fill the groove is calculated by the program to make sure no layer gets above the user specified maximum layer height.

When the different layer heights and the numbers of strings in the different layers have been calculated, the program calculates the rotational speed needed to achieve the desired height for each layer. The rotational speed is calculated by using the volume of weld which is applied by the welder and the cross-section for each welded string. The applied weld volume is given from the feed rate of the welding machine and the weld rod diameter. The applied weld volume per time:

$$\dot{V} = \frac{\pi}{4} \cdot d_{rod}^2 \cdot f \tag{3.3}$$

Where $\dot{V}$ is the applied weld volume per minute, d is the diameter of the weld rod and f is the feed rate of the weld rod.

Further we can convert the weld flow to a volume by dividing by the number of rotations per minute.

$$V = \frac{\dot{V}}{n} \tag{3.4}$$

Where n is the rotations per minute. To connect the applied weld volume to the welded cross-section a model for the weld geometry is needed. The model to be used is a simplified one where the weld cross-section is assumed to be rectangular. Weld volume per revolution is then:

$$V = \frac{\pi}{4} \cdot (D^2 - d^2) \cdot w \tag{3.5}$$

Where D is the desired diameter after the weld and d is the initial diameter while w is the weld string width. Note that a new D and d is applied for each layer so that the difference in height is accounted for, this is needed to achieve the correct weld height.

Finally we can connect the two formulas together and solve for the rotational speed n, which is the parameter that is to be adjusted to control the weld height since the welders feed rate is assumed constant.

$$\frac{\dot{V}}{n} = \frac{\pi}{4} \cdot (D^2 - d^2) \cdot w \tag{3.6}$$

$$n = \frac{4}{\pi} \cdot \frac{\dot{V}}{(D^2 - d^2) \cdot w} \tag{3.7}$$

$$n = \frac{4}{\pi} \cdot \frac{\frac{\pi}{4} \cdot d_{rod}^2 \cdot f}{(D^2 - d^2) \cdot w} \tag{3.8}$$

$$n = \frac{d_{rod}^2 \cdot f}{(D^2 - d^2) \cdot w} \tag{3.9}$$

The output from the weld planner depend on the number of grooves as well as the number or layers needed. Either way the number of strings needed for the layer is given, the height of the layer, the pipe rotational speed, the estimated use of welding rod and the effective welding time, not considering any cooling breaks. A pipe welded in the soft-lagging progress can be seen in figure 3.1.

Figure 3.1: Drill-pipe welded by the author in the summer of 2018, notice the rather flat top as a result of several individual strings

3.5 Temperature measurements

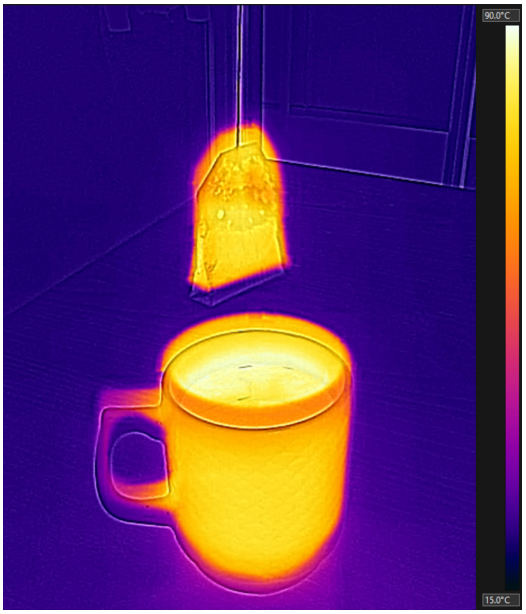
The temperature is an important control parameter in order to make the production line profitable. The temperature decides which pipe it is preferable to weld on in order to avoid burning the coating inside, as well as optimizing the number of transitions between the drill-pipes to complete the welds.

The use of a thermal camera combined with computer vision is intended to create a flexible sensor solution. The location of the camera and the precision of this location is essential to achieve accurate measurements. Since the feature detection program already holds the location of the weld start and stop in 3D coordinates, transforming these coordinates to the camera frame of the thermal camera gives the ability to calculate the image coordinates of each relevant pipe-section seen from the thermal image. This transformation is done by the existing code from the paper (Hermansen, 2018). From this, the region of interest can be found for each pipe and further the temperature of the each region of interest can be found from interpreting the thermal image.

In order to generate the thermal image, a FLIR one pro camera was used. To get familiarized with thermal images some initial testing was performed and a thermal image taken was taken of a cup of tea, as can be seen below. The image is a combination of both the thermal image and a normal image overlapped to provide more detailed contours. This image can be transformed to different types within the FLIR app. The standard in the

app is the iron palette as shown below in figure (b) below. To simplify the image processing and to be able to retrieve temperatures from the image, a grey-scale heat palette can be specified as shown in figure (c) below. The highest temperature of the object in the image can be determined from the app and the IR-scale can be adjusted manually. This means that one can define a maximum temperature which is linked to the pixel value of 255 (white) and a minimum temperature which is linked to the pixel value of 0 (black).

(a) Normal image of a cup of tea



(b) Iron palette underlined with original image

(c) Thermal image using only grey palette

Figure 3.2: Test images of FLIR One Pro camera

To test how the thermal camera works on the drill-pipes, a thermal image was taken of a drill-pipe which had just been welded on. This image was then given the grey-scale

palette and inserted in a python-script. From the Flir app its was given that the maximum temperature in the image was 200 °C. Inside the Flir app, the maximum temperature was set to 200 °C and the minimum temperature was set to 20 °C. This means that a pixel value of 255 equals 200 °C and a pixel value of 0 gives 20 °C. This was then translated inside the script so that the pixel value matches the actual temperature. This could also be done by initially setting the thermal scale in the Flir app from 0 °C to 255 °C. The script then creates a region of interest of which the script finds the highest pixel value, which then also gives the highest temperature. The location of this value is also collected and marked with a blue rectangle. The average temperature is the calculated inside the region of interest which is inside the red rectangle displayed on the image. Finally the result is displayed on the image for convenience.

3.6 ROS

Integrating all the programs created above is the next step, this is to be done using ROS. As mentioned in section 2.4, ROS is useful to create a framework to connect nodes. The first step to start using ROS was installing it on a computer. ROS is available for windows 10, however the author decided to switch operating system to the Linux based Ubuntu 18.04 LTS. Together with this version of Ubuntu the ROS Melodic distribution was acquired. After extensive problems with compatibility between different OpenCV modules such as cv_bridge and ROS modules such as rospkg and image_transport the author was forced to change from Ubuntu 18.04 LTS to Ubuntu 16.04 LTS as well as changing from ROS melodic to ROS kinetic. Still there was some initial problems with PYTHONPATH since the former code used python3 and the packages in ROS kinetic is created for python2, or more specifically python 2.7. This challenge was resolved by editing the _setup_util.py file inside the catkin workspace and the devel directory.

As explained in the theoretical background section 2.4, ROS has three main ways for nodes to communicate together. The method chosen is critical for how the ROS network is going to function. It was chosen to use ROS services for dividing the programs into several smaller nodes. More on why ROS services was chosen will be explained later in the discussion.

The camera_service was the first ROS service created for this report. To create a ROS service, first a new package is created using the catkin_create_package command in the terminal, followed by giving the package a name. In order to send an image from the camera service, it is needed to specify the message type so that ROS understands what we are sending and how to interpret it. For the camera_service, no existing .srv file seemed to work properly with the code, and therefore a custom service message had to be created. To create the new service message another package was created to hold all future custom service messages.

Within this new package a new directory called srv needs to be created, followed by the .srv file to be created inside this srv directory. The file created was called image.srv, and contained a simple string called "camera" as the request. The response was more complicated and contained several pieces of information, such as; width, height, is_bigendian, encoding and data in order to describe the image. See figure 3.3 to see a print screen of the file.

```
1 string camera
2 ---
3 uint32 width
4 uint32 height
5 uint8 is_bigendian
6 string encoding
7 uint8[] data
```

Figure 3.3: Custom service message: image.srv

After the file was created, all the dependencies needed to be updated. In ROS, a few files are created every time a new package is created. These files are placed inside the package to tell ROS information it needs in order to interact with anything inside the package. The files which are always generated by ROS is the CMakeList.txt and Package.xml. It is inside these files we must specify the dependencies for which the package depends on. For this package containing the custom .srv files the important dependencies can be seen in code A.7 and A.8. When all the necessary edits have been saved on the files, the package can be built by using the terminal command catkin_make, this hopefully returns something similar to what can be seen in figure 3.4. Similarly all the same has to be done for the camera_service package after it's dependencies have been edited.

```
kristoffer@HP: ~/catkin_ws 92x33
(base) kristoffer@HP:~/catkin_ws$ catkin_make
Base path: /home/kristoffer/catkin_ws
Source space: /home/kristoffer/catkin_ws/src
Build space: /home/kristoffer/catkin_ws/build
Devel space: /home/kristoffer/catkin_ws/devel
Install space: /home/kristoffer/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/kristoffer/catkin_ws/build"
####
####
#### Running command: "make -j4 -l4" in "/home/kristoffer/catkin_ws/build"
####
[  0%] Built target sensor_msgs_generate_messages_lisp
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target _custom_srvs_generate_messages_check_deps_image_to_image
[  0%] Built target _custom_srvs_generate_messages_check_deps_two_int
[  0%] Built target _custom_srvs_generate_messages_check_deps_custom_service
[  0%] Built target sensor_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_py
[  0%] Built target sensor_msgs_generate_messages_nodejs
[  0%] Built target std_msgs_generate_messages_nodejs
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target sensor_msgs_generate_messages_eus
[  0%] Built target _custom_srvs_generate_messages_check_deps_image
[  0%] Built target sensor_msgs_generate_messages_cpp
[  0%] Built target std_msgs_generate_messages_cpp
[ 22%] Built target custom_srvs_generate_messages_py
[ 40%] Built target custom_srvs_generate_messages_lisp
[ 59%] Built target custom_srvs_generate_messages_nodejs
[ 81%] Built target custom_srvs_generate_messages_eus
[100%] Built target custom_srvs_generate_messages_cpp
[100%] Built target custom_srvs_generate_messages
(base) kristoffer@HP:~/catkin_ws$ _
```

Figure 3.4: The sweet sight of a successful build

The functionality of the camera_service is to send either a snapshot from the computers web camera or get a local file from a given path and send either as an image. To do this any images must be converted from cv2 images into images which ROS can understand, this is done by using the CvBridge() module and the cv2_to_imgmsg function, which takes two arguments, the image and the encoding. Similarly the function imgmsg_to_cv2 transforms an image the other way.

To use the actual server functionality of ROS services, a server was called using the rospy.Service() followed by a rospy.loginfo("Server is running") to indicate that the server is running seen from the terminal and a rospy.spin() is added to keep the server running in standby mode until it is called by a client. A code snippet is added for convenience in figure 3.5

```
46
47          rospy.init_node('camera_service_node', anonymous=True)
48          self.srv = rospy.Service("/camera_service",image,self.callback)
49          rospy.loginfo('Service is running')
50          rospy.spin()
51
```

Figure 3.5: Initializing the camera_service

The next part was to create a new package called edged_image_processing. This package will contain the part of the code that creates the edge detected image with threshold. Similarly to the earlier packages it has to get its dependencies edited and then be built by catkin_make. Inside the auto generated src folder the new script was added. The new python script was equipped with a service client which calls the camera_service created earlier, to get the image from either the camera or the file system, depending on the request delivered in the client. For this case the string passed into the service call is "file", which will choose to get the image from file and not the camera by the use of an if statement. The service call can be seen in figure 3.6

```
63
64      def main(self):
65          print('inside main edged image processing service')
66          self.path = '/home/kristoffer/catkin_ws/src/Thesis/pipe_images/'
67          self.savefile = 'saved_canny_avg.png'
68          self.bridge = CvBridge()
69
70          print "Waiting for camera service"
71          rospy.wait_for_service("/camera_service")
72          try:
73              info = rospy.ServiceProxy("/camera_service",image)
74              response = info("file")
75              print (response.encoding)
76              self.imgur = self.bridge.imgmsg_to_cv2(response,desired_encoding="passthrough")
77          except rospy.ServiceException, e:
78              print "Service call failed: %s"%e
79
```

Figure 3.6: Calling the camera_service from python

To get a better impression on how the ROS services work, two more figures are included below, figure 3.7 is the initialization of the edged_image_processing_service, and figure 3.8 is how the response is stored to the service from inside the callback function.

```
74
75        rospy.init_node('edged_image_processing_node', anonymous=True)
76        # Sending result to callback
77        self.srv = rospy.Service("/edges_image_processing_service",image,self.callback)
78        rospy.loginfo('Edges Service is running')
79        rospy.spin()
80
```

Figure 3.7: Initializing the edged_image_processing_service from python

```
51
52            img = self.bridge.cv2_to_imgmsg(fin_img,"mono8")
53            print('Sending image...')
54            response = imageResponse(img.width,img.height,img.is_bigendian,img.encoding,img.data)
55            print ('Image sent')
56
57            return response
```

Figure 3.8: Saving the response to the edged_image_processing_service from python

The next step was to crop the image as described earlier in section 3.2. To manage this, the new crop_image package was created together with the script crop_image.py. This script requires two images as input, the original image from the camera_service and the edged image from the edge_image_processing_service, both are sent via the image.srv service message. The script creates three services with each a relevant output. First, the cropped edged image, then the cropped original image. Until now all service messages have been of the custom image.srv. For the last message a second custom message was created, the "two_int.srv" which takes an empty request and responds with two integers, the pixel coordinates from the neck. This is needed to transform the potential findings back to the image coordinates of the original image, and further transform the image transformation to 3D coordinates based on this.

At this point it became very apparent why roslaunch exists, without the roslaunch method each service server and service client had to be run individually in separate terminals. To allow for the use of roslaunch, each package needs to have another directory called "launch" created inside. Then a .launch file has to be created inside this directory. In the launch file it has to be specified the parent package name, the name of the python script, in this case these scripts contain both service clients and service servers, the name of the node and the output source, usually "screen" in this case. When all packages contain a launch file, these files can then be called from inside another launch file. This is done by using the include file functionality, and thus allowing the launch file to access the other launch files outside the package. A print screen of a launch file is shown in figure 3.9.

```
<launch>
    <node
    pkg = "feature_detector"
    type = "feature_detection.py"
    name = "feature_detector_node"
    output = "screen">
    </node>

    <include
    file="/home/kristoffer/catkin_ws/src/Thesis/camera_service/launch/camera_service_launch.launch">
    </include>
    <include
    file="/home/kristoffer/catkin_ws/src/Thesis/edged_image_processing/launch/edged_image_processing.launch">
    </include>
    <include
    file="/home/kristoffer/catkin_ws/src/Thesis/crop_image/launch/crop_image.launch">
    </include>
</launch>

```

Figure 3.9: Example of a launch file

The next package that had to be created was the feature detection package, the dependencies and the building is done in a similar matter to the earlier packages. Inside this package the python script ”feature_detection.py” was created, and inside a node called ”feature_detector_node”. This script has both the cropped edged image and the cropped original image as input as together with as the image coordinates from the neck location discovered earlier. The edged cropped image is the image which is used for the actual feature detection, and the cropped original image is only used to display the results. As mentioned earlier the neck image coordinates is used for transform the locations discovered in the cropped image back to the pixel coordinates of the original image. The output for this function is the image displaying the located weld starts and stops as well as the pixel coordinates of the weld start and stops relative to the original image. Furthermore the following step is to create a package for the diameter estimation. Due to time limitations this is as far as the creation of ROS services got for this report.

Chapter 4

Result

4.1 PipeFlow database

From the inspection, the following information in PipeFlow is relevant for the welding program:

- Tong space
  - The distance between hardbanding and the seal face, the available space for gripping when connecting drill-pipes.
- Tool joint outer diameter
- Pipe wall thickness
- Overall pipe length

From discussion with Joakim Linge at Goodtech it became clear that the server is of the type relational database, and can be connected to python using a SQL-library for python called pyodbc.

4.2 Detect starts and stops in 3D coordinates

The original input image taken from Autodesk Inventor environment:

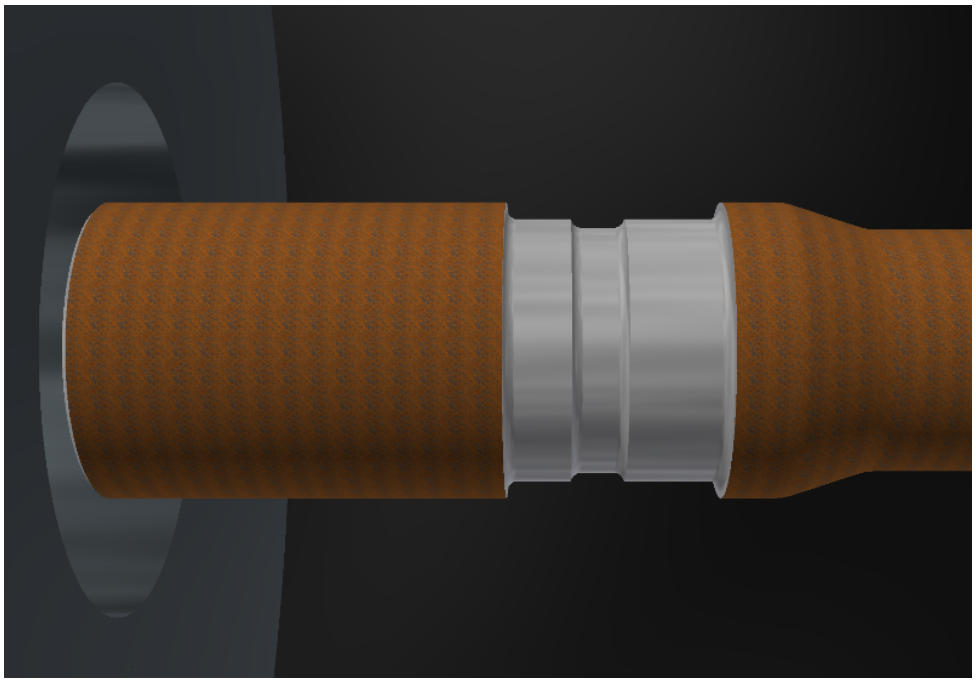


Figure 4.1: Drill-pipe image input

After the original image has been distributed the improved canny edge detector finds the edges deemed relevant.

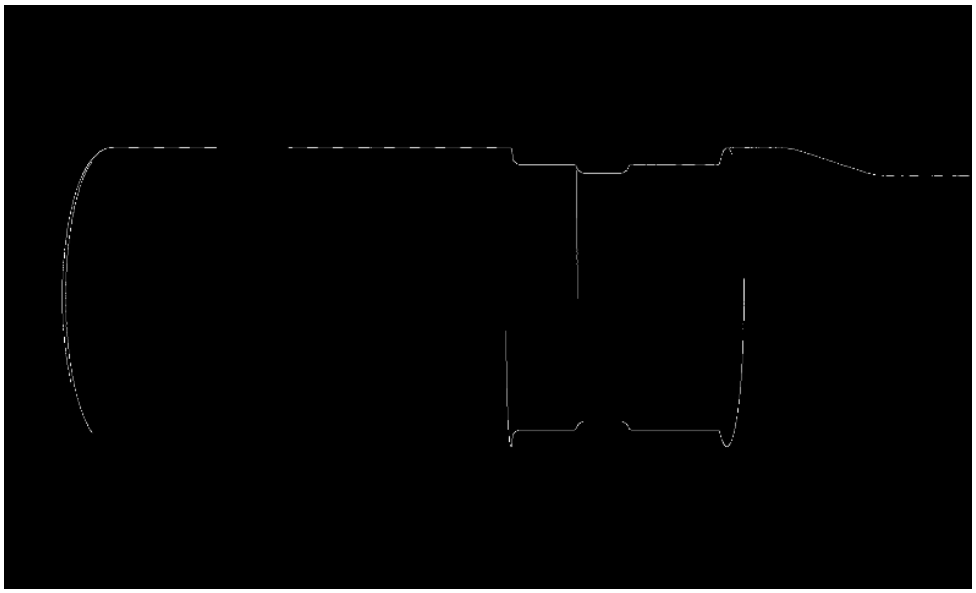


Figure 4.2: Drill-pipe edges

The edged image is then searched for the tool joint diameter and it detects the highest horizontal line within the given constraints.

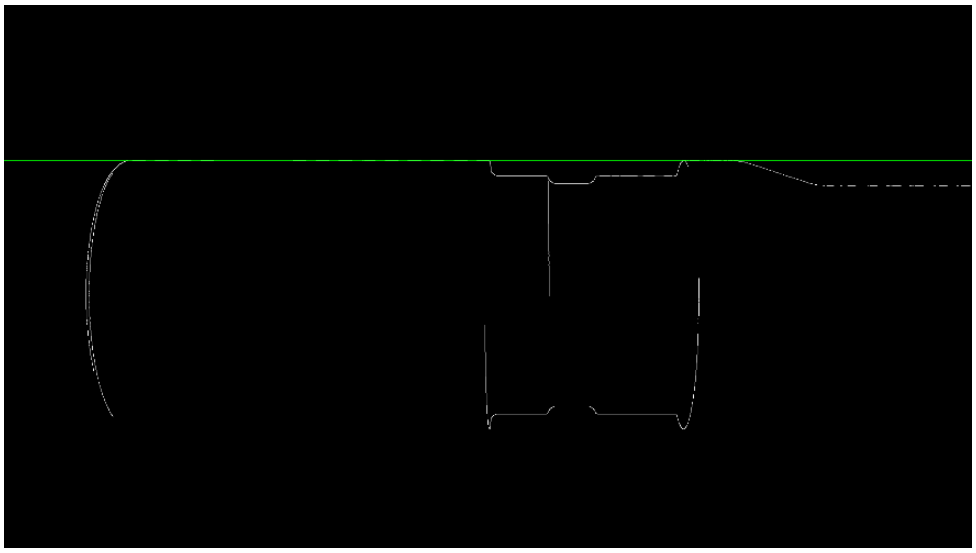


Figure 4.3: Drill-pipe tool joint line

The neck is then found using the highest line within the desired angle interval matching the neck constraints.

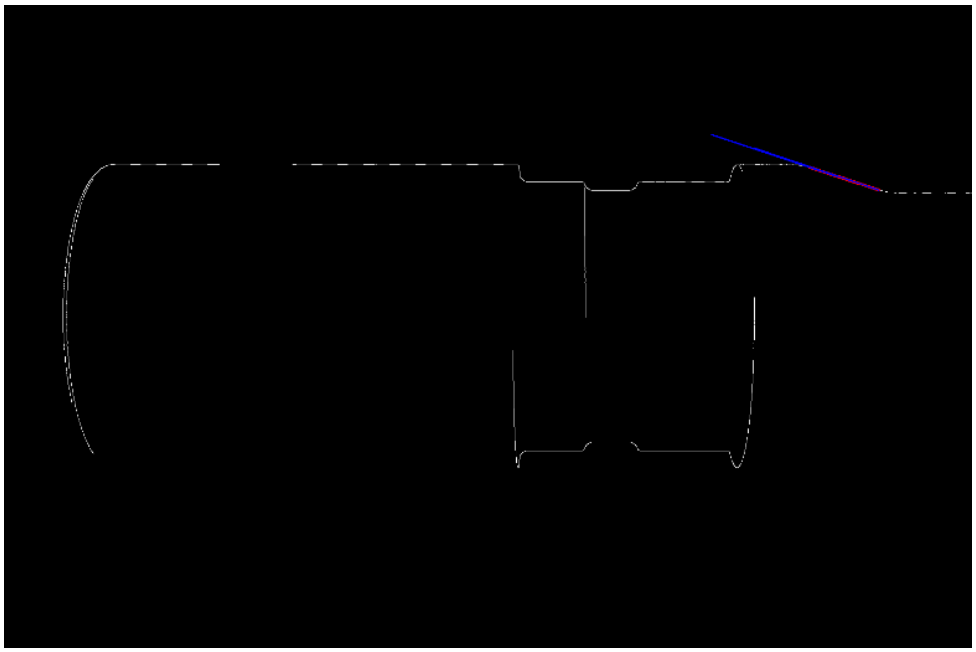


Figure 4.4: Drill-pipe neck detection

Based on the intersection of the tool joint line and the neck line the crop the image is created to avoid searching in non-relevant places.

Figure 4.5: Drill-pipe image Region Of Interest

From the former figure 4.5 the features within the region of interest is found, and then displayed on the cropped color image as seen in figure 4.6. Below is the detected features of five different test pipes, with the locations of the weld starts and weld stops.

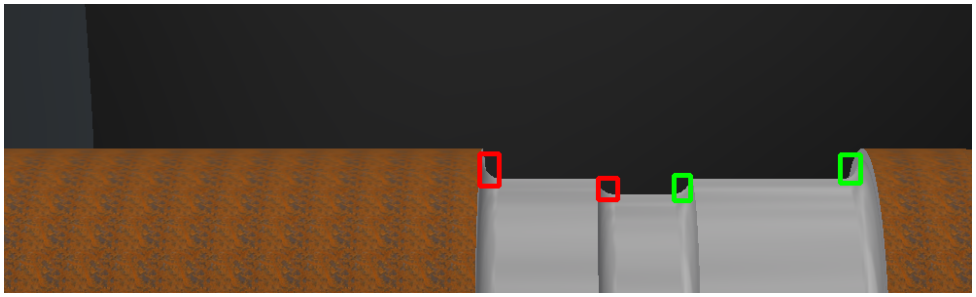


Figure 4.6: Drill-pipe 1 Detected weld start and stops for multiple grooves

Drill-pipe 1	X	Y	Z
Weld start 1	0 mm	324.7 mm	49.9 mm
Weld stop 1	0 mm	307.9 mm	49.3 mm
Weld start 2	0 mm	361.5 mm	53.9 mm
Weld stop 2	0 mm	281.6 mm	54.1 mm

Table 4.1: Weld locations drill-pipe 1

Figure 4.7: Drill-pipe 2. Detected weld start and stop for a single groove

Drill-pipe 2	X	Y	Z
Weld start 1	0 mm	307.6 mm	50.4 mm
Weld stop 1	0 mm	227.7 mm	51.4 mm

Table 4.2: Weld locations drill-pipe 2

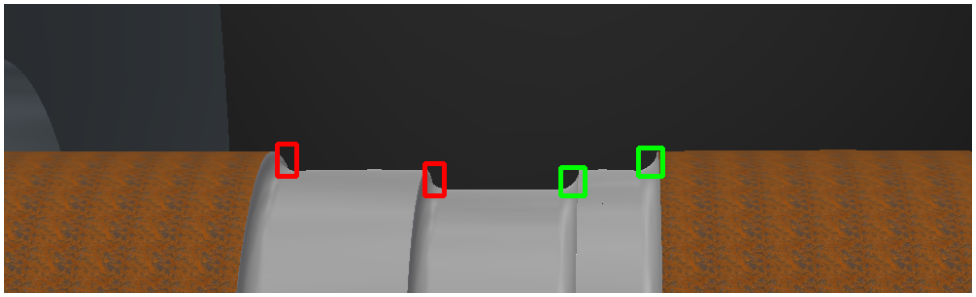


Figure 4.8: Drill-pipe 3. Detected weld start and stops for multiple grooves

Drill-pipe 3	X	Y	Z
Weld start 1	0 mm	290.0 mm	75.8 mm
Weld stop 1	0 mm	259.8 mm	76.5 mm
Weld start 2	0 mm	306.6 mm	79.6 mm
Weld stop 2	0 mm	227.1 mm	80.4 mm

Table 4.3: Weld locations drill-pipe 3

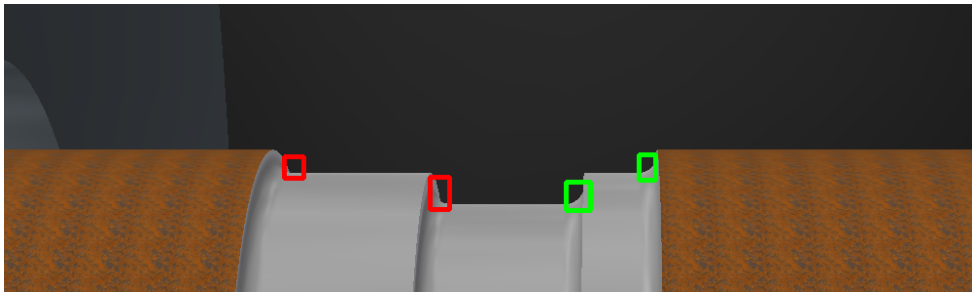


Figure 4.9: Drill-pipe 4. Detected weld start and stops for multiple grooves

Drill-pipe 4	X	Y	Z
Weld start 1	0 mm	291.3 mm	72.9 mm
Weld stop 1	0 mm	261.1 mm	73.8 mm
Weld start 2	0 mm	307.4 mm	78.2 mm
Weld stop 2	0 mm	228.9 mm	77.9 mm

Table 4.4: Weld locations drill-pipe 4

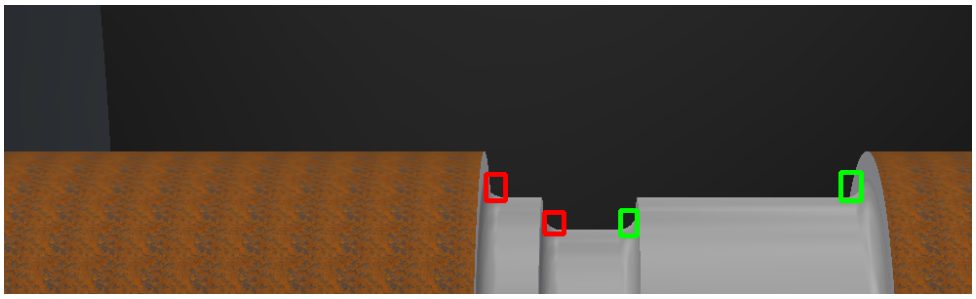


Figure 4.10: Drill-pipe 5. Detected weld start and stops for multiple grooves

Drill-pipe 4	X	Y	Z
Weld start 1	0 mm	308.1 mm	54.3 mm
Weld stop 1	0 mm	291.3 mm	53.9 mm
Weld start 2	0 mm	356.8 mm	61.8 mm
Weld stop 2	0 mm	278.4 mm	61.4 mm

Table 4.5: Weld locations drill-pipe 5

Where the coordinate system referred to is the same as in the former report (Hermansen, 2018), having the origo just in the center of the clamping chuck, the X-axis orthogonal of the image plane, the Y-axis along the drill-pipe, and the Z-axis is up. The complete code can be seen in the appendix as "Main code", this also include the diameter estimation and weld planner.

4.3 Diameter estimation

Using the data from the feature detection, the diameter is found by setting constraints eliminating lines in between any of the start and stops, searching for the highest lines. Results are given in figure 4.11 to 4.15:

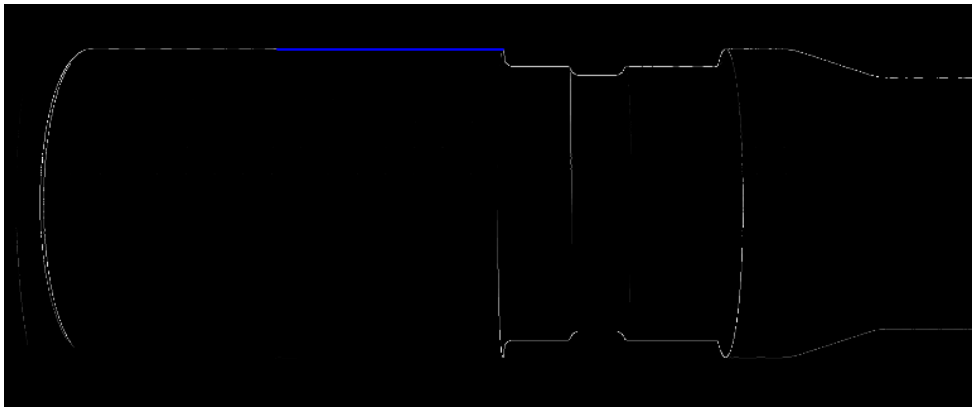


Figure 4.11: Drill-pipe 1. Estimation tool joint diameter

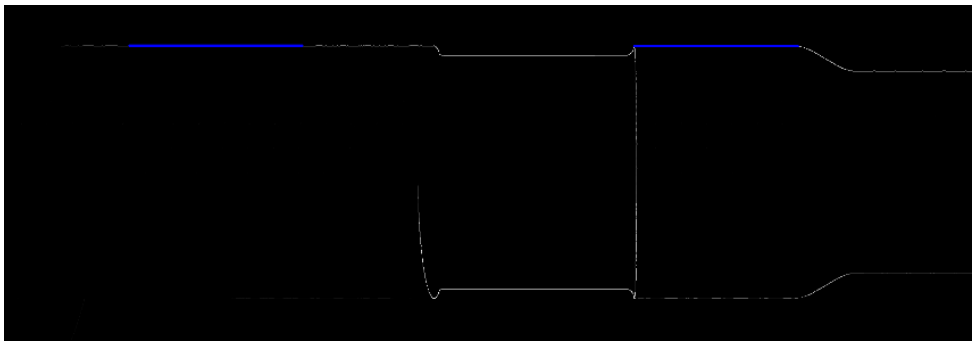


Figure 4.12: Drill-pipe 2. Estimation tool joint diameter

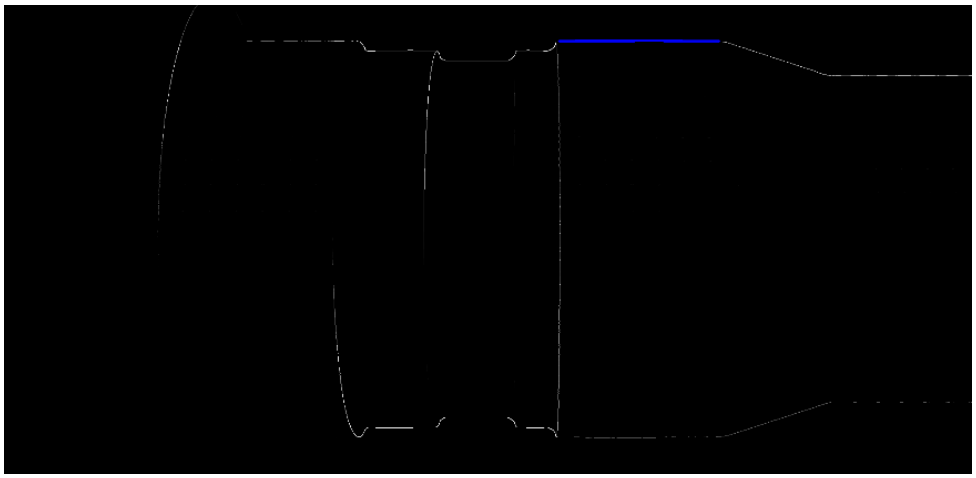


Figure 4.13: Drill-pipe 3. Estimation tool joint diameter

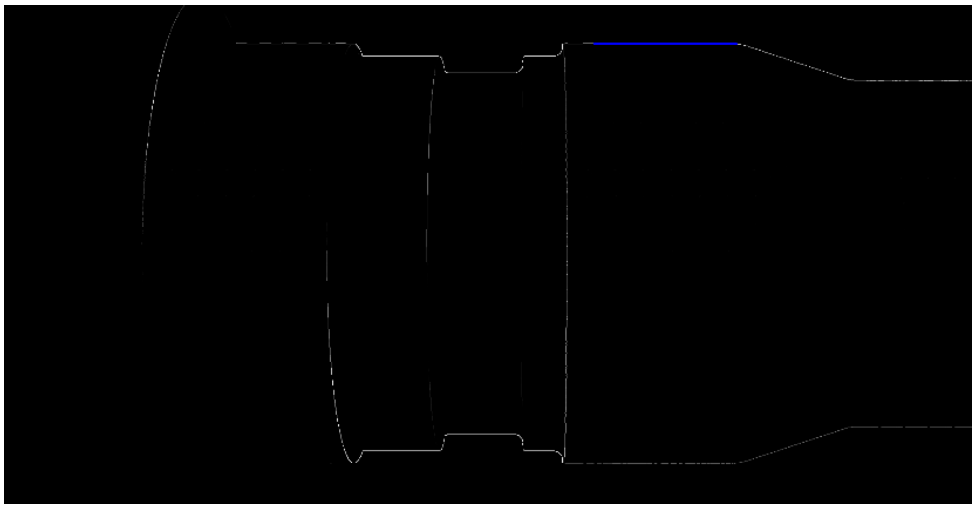


Figure 4.14: Drill-pipe 4. Estimation tool joint diameter

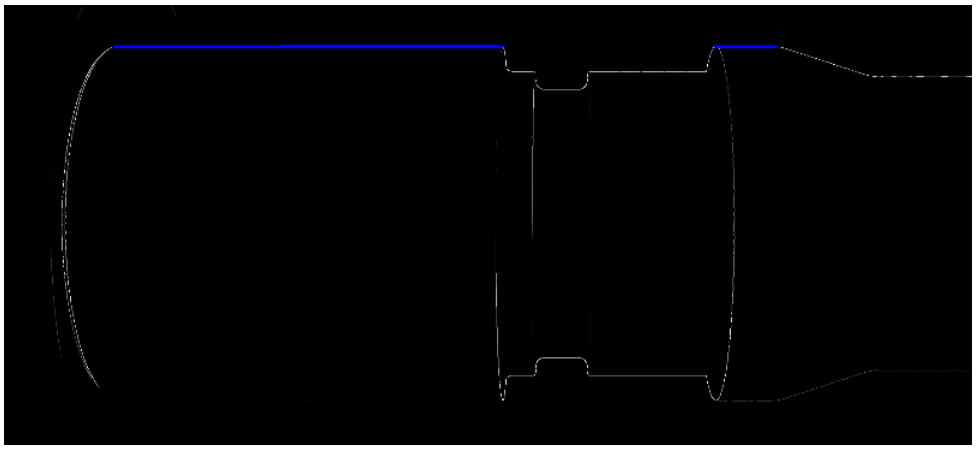


Figure 4.15: Drill-pipe 5. Estimation tool joint diameter

When estimating the diameter at the groove, all lines outside the final weld start and weld ends are ignored. The "groove 1" line is found by looking for lines lower than the tool joint but higher than the deepest weld start and stop, including potential offsets to compensate for the height of the template image. There are also constraints on the length of the detected line, results are given in figure 4.16 to figure 4.20:

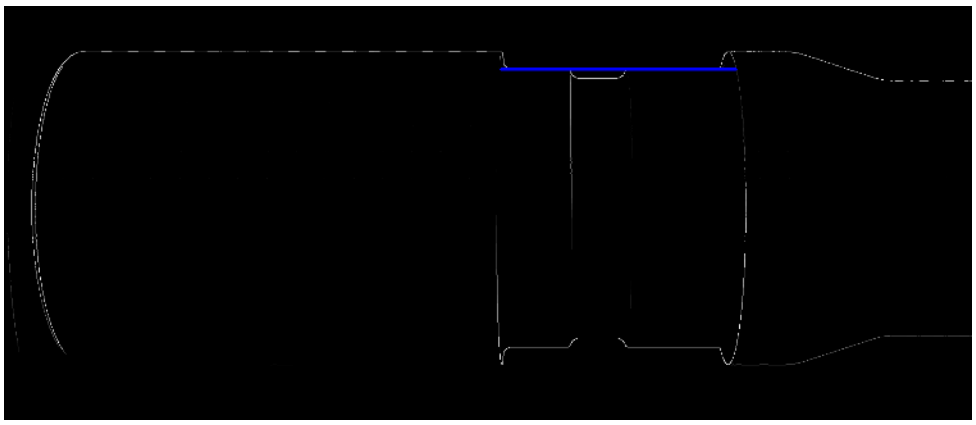


Figure 4.16: Drill-pipe 1. Estimation groove 1 diameter

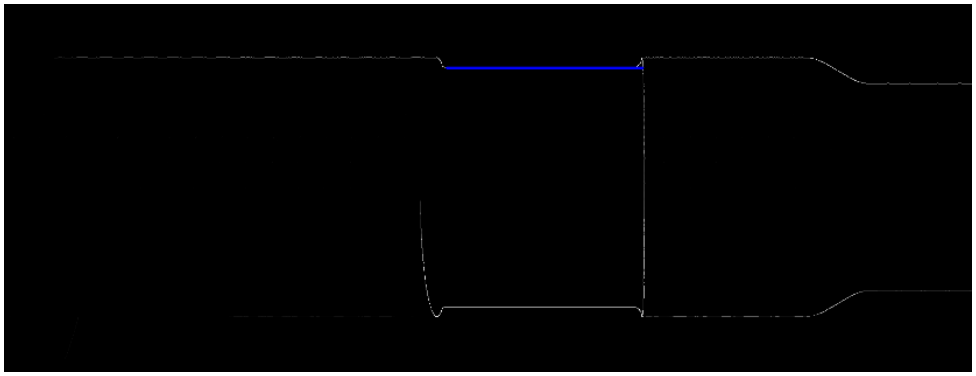


Figure 4.17: Drill-pipe 2. Estimation groove 1 diameter

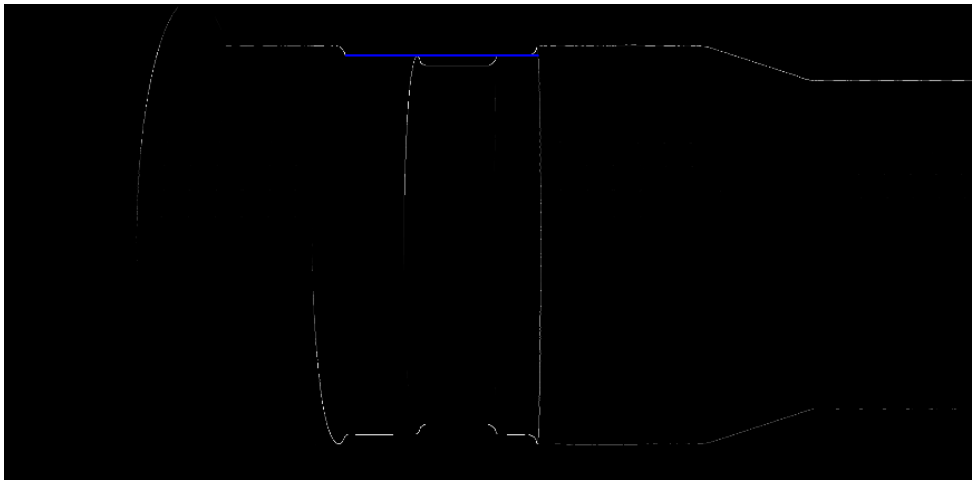


Figure 4.18: Drill-pipe 3. Estimation groove 1 diameter

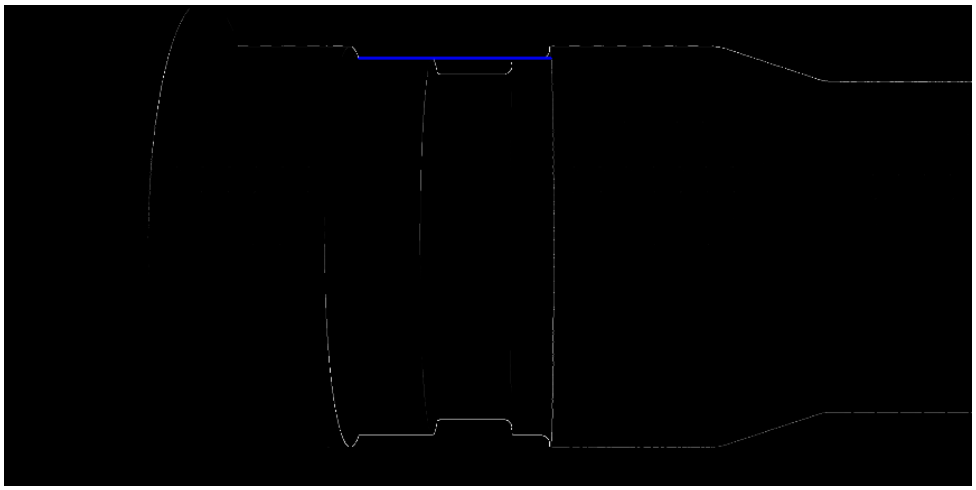


Figure 4.19: Drill-pipe 4. Estimation groove 1 diameter

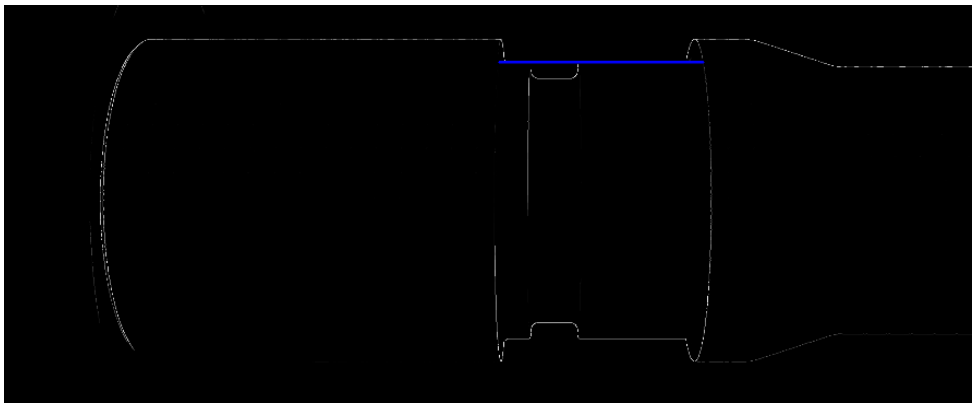


Figure 4.20: Drill-pipe 5. Estimation groove 1 diameter

The final groove diameter is estimated by the line detected in figures 4.21 to 4.24. Here the former lines are used to eliminate false lines and the line needs to be located between the first weld start and end.

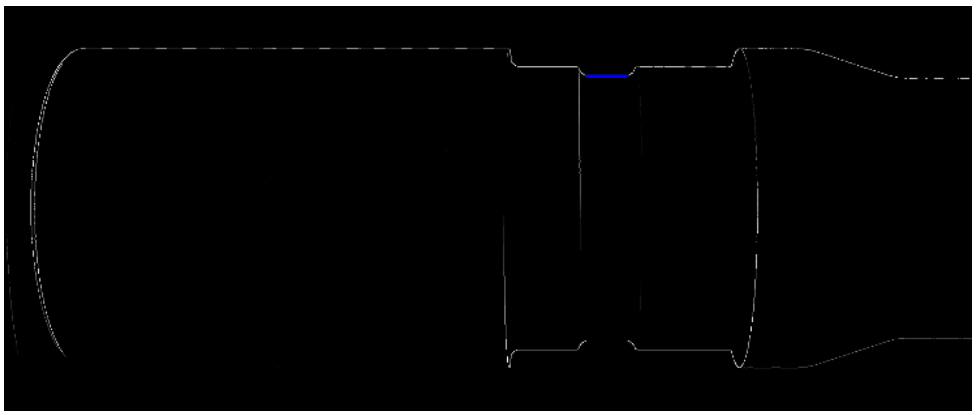


Figure 4.21: Drill-pipe 1. Estimation groove 2 diameter

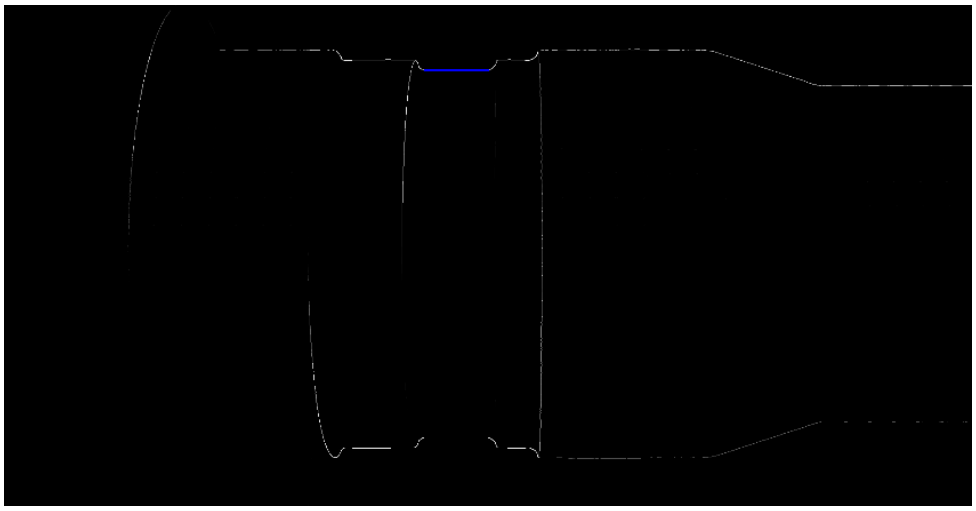


Figure 4.22: Drill-pipe 3. Estimation groove 2 diameter

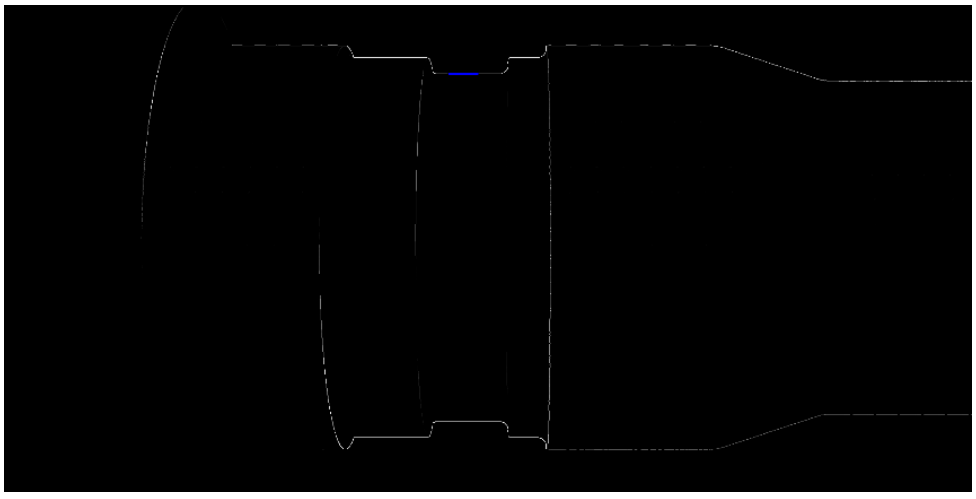


Figure 4.23: Drill-pipe 4. Estimation groove 2 diameter

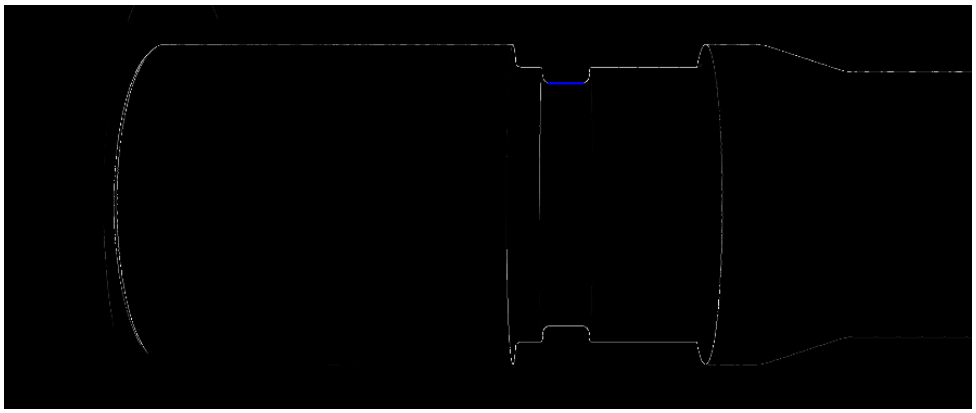


Figure 4.24: Drill-pipe 5. Estimation groove 2 diameter

Drill-pipe	Tool Joint diameter	Groove 1 diameter	Groove 2 diameter
Drill-pipe 1	110.1 mm	98.6 mm	92.4 mm
Drill-pipe 2	100,5 mm	93.2 mm	NA
Drill-pipe 3	158.0 mm	150.7 mm	143.1 mm
Drill-pipe 4	158.4 mm	149.6 mm	137.3 mm
Drill-pipe 5	131.9 mm	113.9 mm	101.3 mm

Table 4.6: Weld locations drill-pipe 3

4.4 Weld planning

The results from the weld planner has been summarized in the following tables. Note that the pipe rotational speed is highly dependent on the dimensions of the pipe, meaning that a weld height of 3mm will result in a very different pipe rotational speed for a 4" drill-pipe and a 6-5/8" drill-pipe.

Drill-pipe 1								
Groove	Strings	Layers	1st layer height	1st Pipe rpm	2nd layer height	2nd Pipe rpm	Effective weld time	Meters of weld rod
Groove 2	2	1	3.06 mm	3.01 rpm	NA	NA	0.6 min	5.4 m
Groove 1	11	2	2.88 mm	3.12 rpm	2.88 mm	2.81 rpm	7.4 min	68.2 m

Table 4.7: Weld program. Drill-pipe 1

Drill-pipe 2								
Groove	Strings	Layers	1st layer height	1st Pipe rpm	2nd layer height	2nd Pipe rpm	Effective weld time	Meters of weld rod
Groove 2	NA	NA	NA	NA	NA	NA	NA	NA
Groove 1	11	2	3.64 mm	2.56 rpm	NA	NA	4.3 min	39.2 m

Table 4.8: Weld program. Drill-pipe 2

Drill-pipe 3								
Groove	Strings	Layers	1st layer height	1st Pipe rpm	2nd layer height	2nd Pipe rpm	Effective weld time	Meters of weld rod
Groove 2	4	1	3.84 mm	1.62 rpm	NA	NA	2.5 min	22.7 m
Groove 1	11	2	3.64 mm	1.63 rpm	NA mm	NA	6.8 min	61.4 m

Table 4.9: Weld program. Drill-pipe 3

Drill-pipe 4								
Groove	Strings	Layers	1st layer height	1st Pipe rpm	2nd layer height	2nd Pipe rpm	Effective weld time	Meters of weld rod
Groove 2	4	2	3.07 mm	2.13 rpm	3.07 mm	1.96	3.9 min	35.5 m
Groove 1	11	1	4.41 mm	1.34 rpm	NA mm	NA	8.2 min	74.6 m

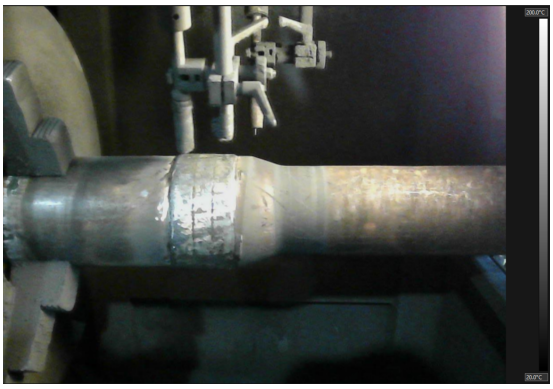
Table 4.10: Weld program. Drill-pipe 4

Drill-pipe 5								
Groove	Strings	Layers	1st layer height	1st Pipe rpm	2nd layer height	2nd Pipe rpm	Effective weld time	Meters of weld rod
Groove 2	2	2	3.16 mm	2.75 rpm	2.75 mm	2.46 rpm	1.5 min	13.7 m
Groove 1	11	2	4.51 mm	1.69 rpm	4.51 mm	1.47 rpm	14.0 min	127.4 m

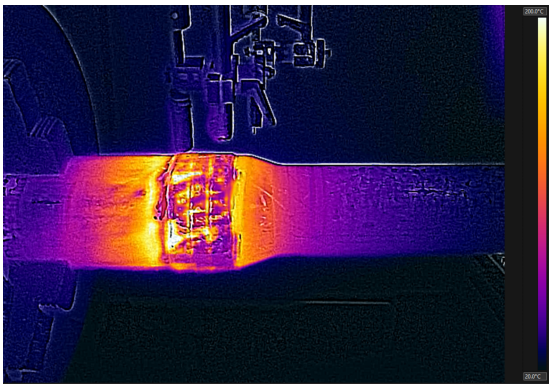
Table 4.11: Weld program. Drill-pipe 5

4.5 Thermal measurements

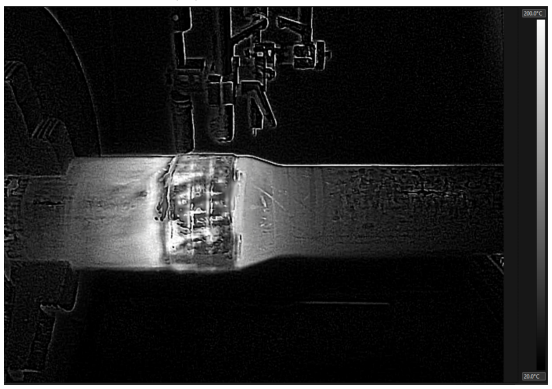
The following image has been taken of a drill-pipe just after it has been welded with new hardbanding.



(a) Normal image



(b) Thermal and normal image



(c) Grey thermal and normal image

(d) Grey-scale thermal image

Collecting thermal data from the images: Input image was an image where the scale was set to 20-200 °C from the app. The image can be seen in figure 4.26.

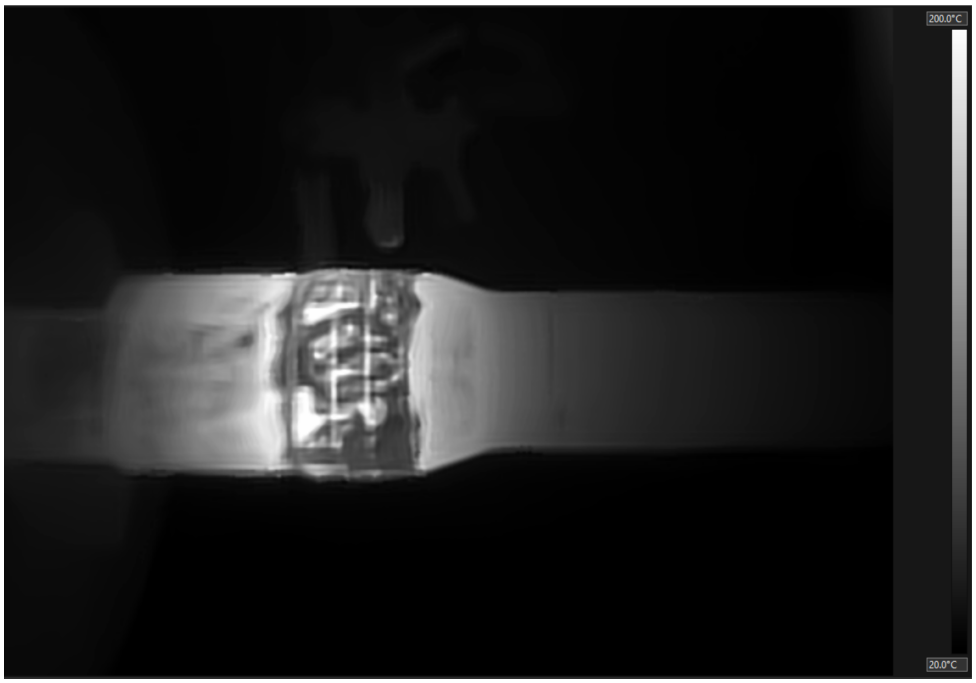


Figure 4.26: Thermal image scaled between 20-200 degrees Celsius

The thermal image was further scaled to make the pixel values read the measured values, the result can be seen in figure 4.27

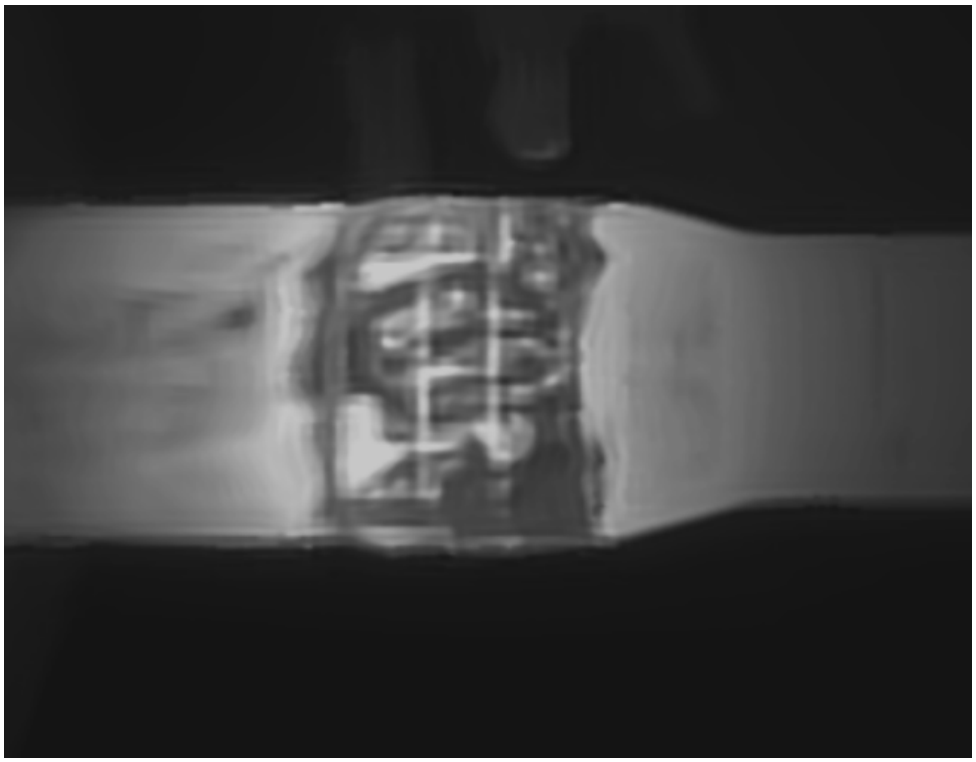


Figure 4.27: Thermal image scaled for pixel value to match degrees Celsius

The program then allows for regions of interests which can be specified, here the region focused on is the hardbanding area of the tool joint.

Figure 4.28: Closeup on the region of interest

When any area of the scale thermal image from figure 4.29 is in close focused, it is easy to see that the pixel values correspond to the measured temperature of the pixel area.

135	146	154	162	164	161	154	141	124	104	90	78	70	68	68	70	74	77	80	82
144	156	164	172	172	168	160	147	128	107	92	80	72	68	68	71	75	77	80	82
150	162	170	176	176	171	162	148	128	107	93	82	73	70	70	72	75	78	82	84
153	165	172	178	177	172	163	147	128	106	94	82	75	72	72	74	77	81	84	86
148	159	166	172	171	166	157	144	126	107	96	87	81	79	80	82	86	89	93	95
141	152	158	164	163	159	151	139	124	108	99	92	89	89	91	94	99	104	106	109
132	141	147	152	152	149	142	133	122	109	104	101	100	102	106	111	118	123	126	127
123	132	137	141	141	139	135	128	120	111	107	106	106	110	115	121	128	135	137	139
118	125	130	133	134	133	130	126	119	112	109	108	111	114	120	126	134	140	143	144
117	124	128	132	132	132	131	128	122	116	114	113	114	118	124	130	137	144	147	147
120	126	131	135	137	137	137	134	129	123	121	120	122	125	130	137	143	148	151	152
127	133	140	144	147	149	149	148	143	137	137	136	137	140	144	149	156	161	162	163
136	144	152	159	163	166	168	168	166	162	161	159	160	163	167	173	178	183	184	184

Figure 4.29: Further closeup of the pixel values which represent temperature in Celsius

Finally the program finds the max temperature inside the region of interest and marks it with a small blue square. The average temperature within the red region of interest is also calculated, as shown on the image.

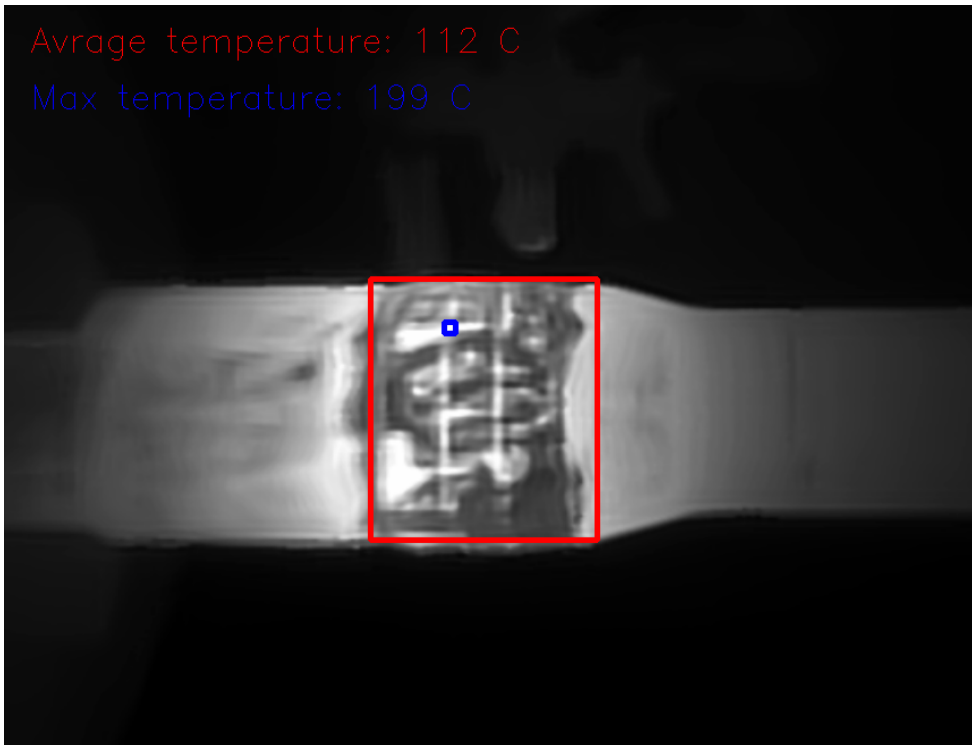


Figure 4.30: Generated temperature results using computer vision with thermal image

4.6 ROS

The results for the ROS implementation can be seen in the appendix

Chapter 5

Discussion

5.1 PipeFlow database

All the pipes which are to be welded, are first cleaned and inspected. During inspection, all results are reported and automatically stored in a database for customers to see. When the pipes are inserted into the existing production-line, the pipes are loaded in while the operator assigns the pipe identity to the pipe, so that relevant information is available. Since this is already being done in other parts of the existing production, it was not highly prioritized to spend time on this during the investigation of the PipeFlow database in this report.

A lot of time was spent learning about the different types of databases and their differences before Joakim Linge from Goodtech was contacted. The conversation with Joakim was very informative and rendered some of the prior work unnecessary in this report.

The research done to see whether there are any information which might be used in the feature detection program as well as the later parts of this process was useful. From the inspection, there are several measured dimensions which might be of interest these are all mentioned in the results. These measured dimensions could work as additional input in the other programs such as the diameter estimator, where one could use the measured diameter together with the tool joint diameter derived from computer vision to validate the computer vision result.

When the PipeFlow database is to be implemented in the future, it would be possible to implement with all existing python code via the pyodbc library. The PipeFlow database is a relational database which uses SQL to interact. In order to interact with relational databases using SQL, the python module pyodbc can be used to query the database with SQL from python. During the research on the PipeFlow database, it was created an additional user with read access which is available to be used to read from the database. This additional user was not tested during this report due to time limitations.

5.2 Detect starts and stops in 3D coordinates

After some work on the new weld parameter code, it became very apparent that the complexity of the detection software from the (Hermansen, 2018) needed to be upgraded. The code was not able to conclude how many grooves was present, and the detected weld start and stop was an average of the detected features at the set threshold. Meaning that several grooves or miss readings made the detection software give wrong readings.

The challenge was to keep the important contours which describes the tool joint and the machined groves, while discarding textures and other misreadings. To do this the canny edge detection algorithm was used in a loop where the tuning parameters where variables within some specified set points. All the images from the loops was then combined to an average, finally thresholding was applied to the averaged image to filter out the lower pixel values as not edges. Further there was some challenges with the number of features detected, as long as all detected features where in close proximity of the actual feature, it could easily be averaged. However, one misreadings could offset the average enough to completely miss the target. One of the root causes for this problem was the fact that the searching algorithm used to discover the weld start and stop depended on comparing an image of the start or stop to all areas on the edged image. Whether the start or stop was detected depended only on how similar the pixels in the search area are to the detection input. This was given with another threshold, a number between 0.00 and 1.00, where 0.00 would approve anything as a feature and 1.00 would demand an perfect similarity to approve a feature.

This was remedied with the use of a simple algorithm which searched for the feature in the image with a decreasing threshold value, starting at a perfect match at 1.00 and decreasing the value until a feature was detected. Still there where some issues where the program would detect the start just fine, and miss on the stop. To avoid most of these issues a constraint was added to the program where the height difference between the start and stop was set to a maximum, where larger differences in height made the software ignore any features found and continue the search until a feature within he allowed height difference was detected.

The next issue needed to be solved was that all weld starts and weld stops does not necessarily look alike, different locations compared to the camera location yield different results. Therefore a new functionality had to be implemented. Two new directories was created, one containing several images of different weld starts, and the other containing several weld stops. The search algorithm then completed all steps above within a set threshold, first, weld start image one was used, if any features was detected, the algorithm took the next weld stop image and searched for a feature which had all requirements from above, if no weld features was approved, then the algorithm takes the next weld start image and the loop continues. The break conditions is either by fulfilling feature requirements or by not detecting any features fulfilling all requirements.

The latest addition has the potential to greatly increase the run-time of the program. Since the images used was generated in Autodesk Inventor had a large pixel size (3017x1704), the image needed to be cropped to reduce run-time. The cropping of the image was done by first locating the neck of the pipe, this gave a logical input to where it was logical to look for features, anything behind the neck of the pipe it irrelevant. The neck was found

by looking for places where continuous lines had an angle between 15 and 25 degrees, then looking for horizontal lines. Finding the highest intersection between the angled line and the horizontal lines gave the neck location. From the neck location, a region of interest could be decided, the highest horizontal line gave the tool joint diameter, and any point higher than this would not be logical to search through. Further the pixel location of the inner diameter of the tool joint can be calculated, and any points lower than this can also be ignored. Further the length of the tool joint can be collected and any points to the left of this is ignored. The sum of the tool joint diameter, neck location, tool joint inner diameter and tool joint length gives a logical area to search. This greatly reduced the run-time. Further some translations or offset was however needed to reconnect the location of the region of interest to the image coordinates which then is connected to camera the location.

5.3 Diameter estimation

The diameter detects the wanted diameter on most occasions. It is dependent on the work of the feature detector to be successful, since it uses the weld start and stop locations to categorize the approved lines. Based on the length and placement of the lines, it is logically determined if the line is of interest and further which line it should represent. The program deliver consistent results, even though there will be an increasing error with larger pipe diameters as the curvature seen by the camera looking at the center of pipe. It was initially attempted to tune the camera settings for inventor such that actual dimensions of the 3D model could work as a reference to which the result could be tested. However insufficient calibration made this difficult.

From the former paper (Hermansen, 2018) a precisely 3D modelled chessboard was created in Autodesk Inventor, and then with the same camera setting as the pipe images several images was taken from different angles, and these were used in a camera tuning program to create the intrinsic camera matrix. This camera tuning camera was lightly modified from (Lars Tingelstad, 2018). The K-matrix did not yield satisfactory camera tuning, and therefore it did not make sense to compare the derived result to the dimensions in the 3D model. This might be problems related to the images collected from Autodesk Inventor. The use of the PipeFlow database could assist the estimator by accurately giving the tool joint diameter which was measured in the inspection earlier. This could help calibrate the image to achieve satisfactory results.

Given more time and improvement of the precision, the use of computer vision to measure parts might rapidly speed up the inspection time by having a camera taking an image of the drill-pipes which are being loaded into the inspection area. An approach might be using this computer vision in parallel with the existing inspection and using the traditional measurements as training data to a machine learning program.

5.4 Weld planning

The weld planner seem to deliver both reliable and consistent results. The rectangular model might be a poor simplification for one single weld string. Meanwhile, when welding several strings after one another, the finished result can appear to be rather straight with correct settings and therefore producing a better model for multiple strings, this can be seen from figure 3.1. As one would expect, it is possible to change this model for a more precise model in the future, or to tune the result from this model based on experience, which might give a desired result. It is assumed that the welding rod is solid, this could be adjusted with an experimental coefficient

Moreover the benefits of having a clear and specified weld program allows for more consistent results. This provides a more traceable weld, making it possible to improve the weld results in the long run by analysing the pipes which come back for later repairs within the pipes lifetime, and thereby improving the weld program to correct any apparent errors discovered.

Another benefit from the use of this welding program is the potential savings by having the consistent weld height. A good weld height, meaning just a little higher than level with the rest of the tool joint. When welding in the existing production line it is difficult to get the height just right, and therefore, the safest bet is to ensure that the weld height surpass the tool joint height, so than one does not have to add jet another layer of weld. Any excessive height is then grinded back to be level with the tool joint. By using computer vision to derive the needed weld height and a mathematical model adjusted from experience, the amount of grinding could be reduced to a minimum. The weld time and applied heat is also going to be affected by a well functional weld planner.

5.5 Thermal measurements

The use of thermal images as a thermal sensor is an intriguing idea for a setup containing several pipes. This is especially so with the geometrical differences on the drill-pipes, such as different diameters, different tool joint lengths and the subsequent different groove locations. This means that the area needed to be measured changes from pipe to pipe. Using conventional sensors would either depend on a great number of sensors to cover all relevant areas, or the thermal sensor needs to be able to move from pipe to pipe.

When using a thermal camera as a thermal sensor, the optical resolution gives the number of optical pixels available to read. For the consumer product which where used in conjunction with this report, this results in 160x120 pixels, which results in 19200 measurements. In order to be able to use the thermal camera as a sensor, it is essential to be able to recognize the parts of the image to the objects of interest.

Unfortunately a lot of time was spent trying to generate a computer vision program which could discover the relevant regions to be used together with the thermal image to extract temperature data. However, by using the existing data from the feature detection, one could simply calculate how this portrays from the original camera frame to the thermal camera frame. This would make it rather straight forward to extract the temperatures in the relevant region as seen in figure 4.30.

Some challenges with using the thermal camera is the different emissivity on the sections of the pipes as can be seen from figure 4.28. This result might come as a consequence of the welding slag creating less reflective surfaces in some places. The inconsistency of the reflective surfaces yields different measurements since the emissivity is set as a constant in the FLIR app for this device. For the result in figure 4.30 both the maximum and the average temperature is calculated. Actual measurements with a calibrated probe showed results just around 200 °C, indicating that the maximum temperature in the region gave the best result with the emissivity settings. A possible work around to this could be to use computer vision to estimate how reflective the different parts of the pipe by the "shine" seen in the regular image. Another possibility could be to use a better thermal camera, in which this might not be similar program due to internal software.

The cooling of the pipes is still an unresolved issue, which makes it difficult estimate the rate of cooling. Further the different rate of heating and cooling in the different pipe dimensions is rather complex to derive mathematically with the formulas given in equations 2.8, 2.7 and 2.6. The drill-pipes that are welded today all use internal air cooling to shorten the cool-down time and preserve the inner coating from burning. For the new system with one robot changing which drill-pipe to weld on to allow the other pipes to cool might be viable without internal cooling. Further data of a drill-pipe cooling down and analysing the data derived from the thermal camera is necessary to derive the number of drill-pipes which it is possible to weld without additional cooling.

5.6 ROS

The use of ROS together with robotics and computer vision from OpenCV is a powerful setup, making it possible to link the software to the hardware of any robotics system. In this report ROS has been vital to finding a way in which to implement the programs created to future hardware. In a simple way this has already been achieved in this report accessing the computers camera and sending the image over either a ROS topic of a ROS service followed by analysing the image.

The former work from (Hermansen, 2018) has been improved earlier in the report, and the work of transforming the program into ROS services has been implemented on all the steps including deriving the feature detection image coordinates result as well as a visual representation, see figure 4.6. Time limitations due to the complexity of learning to use ROS as well as the modifications needed to make programs "ROS-friendly" led to the fact that all parts of the programs created has not been implemented in ROS.

Future work on this system will allow more of ROS to be explored. For this report most of the parts of the code which has been turned into nodes and services has been of the kind that only needs to happen once, and then having the ability to access the derived results later. This means that both ROS topics and ROS actions has been left somewhat unexplored. The logical implementation to come will depend heavily on both topics and actions.

Topics will be useful for all sensor which needs to constantly update a value to give input which allows the welding robot and surrounding system to make decisions. These sensors might be the thermal camera monitoring the temperatures, the encoder counting the revolutions of the drill-pipe as it is being welded as well as the pose of the robot welder.

Actions will be crucial for any moving robotics as it allows for goals and operations to reach the goal to be canceled. This is especially useful for both the pipe rotation and the trajectory of the robot welder. Each action contains several topics, which allow you to send goals over a specific topic, it allows you to get some sensory feedback from the feedback topic and one gets the result from the robot telling whether or not the goal was successful.

Chapter 6

Conclusion

This report has continued the work completed in the former report by the same author (Hermansen, 2018). The report is part of a larger project conducted by WellConnection Mongstad to evaluate the possibility for creating a new automated production line for welding soft-lagging on drill-pipes.

The following is a summary of what has been achieved in this report:

- The retrieval of information from the PipeFlow database is realizable by the use of the python package pyodbc. Several of the measurements found in PipeFlow is relevant and should be integrated as part of the system.
- The improvements of the existing code from (Hermansen, 2018) was successful. The program can differentiate between pipes that has one or two grooves and give the 3D location of all weld starts and weld stops discovered.
- A program using computer vision to both detect the different diameters of the tool joint and grooves was created and the use of the transformation program converting pixel coordinates to 3D-coordinates returns the different diameters.
- A welding program to calculate the needed pipe rotational speed was created. This program uses the weld start and weld stop information together with the estimated diameter to ensure that the finished weld has the correct height. This should reduce the material cost linked to welding rod, it will reduce the welding time and heat applied to the pipe and it will decrease the time spent grinding excessive weld.
- The use of a thermal camera to measure the temperature of several pipes is both possible and desirable as the needed data to locate the individual pipes in the thermal image is available in this setup.
- Most of the upgraded python program has been implemented into ROS using ROS services. The use of ROS in this solution is considered by the author as a very good method to ensure that the code can be implemented with hardware as smoothly as possible.

Chapter 7

Future work

As this report is only a part of a larger project at WellConnection Mongstad, it is only natural to include how the results of this report has impacted the future work in the topics it concerns.

The following is a summary of the recommended continued work from the view of the author:

- Creating a python script to collect data from PipeFlow and have it available for the rest of the system via ROS.
- Create a database to collect all the images used for feature detection to generate training data to create an improved feature detection program in the future.
- Create a database for weld data generated from each drill-pipe to allow for the weld procedure to be evaluated when repaired drill-pipes come back for inspection or repair after use offshore.
- Further testing of the feature detection algorithm on real images.
- Continue the work of modifying the existing programs into ROS nodes

Bibliography

Thermodynamics. Mc Graw Hill Education, 2015.

Alberto Ezquerro. How to create a custom action message. webpage, 4 2019. URL http://www.theconstructsim.com/custom-action-message-ros-actions/. Last checked: 26.04.19.

AnisKoubaa. Services. wiki, 4 2019. URL http://wiki.ros.org/Services. Last checked: 23.04.19.

AustinHendrix. msg. wiki, 4 2019. URL http://wiki.ros.org/msg. Last checked: 21.04.19.

DirkThomas. Creating a ros msg and srv. wiki, 4 2019. URL http://wiki.ros.org/msg. Last checked: 21.04.19.

Olav Egeland. *A note on robot kinematics*. Unknown, 2018a.

Olav Egeland. *A note on vision*. unknown, 2018b.

FLIR. Flir one pro. Nettside, 5 2019. URL https://www.flir.com/products/flir-one-pro/?model=435-0007-02. Last checked: 28.05.19.

Kristoffer Hermansen. *TTK4551 Specialization Project, Using computer vision to control a robotic welder*. January 2018.

IsaacSaito. actionlib. wiki, 4 2019a. URL http://wiki.ros.org/actionlib. Last checked: 26.04.19.

IsaacSaito. Packages. wiki, 4 2019b. URL http://wiki.ros.org/Packages. Last checked: 21.04.19.

Lars Tingelstad. Camera calibration. web page, october 2018. URL https://github.com/tingelst/tpk4170-robotics/blob/master/tpk4170/camera_calibration/CameraCalibration.ipynb. Last checked: 30.11.18.

David Maier. *The Theory of Relational Databases*. Unknown, 1983.

Microsoft. Introducing sql server 2019. webpage, 3 2019. URL https://www.microsoft.com/en-us/sql-server/sql-server-2019. Last checked: 10.03.19.

Noel Martignoni. Packages. wiki, 4 2019. URL https://commons.wikimedia.org/wiki/File:ROS-master-node-topic.png. Last checked: 21.04.19.

NOV. St-120 iron roughneck. webpage, 6 2019. URL https://www.nov.com/Segments/Rig$_T$*echnologies*/*Rig*$_E$*quipment*/*Offshore*/*Iron*$_R$*oughnec* 120$_I$*ron*$_R$*oughneck.aspx*. *Lastchecked* : 05.06.19.

Open Source Robotics Foundation. About ros. webpage, 4 2019. URL http://www.ros.org/about-ros/. Last checked: 21.04.19.

Raymond Chen. Ros client. webpage, 4 2019. URL https://medium.com/raymonduchen. Last checked: 23.04.19.

Robert Latek. What is a database? Published Lecture, 2004. URL http://barc.wi.mit.edu/education/bioinfo-mini/db4bio/lecture1.pdf. Last checked: 10.03.19.

TullyFoote. Topics. wiki, 4 2019. URL http://wiki.ros.org/Topics. Last checked: 21.04.19.

Wikipedia. Gas metal arc welding. wiki, 4 2019. URL https://en.wikipedia.org/wiki/Gas$_m$*etal*$_a$*rc*$_w$*eldings*. *Lastchecked* : 12.02.19.

YanqingWu. Master. wiki, 4 2019. URL http://wiki.ros.org/Master. Last checked: 21.04.19.

Appendices

Appendix A

Python code

A.1 Main code

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Created on Wed Jan 30 13:19:55 2019

@author: Kristoffer
"""
from __future__ import division # allows python 2.7 to use true division
import cv2
import numpy as np
from os import listdir
import time
import math
start_time = time.time()

pi = np.pi
# Read image
path = '/home/kristoffer/catkin_ws/src/Thesis/pipe_images/'
#filename = "Weld_station_coord_4_35.png" #Filename Input OK done 2
#filename = "Weld_station_coord.png" #Filename Input OK LOOKS best! 3
#filename = "Weld_station_coord_two_grooves_4_5.png" #Filename Input OK
    done 1
filename = "Weld_station_coord_two_grooves_5_5.png" #Filename Input OK 5
#filename = "Weld_station_coord_deeper.png" #Filename Input OK done 4
savefile = "savefile.png" #Savename image with edges
saveresult = "dp_texture_result.png"#Savename image detected feature

image = path + filename
img = cv2.imread(image,0) # image is grey scale due to (,0)
img1 = cv2.imread(image,1) # used to overlay result on original image
resize_img = cv2.resize(img,(1920,1080))
resize_img1 = cv2.resize(img1,(1366,768))
cv2.imshow('Original image:',resize_img1)

cv2.waitKey() & 0xFF
cv2.destroyAllWindows()

# Blurring tuning:
```

```
[sigma,kernel] = [2,3]
# applying gausian filter to the image
GaussianBlur = cv2.GaussianBlur(img,(kernel,kernel),sigma)
# Edge detector tuning
cnt = 0
canny_sum = 0
[start,stop,step] = [80,240,20]
for i in range(start,stop,step):
    canny1 = i
    print('Loading:',canny1,'out of',stop)
    for j in range(start,stop,step):
        cnt += 1
        canny2 = j
        canny = cv2.Canny(GaussianBlur,canny1,canny2)
        canny_int32 = np.int32(canny)
        canny_sum += canny_int32
print('cnt',cnt)
canny_avg = canny_sum/cnt
print ('canny avg' , canny_avg)
canny_avg = canny_avg.astype(np.uint8)
ret,canny_thr = cv2.threshold(canny_avg,150,255,cv2.THRESH_BINARY)#
    Thresholding
print('ret',ret)
resize_canny_thr = cv2.resize(canny_thr,(1366,768))
cv2.imshow('resized bluh',resize_canny_thr)
cv2.imwrite(path+savefile,canny_thr)# Saving generated canny image
canny_path = path+savefile
print(canny_path)
final_canny_avg = cv2.imread(canny_path,0)
resize_canny_avg = cv2.resize(final_canny_avg,(1366,768))

print('zups')
cv2.imshow('Final Canny avg ',resize_canny_avg)

cv2.waitKey() & 0xFF
cv2.destroyAllWindows()

def neck(image,minValue,maxValue):
    print('Inside Neck detector')
    # Copy canny image to overlay colored lines
    copy_canny = cv2.cvtColor(image,cv2.COLOR_GRAY2BGR)
    # Detect lines
    lines = cv2.HoughLinesP(image,1,1*np.pi/180,1,None,50,10)
    x1_list = []
    y1_list = []
    x2_list = []
    y2_list = []
    u, v = image.shape[::-1]
    if lines is not None:
        for i in range(0,len(lines)):
            l = lines[i][0]
            print(l)
            delta_u = float(np.abs(l[2]-l[0]))
            delta_v = float(np.abs(l[3]-l[1]))
            radangle = math.atan2(delta_v,delta_u)
            print radangle
            degangle = radangle*180/np.pi
            print degangle
```

```
95              if np.abs(degangle) < 35 and np.abs(degangle) > 17:
96                  # Draw lines
97                  print('approved')
98                  print(l)
99                  print(degangle,'deg')
100
101                 x1_list.append(l[0])
102                 y1_list.append(l[1])
103                 x2_list.append(l[2])
104                 y2_list.append(l[3])
105                 d_u = 2*(np.abs(l[0]-l[2]))
106                 d_v = 2*(np.abs(l[1]-l[3]))
107                 if l[1]<v/2 and l[3]<v/2:
108                     cv2.line(copy_canny,(l[0],l[1]),(l[2],l[3]),(0,0,255)
    ,3)
109                     cv2.line(copy_canny,(l[0]-d_u,l[1]-d_v),(l[2],l[3])
    ,(255,0,0),2)
110                     resize_copy_canny = cv2.resize(copy_canny,(1366,768))
111                     X2 = l[2]
112                     Y2 = l[3]
113                 else:
114                     cv2.line(copy_canny,(l[0],l[1]),(l[2],l[3]),(0,0,255)
    ,3)
115                     cv2.line(copy_canny,(l[0]-d_u,l[1]+d_v),(l[2],l[3])
    ,(255,0,0),2)
116                     resize_copy_canny = cv2.resize(copy_canny,(1366,768))
117                     pass
118                 cv2.imshow('neck',resize_copy_canny)
119             else:
120                 print('neck pass')
121                 pass
122         return d_u,d_v,X2,Y2
123
124 def minimum(aList):
125     minPos = aList.index(min(aList))
126     maxPos = aList.index(max(aList))
127     return minPos,maxPos
128
129 print('neck lines:')
130 print('-_-_-_-_-_-_-_-_-_'*3)
131 def tool_joint_top(image):
132     # Copy canny image to overlay colored lines
133     copy_canny = cv2.cvtColor(image,cv2.COLOR_GRAY2BGR)
134     print('Inside Tool Joint detector')
135     # Detect lines
136     lines = cv2.HoughLinesP(image,1,1*np.pi/180,100,None,80,40)
137     x1_list = []
138     y1_list = []
139     x2_list = []
140     y2_list = []
141     u, v = image.shape[::-1]
142     if lines is not None:
143         for i in range(0,len(lines)):
144             l = lines[i][0]
145             degangle = np.arctan([np.abs(l[3]-l[1]),np.abs(l[2]-l[0])])
    *180/np.pi
146 #             print(degangle)
147             if np.abs(degangle[1])<0.5:
```

```
148                     print('pass')
149                     pass
150                 else:# Draw lines
151                     print('x1',l[0],'y1',l[1],'x2',l[2],'y2',l[3])
152                     print (degangle[0], degangle[1])
153                     x1_list.append(l[0])
154                     y1_list.append(l[1])
155                     x2_list.append(l[2])
156                     y2_list.append(l[3])
157             minPos,maxPos = minimum(y1_list)
158             print('Minimum element location',minPos)
159             minValue = y1_list[minPos]
160             print('minValue',minValue)
161             maxValue = y1_list[maxPos]
162             print('maxValue',maxValue)
163 
164             cv2.line(copy_canny,(0,y1_list[minPos]),(u,y2_list[minPos])
    ,(0,255,0),1)
165             resize_copy_canny = cv2.resize(copy_canny,(1366,768))
166             cv2.imshow('tool_joint',resize_copy_canny)
167             return minValue,maxValue
168 
169 def ignorator(image):
170     minValue,maxValue=tool_joint_top(image)
171     cv2.waitKey(0) &0xFF
172     cv2.destroyAllWindows()
173 
174     [d_u,d_v,X2,Y2]=neck(image,minValue,maxValue)
175     cv2.waitKey(0) &0xFF
176     cv2.destroyAllWindows()
177     print(d_u,d_v,X2,Y2)
178     d_v = float(d_v)
179     d_u = float(d_u)
180     a = (((d_v/d_u)*X2)-Y2)
181     b = float(d_v/d_u)
182     print('a',a,'b',b)
183     X = (a+minValue)/b
184     print('X',X)
185     return X,minValue
186 
187 ign_u,tj = ignorator(final_canny_avg)
188 
189 
190 crop_img = final_canny_avg[tj-150:tj+150,int(ign_u)-1000:int(ign_u)]
191 cv2.imshow('cropped',crop_img)
192 cv2.waitKey(0)&0xFF
193 cv2.destroyAllWindows()
194 cv2.imwrite(savefile+path,crop_img)
195 crop_img1 = img1[tj-150:tj+150,int(ign_u)-1000:int(ign_u)]
196 
197 
198 def loadImages(path):
199     # returns an array of images
200     imagesList = listdir(path)
201     loadedImages = []
202     for image in imagesList:
203         image = path+image
204         img = cv2.imread(image,0)
```

```
205         loadedImages.append(img)
206     return loadedImages,imagesList
207 
208 start = '/home/kristoffer/catkin_ws/src/wcm/Master/Code/start/'
209 imgs, imgsList = loadImages(start)
210 print(imgsList)
211 end = '/home/kristoffer/catkin_ws/src/wcm/Master/Code/end/'
212 imge, imgeList = loadImages(end)
213 print(imgsList)
214 
215 width = []
216 height = []
217 # Detecting the start of weld
218 def detectorstart(search, imgs, imge, V0, U1, ign_u, tj, color):
219     # Import image to search in
220     print('Searching in: ',search)
221 
222     image = path + search
223 
224     img_bgr = cv2.imread(image)
225     img_gray = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2GRAY)
226 #    ##########################################################################################
227     cnt = 0
228 
229     for img in imgs:
230         X_0 = img
231         cnt +=1
232         print type(X_0)
233         print('DETECT START:',cnt)
234 
235 #    ##########################################################################################
236         X_0_Blur = cv2.GaussianBlur(X_0,(1,1),2)
237         # applying canny edge detection to the blured image
238         canny2 = cv2.Canny(X_0_Blur,100,200)
239         X_0 = canny2
240         # Get the size of image
241         w, h = X_0.shape[::-1]
242         w1, h1 = img_gray.shape[::-1]
243         threshold = np.linspace(0.7,0.4,num=20)
244         res = cv2.matchTemplate(crop_img,X_0,cv2.TM_CCOEFF_NORMED)
245 
246         for element in threshold:
247             location = np.where(res>= element)
248             # Note the if statment ignoring false points, limits depend on
    image
249             t,p_u,p_v = [0,0,0]
250             for p in zip(*location[::-1]):
251                 if p[1]<V0+20 and p[1]>V0-20: # choses which points to
    ignore
252                     pass# ignore
253                 else:
254                     print('Feature detected at pixel coordinates: ')
255                     print('u = ',p[0])
256                     print('v = ',p[1])
```

```
257                     t += 1
258                     p_u += p[0]
259                     p_v += p[1]
260                     # While loop stopper
261             key = cv2.waitKey(1)
262             if key == 27:
263                 print('while loop engaged')
264                 cv2.destroyAllWindows()
265             if t == 0:
266                 Found = False
267                 pass
268             else:
269                 print('Detected at:',element)
270                 U_s = int(p_u/t)
271                 V_s = int(p_v/t)
272 
273                 [U_e,V_e,we,he]=detectorend(savefile,imge,(0,0,255),U_s,V_s
    )
274                 if U_e == 1 and V_e == 1:
275                     Found = False
276                     print('Not found')
277                     break
278                 else:
279                     Found = True
280                     cv2.rectangle(crop_img1,(U_s,V_s),(U_s+w,V_s+h),(color)
    ,3)
281                     cv2.imshow('detected', crop_img1)
282 
283                     height.append([h,he])
284                     width.append([w,we])
285                     break
286             if Found == True:
287                 print('t',t)
288                 break
289         if Found == True: # stopper    leite gjennom flere bilder
290             break
291     try:
292         return U_s,V_s,U_e,V_e,width,height,Found
293     except:
294         print('except start')
295         a,b,c,d = [1,1,1,1]
296         return a,b,c,d,width,height,True
297 
298 # Detecting the end of weld
299 def detectorend(search,imge,color,u,v):
300     # Import image to search in
301     print('Searching in: ',search)
302     image = path + search
303 
304     img_bgr = cv2.imread(image)
305     img_gray = cv2.cvtColor(img_bgr,cv2.COLOR_BGR2GRAY)
306     # Processing detection image
307     cnt = 0
308     for img in imge:
309         cnt +=1
310         print('DETECT END:',cnt)
311         X_0 = img
312 
```

```
313         X_0_Blur = cv2.GaussianBlur(X_0,(1,1),2)
314         X_0 = cv2.Canny(X_0_Blur,100,200)
315         # Get the size of image
316         w, h = X_0.shape[::-1]
317         w1, h1 = img_gray.shape[::-1]
318         threshold = np.linspace(0.7,0.3,num=20)
319         res = cv2.matchTemplate(crop_img,X_0,cv2.TM_CCOEFF_NORMED)
320         for element in threshold:
321             location = np.where(res>= element)
322             t,p_u,p_v = [0,0,0]
323             for p in zip(*location[::-1]):
324                 if p[1] > v+5 or p[1]<v-5 or u-p[0]<20: # choses which
    points to ignore
325                     pass
326                 else:
327#                     print('Image saved')
328                     print('Feature detected at pixel coordinates: ')
329                     print('u = ',p[0])
330                     print('v = ',p[1])
331                     t = t+1
332                     p_u = p_u + p[0]
333                     p_v = p_v + p[1]
334             key = cv2.waitKey(1)
335             if key == 27:
336                 print('while loop engaged')
337                 cv2.destroyAllWindows()
338             if t == 0:
339     #                print('No features detected')
340                 pass
341             else:
342                 U_e = int(p_u/t)
343                 V_e = int(p_v/t)
344                 cv2.rectangle(crop_img1,(U_e,V_e),(U_e+w,V_e+h),(color),3)
345                 cv2.imshow('detected', crop_img1)
346                 break
347         if t != 0:
348             break
349     try:
350         return U_e,V_e,w,h
351     except:
352         print('except')
353         a,b,c,d = [1,1,1,1]
354         return a,b,c,d
355
356 print('-_-_-_-_-_-_-_-_-_'*3)
357 print('Weld start and end detector:')
358 print('-_-_-_-_-_-_-_-_-_'*3)
359
360 print('Detecting first round: ')
361 [U0,V0,U1,V1,w,h,Found]=detectorstart(savefile,imgs,imge,10,1,ign_u,tj
    ,(0,255,0))
362 print('width',w,'height',h)
363
364 print('Detecting second round: ')
365 [U2,V2,U3,V3,w,h,Found]=detectorstart(savefile,imgs,imge,V0,U1,ign_u,tj
    ,(0,255,0))
366 print('width',type(w),'height',h)
367
```

```
368 if np.abs(U0-U2)<2 and np.abs(U1-U3)<2:
369     V_stop1 = 1
370
371
372 def rename(U_start1,V_start1,U_stop1,V_stop1,U_start2,V_start2,U_stop2,
    V_stop2,w,h):
373     if V_start1 < V_start2:
374         # first detected start is lower
375         x1,y1,x2,y2 = [U_start1,V_start1,U_stop1,V_stop1]
376         U_start1,V_start1,U_stop1,V_stop1 = [U_start2,V_start2,U_stop2,
    V_stop2]
377         U_start2,V_start2,U_stop2,V_stop2 = [x1,y1,x2,y2]
378         U = list([U_start1,U_stop1,U_start2,U_stop2])
379         V = list([V_start1,V_stop1,V_start2,V_stop2])
380         if V_stop1 == 1 or V_stop2 == 1:
381             W = w
382             H = h
383         else:
384             [w0,w1],[w2,w3] = w
385             W = list([w2,w3,w0,w1])
386             [h0,h1],[h2,h3] = h
387             H = list([h2,h3,h0,h1])
388     else:
389         U = list([U0,U1,U2,U3])
390         V = list([V0,V1,V2,V3])
391         if V_stop1 == 1 or V_stop2 == 1:
392             W = w
393             H = h
394         else:
395             [w0,w1],[w2,w3] = w
396             W = list([w0,w1,w2,w3])
397             [h0,h1],[h2,h3] = h
398             H = list([h0,h1,h2,h3])
399         pass
400     return U,V,W,H
401
402 U,V,W,H = rename(U0,V0,U1,V1,U2,V2,U3,V3,w,h)
403 print('U',U)
404
405 def translator(U,V,tj,ign_u):
406     if Found == True:
407         U = U + ign_u-1000
408         U = [int(U[0]),int(U[1]),int(U[2]),int(U[3])]
409         V = V + tj-150
410     else:
411         U2 = int(U[2] + ign_u - 1000)
412         U3 = int(U[3] + ign_u - 1000)
413         V2 = V[2] + tj - 150
414         V3 = V[3] + tj - 150
415         U = [int(U[0]),int(U[1]),U2,U3]
416         V = [V[0],V[1],V2,V3]
417     return U, V
418 U,V = translator(U,V,tj,ign_u)
419 print('U',U)
420 print('First weldgun position from: [',U[0],',',V[0],']to: [',U[1],',',V
    [1],']')
421 print('Second weldgun position from: [',U[2],',',V[2],']to: [',U[3],',',V
    [3],']')
```

```
cv2.waitKey(0) &0xFF
cv2.destroyAllWindows()

print('-_-_-_-_-_-_-_-_-_'*3)
print('Diameter estimation:')
print('-_-_-_-_-_-_-_-_-_'*3)

def tool_joint(image,U,V,ign_u,W,H):
    # Copy canny image to overlay colored lines
    copy_canny = cv2.cvtColor(image,cv2.COLOR_GRAY2BGR)
    print('Inside Tool Joint detector')
    # Detect lines
    lines = cv2.HoughLinesP(image,1,1*np.pi/180,30,None,100,20)
    x1_list = []
    y1_list = []
    x2_list = []
    y2_list = []
    if lines is not None:
        for i in range(0,len(lines)):
            l = lines[i][0]
            degangle = np.arctan(np.abs(l[3]-l[1])/np.abs(l[2]-l[0]))*180/
    np.pi
            if np.abs(degangle)>0.5 or l[1]>V+5 or l[1]<V-50:
                pass
            else:# Draw lines
                print('x1',l[0],'y1',l[1],'x2',l[2],'y2',l[3])
                x1_list.append(l[0])
                y1_list.append(l[1])
                x2_list.append(l[2])
                y2_list.append(l[3])

                cv2.line(copy_canny,(l[0],l[1]),(l[2],l[3]),(255,0,0),3)
                resize_copy_canny = cv2.resize(copy_canny,(1366,768))
                cv2.imshow('tool_joint',resize_copy_canny)
        return x1_list,y1_list,x2_list,y2_list

def groove1(search,U3,U2,V,a,tj,ign_u,W1,W2,H):# Last weld
    # Copy canny image to overlay colored lines
    copy_canny = cv2.cvtColor(search,cv2.COLOR_GRAY2BGR)
    # Detect lines
    length = (U2-U3)*5/10
    gap = int(length)
    lines = cv2.HoughLinesP(search,1,1*np.pi/180,30,None,length,(gap))
    x1_list = []
    y1_list = []
    x2_list = []
    y2_list = []
    if lines is not None:
        print('tj',tj,'U3',U3,'U2',U2,'W1',W1,'W2',W2,'H',H)
        for i in range(0,len(lines)):
            l = lines[i][0]
            print(l)
            degangle = np.arctan(np.abs(l[3]-l[1])/np.abs(l[2]-l[0]))*180/
    np.pi

            if np.abs(degangle)>0.5 or l[1]<tj or l[1]>tj+150 or l[0]<=U3
    -4*W1 or l[2]>=U2+4*W2 or l[1]>tj+3*H:#or l[1]<V_stop2-15
                pass
```

```
477             else:# Draw lines
478                 print('x1',l[0],'y1',l[1],'x2',l[2],'y2',l[3])
479                 x1_list.append(l[0])
480                 y1_list.append(l[1])
481                 x2_list.append(l[2])
482                 y2_list.append(l[3])
483         print('tj',tj,'U3',U3,'U2',U2,'W1',W1,'W2',W2,'H',H)
484         minPos,maxPos = minimum(y1_list)
485         cv2.line(copy_canny,(x1_list[minPos],y1_list[minPos]),(x2_list[
    minPos],y2_list[minPos]),(255,0,0),3)
486         resize_copy_canny = cv2.resize(copy_canny,(1366,768))
487         cv2.imshow('groove_1',resize_copy_canny)
488         return x1_list,y1_list,x2_list,y2_list
489
490 def groove2(search,U,V,g1,ign_u,W1,W2,H):# First weld
491     # Copy canny image to overlay colored lines
492     copy_canny = cv2.cvtColor(search,cv2.COLOR_GRAY2BGR)
493     # Detect lines
494     length = (U[0]-U[1])/4
495     gap = 10
496     lines = cv2.HoughLinesP(search,1,1*np.pi/180,30,None,length,(gap))
497     x1_list = []
498     y1_list = []
499     x2_list = []
500     y2_list = []
501     if lines is not None:
502         for i in range(0,len(lines)):
503             l = lines[i][0]
504             degangle = np.arctan(np.abs(l[3]-l[1])/np.abs(l[2]-l[0]))*180/
    np.pi
505             if np.abs(degangle)>0.5 or l[1]<g1+5 or l[1]>g1+200 or l[0]<U
    [3]-W1 or l[2]>U[2]+W2:
506                 pass
507             else:# Draw lines
508                 print('x1',l[0],'y1',l[1],'x2',l[2],'y2',l[3])
509                 x1_list.append(l[0])
510                 y1_list.append(l[1])
511                 x2_list.append(l[2])
512                 y2_list.append(l[3])
513
514                 cv2.line(copy_canny,(l[0],l[1]),(l[2],l[3]),(255,0,0),3)
515                 resize_copy_canny = cv2.resize(copy_canny,(1366,768))
516                 cv2.imshow('groove_2',resize_copy_canny)
517         return x1_list,y1_list,x2_list,y2_list
518
519 def transformation(U,V):
520     pi = np.pi
521     def rotx(theta):
522         ct = np.cos(theta);
523         st = np.sin(theta);
524         R = np.array([[1, 0,  0],
525                       [0,ct,-st],
526                       [0,st, ct]])
527         return R # theta rotation about x-axis
528     def rotz(theta):
529         ct = np.cos(theta);
530         st = np.sin(theta);
531         R = np.array([[ct,-st,0],
```

```
532                         [st, ct,0],
533                         [0,  0, 1]])
534         return R # theta rotation about z-axis
535     def tran(R,v):
536         T = np.array([[R[0][0],R[0][1],R[0][2],float(v[0])],
537                         [R[1][0],R[1][1],R[1][2],float(v[1])],
538                         [R[2][0],R[2][1],R[2][2],float(v[2])],
539                         [0,0,0,1]])
540         return np.round(T,3)
541 
542     K = np.array([[3250, 0, 3017/2],
543                   [0, 3650, 1704/2],
544                   [0, 0, 1]])
545 
546     empt = []
547 
548     z = 0.7 # z is distance between camera and point along optical axis
549     p = np.array([[U],
550                   [V]])
551     tc_co = np.array([[-0.3],
552                         [0],
553                         [z]])
554     s_x = (p[0]-K[0][2])/K[0][0]
555     s_y = (p[1]-K[1][2])/K[1][1]
556     s = np.append(empt,[s_x,s_y])
557     s = np.reshape(s,(2,1))
558     s_tilde = np.append(s,1)
559     s_tilde = np.reshape(s_tilde,(3,1))
560     rc_cp = (s_tilde)*z
561     rc_cp_tilde = np.append(rc_cp,1)
562     rc_cp_tilde = np.reshape(rc_cp_tilde,(4,1))
563     Rx = rotx(pi/2)
564     Rz = rotz(-pi/2)
565     RT = np.dot(Rx,Rz)
566 #     RT = Rx@Rz
567     T = tran(RT,tc_co)
568     T_inv = np.linalg.inv(T)
569     ro_op_tilde = np.dot(T_inv,rc_cp_tilde)
570     return ro_op_tilde[0],ro_op_tilde[1],ro_op_tilde[2]
571 
572 # Welding programme
573 def layer(soft_height,max_weld_height,d):
574     if soft_height/max_weld_height<=(1.01):
575         string_height = soft_height
576         layers = 1
577         D = d+2*string_height
578         weld_area = (pi/4)*(D**2-d**2)
579         print('Need one layer of weld, ',np.round(string_height*1000,3),'mm
    each layer')
580     elif soft_height/max_weld_height>(1.01) and soft_height/max_weld_height
    <=(2.01):
581         string_height = soft_height/2
582         layers = 2
583         D = d+2*string_height
584         weld_area1 = (pi/4)*(D**2-d**2)
585         D1 = d+2*2*string_height
586         weld_area2 = (pi/4)*(D1**2-D**2)
587         weld_area = [weld_area1,weld_area2]
```

```
588         print('Need two layers of weld, ',np.round(string_height*1000,3),'
    mm each layer')
589     elif soft_height/max_weld_height>(2.01) and soft_height/max_weld_height
    <=(3.01):
590         string_height = soft_height/3
591         layers = 3
592         D = d+1*2*string_height
593         weld_area1 = (pi/4)*(D**2-d**2)
594         D1 = d+2*2*string_height
595         weld_area2 = (pi/4)*(D1**2-D**2)
596         D2 = d+3*2*string_height
597         weld_area3 = (pi/4)*(D2**2-D1**2)
598         weld_area = [weld_area1,weld_area2,weld_area3]
599         print('Need three layers of weld, ',np.round(string_height*1000,3),
    'mm each layer')
600     else:
601         string_height = soft_height/4
602         layers = 4
603         D = d+1*2*string_height
604         weld_area1 = (pi/4)*(D**2-d**2)
605         D1 = d+2*2*string_height
606         weld_area2 = (pi/4)*(D1**2-D**2)
607         D2 = d+3*2*string_height
608         weld_area3 = (pi/4)*(D2**2-D1**2)
609         D3 = d+4*2*string_height
610         weld_area4 = (pi/4)*(D3**2-D2**2)
611         weld_area = [weld_area1,weld_area2,weld_area3,weld_area4]
612         print('Need four layers of weld, ',np.round(string_height*1000,3),'
    mm each layer')
613     return layers,string_height,weld_area
614
615 def weld_strings(aksial_start,aksial_stop,weld_width):
616     weld_length = aksial_start-aksial_stop
617     strings = int(weld_length / weld_width)
618     return strings
619
620 def grooveLogic(U,V,tj):
621     weld_feed = 9.1 # 8-9.1 [m/min]
622     max_weld_height = 0.005 # 5-6mm [m]
623     rod_dia = 0.0012 # 1.2mm [m]
624     rod_area = pi*(rod_dia*rod_dia)/4 # m^2
625     weld_width = 0.007 # ca 7mm
626     print ('U',U)
627     print ('V',V)
628     print ('tj',tj)
629
630     if np.abs((V[0]+V[1])/2-(V[2]+V[3])/2)<=5 or V[3] == 1 or V[1] == 1 or
    V[2]-V[3] == 0:
631         print('Pipe has only one groove')
632 #         V[1] = V[3]
633         print('tool joint')
634         [tjx1,tjy1,tjx2,tjy2]=tool_joint(canny_avg,U,tj,ign_u,W,H)
635         cv2.waitKey() &0xFF
636         cv2.destroyAllWindows()
637         print('groove1')
638         print(W)
639         print(W[0])
640         print(W[0][1])
```

```
641         [g1x1,g1y1,g1x2,g1y2]=groove1(canny_avg,U[1],U[0],tj,50,tj,ign_u,W
    [0][1],W[0][0],H[0][0])
642         cv2.waitKey() &0xFF
643         cv2.destroyAllWindows()
644         print('g1y1',g1y1,type(g1y1))
645         [ws1_x,ws1_y,ws1_z] = transformation(U[0],V[0])
646         [we1_x,we1_y,we1_z] = transformation(U[1],V[1])
647         print('Start position for weld gun: ')
648         print(' X: ',ws1_x*1000,'\n Y: ',ws1_y*1000,'\n Z: ',ws1_z*1000)
649         print('End position for weld gun: ')
650         print(' X: ',we1_x*1000,'\n Y: ',we1_y*1000,'\n Z: ',we1_z*1000)
651 
652         groove_dia = (g1y1[0]+g1y2[0])/2
653         print('Machined groove pixel ',groove_dia)
654         [g_x,g_y,g_z] = transformation(g1x1[0],groove_dia)
655         print('Machined groove diameter: ',g_z*1000*2,'mm')
656         tool_joint_dia = (tjy1[0]+tjy2[0])/2
657         [tj_x,tj_y,tj_z] = transformation(tjx1[0],tool_joint_dia)
658         print('Tool joint diameter: ',tj_z*1000*2,'mm')
659 
660         # weld program
661         soft_height = tj_z-g_z
662         print('soft-lagging height: ',np.round(soft_height*1000,3),'mm')
663 
664         layers,string_height,weld_area = layer(soft_height,max_weld_height,
    g_z)
665         strings = weld_strings(ws1_y,we1_y,weld_width)
666         print('Number of weld strings per layer: ',strings)
667         weld_volume_pr_min = rod_area*weld_feed
668         weld_volume_pr_rev = weld_area*weld_width
669         pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
670         print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
671         string_time = 60/pipe_rev
672         print('time per string',np.round(string_time,1),'seconds')
673         layer_time = string_time*strings
674         print('time per layer',np.round(layer_time/60,1),'minutes')
675         weld_time = layer_time*layers
676         print('Total weld time',np.round(weld_time/60,1),'minutes')
677         rod_used = weld_time/60*weld_feed
678         print('Meters of wire used: ',np.round(rod_used,1))
679 
680     elif (V[0]+V[1]>V[2]+V[3]) and np.abs(U[1]-U[0])<np.abs(U[3]-U[2]):
681         print('Pipe has two grooves')
682         [tjx1,tjy1,tjx2,tjy2]=tool_joint(canny_avg,U,V[2],ign_u,W,H)
683         cv2.waitKey() &0xFF
684         cv2.destroyAllWindows()
685         print('groove1')
686         [g1x1,g1y1,g1x2,g1y2]=groove1(canny_avg,U[3],U[2],V,15,tj,ign_u,W
    [3],W[2],2*H[0])
687         cv2.waitKey() &0xFF
688         cv2.destroyAllWindows()
689         print('g1y2',g1y2)
690         print('groove2')
691         [g2x1,g2y1,g2x2,g2y2]=groove2(canny_avg,U,V,g1y2[0],ign_u,W[3],W
    [2],H)
692         cv2.waitKey() &0xFF
693         cv2.destroyAllWindows()
694 
```

```
695         [ws2_x,ws2_y,ws2_z] = transformation(U[0],V[0])
696         [we2_x,we2_y,we2_z] = transformation(U[1],V[1])
697         print('First start position for weld gun: ')
698         print(' X: ',ws2_x*1000,'\n Y: ',ws2_y*1000,'\n Z: ',ws2_z*1000)
699         print('First end position for weld gun: ')
700         print(' X: ',we2_x*1000,'\n Y: ',we2_y*1000,'\n Z: ',we2_z*1000)
701 
702         [ws1_x,ws1_y,ws1_z] = transformation(U[2],V[2])
703         [we1_x,we1_y,we1_z] = transformation(U[3],V[3])
704         print('Last start position for weld gun: ')
705         print(' X: ',ws1_x*1000,'\n Y: ',ws1_y*1000,'\n Z: ',ws1_z*1000)
706         print('Last end position for weld gun: ')
707         print(' X: ',we1_x*1000,'\n Y: ',we1_y*1000,'\n Z: ',we1_z*1000)
708 
709         groove_dia = (g1y1[0]+g1y2[0])/2
710         [g_x,g_y,g_z] = transformation(g1x1[0],groove_dia)
711 
712         groove_dia2 = (g2y1[0]+g2y2[0])/2
713         [g_x,g_y,g_z2] = transformation(g2x1[0],groove_dia2)
714 
715         tool_joint_dia = (tjy1[0]+tjy2[0])/2
716         [tj_x,tj_y,tj_z] = transformation(tjx1[0],tool_joint_dia)
717         print('Tool joint diameter: ',tj_z*1000*2,'mm')
718         print('Machined groove diameter highest diameter: ',g_z*1000*2,'mm'
    )
719         print('Machined groove diameter lowest diameter: ',g_z2*1000*2,'mm'
    )
720 
721         # weld program
722         print('-_-_-_-_-_-_-_-_-_-_'*3)
723         print('Welding programme:')
724         print('-_-_-_-_-_-_-_-_-_-_'*3)
725         # lowest diameter groove:
726 
727         soft_height = g_z-g_z2
728         print('soft-lagging height: ',np.round(soft_height*1000,3),'mm')
729 
730         layers,string_height,weld_area = layer(soft_height,max_weld_height,
    g_z2)
731         strings = weld_strings(ws2_y,we2_y,weld_width)
732         print('Number of weld strings smaller groove: ',strings)
733         weld_volume_pr_min = rod_area*weld_feed
734 
735         if layers == 1:
736             weld_volume_pr_rev = weld_area[0]*weld_width
737             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
738             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
739             string_time = 60/pipe_rev
740             weld_time1 = string_time*strings
741             print('time for first groove',np.round(weld_time1/60,1),'
    minutes')
742         elif layers == 2:
743             weld_volume_pr_rev = weld_area[0]*weld_width
744             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
745             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
746             string_time = 60/pipe_rev
747             layer_time1 = string_time*strings
748             print('time first layer',np.round(layer_time1/60,1),'minutes')
```

```
749
750             weld_volume_pr_rev = weld_area[1]*weld_width
751             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
752             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
753             string_time = 60/pipe_rev
754             layer_time2 = string_time*strings
755             print('time first groove',np.round(layer_time2/60,1),'minutes')
756             weld_time1 = layer_time1 + layer_time2
757         elif layers == 3:
758             weld_volume_pr_rev = weld_area[0]*weld_width
759             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
760             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
761             string_time = 60/pipe_rev
762             layer_time1 = string_time*strings
763             print('time first layer',np.round(layer_time1/60,1),'minutes')
764
765             weld_volume_pr_rev = weld_area[1]*weld_width
766             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
767             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
768             string_time = 60/pipe_rev
769             layer_time2 = string_time*strings
770             print('time second layer',np.round(layer_time2/60,1),'minutes')
771
772             weld_volume_pr_rev = weld_area[2]*weld_width
773             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
774             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
775             string_time = 60/pipe_rev
776             print('time per string',np.round(string_time,1),'seconds')
777             layer_time3 = string_time*strings
778             print('time third layer',np.round(layer_time3/60,1),'minutes')
779             weld_time1 = layer_time1+layer_time2+layer_time3
780
781         # Larger diameter groove
782
783         soft_height = tj_z-g_z
784         print('soft-lagging height: ',np.round(soft_height*1000,3),'mm')
785
786         layers,string_height,weld_area = layer(soft_height,max_weld_height,
    g_z)
787         strings = weld_strings(ws1_y,we1_y,weld_width)
788         print('Number of weld strings larger groove: ',strings)
789         weld_volume_pr_min = rod_area*weld_feed
790
791         if layers == 1:
792             weld_volume_pr_rev = weld_area[0]*weld_width
793             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
794             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
795             string_time = 60/pipe_rev
796             weld_time2 = string_time*strings
797             print('time second groove',np.round(weld_time2/60,1),'minutes')
798         elif layers == 2:
799             weld_volume_pr_rev = weld_area[0]*weld_width
800             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
801             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
802             string_time = 60/pipe_rev
803             layer_time1 = string_time*strings
804             print('time first layer',np.round(layer_time1/60,1),'minutes')
805
```

```
806             weld_volume_pr_rev = weld_area[1]*weld_width
807             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
808             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
809             string_time = 60/pipe_rev
810             layer_time2 = string_time*strings
811             print('time second layer',np.round(layer_time2/60,1),'minutes')
812             weld_time2 = layer_time1 + layer_time2
813         elif layers == 3:
814             weld_volume_pr_rev = weld_area[0]*weld_width
815             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
816             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
817             string_time = 60/pipe_rev
818             layer_time1 = string_time*strings
819             print('time first layer',np.round(layer_time1/60,1),'minutes')
820 
821             weld_volume_pr_rev = weld_area[1]*weld_width
822             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
823             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
824             string_time = 60/pipe_rev
825             layer_time2 = string_time*strings
826             print('time second layer',np.round(layer_time2/60,1),'minutes')
827 
828             weld_volume_pr_rev = weld_area[2]*weld_width
829             pipe_rev = weld_volume_pr_min/weld_volume_pr_rev # rev/min
830             print('Pipe rotational speed: ',np.round(pipe_rev,3),'rev/min')
831             string_time = 60/pipe_rev
832             print('time per string',np.round(string_time,1),'seconds')
833             layer_time3 = string_time*strings
834             print('time third layer',np.round(layer_time3/60,1),'minutes')
835             weld_time2 = layer_time1+layer_time2+layer_time3
836         weld_time = weld_time1+weld_time2
837         print('Total weld time',np.round(weld_time/60,1),'minutes')
838         rod_used = weld_time/60*weld_feed
839         print('Meters of wire used: ',np.round(rod_used,1))
840 
841     else:
842         print('Kristoffer is confused')
843 
844 grooveLogic(U,V,tj)
845 
846 print('--- %s seconds ---' % np.round((time.time()- start_time),2))
```

A.2 ROS Nodes

A.2.1 Camera service node

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import cv2
from cv_bridge import CvBridge
from custom_srvs.srv import image, imageResponse

class Camera_service():

    def callback(self, request): # request is just a string from client to initialize service
        while not rospy.is_shutdown():
            if request.camera == "cam":
                print "requested camera"
                camera_tag = 0 # ls -ltrh /dev/video* (Find Cam in terminal)
                cap = cv2.VideoCapture(camera_tag)

                if not cap.isOpened():
                    print('Camera not opened!')
                rval, frame = cap.read()
                if rval:
                    rval, frame = cap.read()
                    print frame
                    img = self.bridge.cv2_to_imgmsg(frame, "bgr8")
                    print("Sending image...")
                    response = imageResponse(img.width,img.height,img.is_bigendian,img.encoding,img.data)
                    print('Image sent')
            elif request.camera == "file":
                print "requested image from file"
                camera_tag = str(self.path+self.filename)
                print camera_tag
                imgur = cv2.imread(camera_tag,1)
                img = self.bridge.cv2_to_imgmsg(imgur,"bgr8")
                print('Sending image...')
                response = imageResponse(img.width,img.height,img.is_bigendian,img.encoding,img.data)
                print ('Image sent')
            else:
                print "Request error"
            return response

    def main(self):
        print('inside main camera_service')
        self.path = "/home/kristoffer/catkin_ws/src/Thesis/pipe_images/"
#        self.filename = "Weld_station_coord_two_grooves_5_5.png" # ok
#        self.filename = "Weld_station_coord_deeper.png" # ok
#        self.filename = "Weld_station_coord_two_grooves_4_5.png" # not
        self.filename = "Weld_station_coord.png" # ok
#        self.filename = "Weld_station_coord_4_35.png" # not
```

```
49          self.bridge = CvBridge()
50
51          rospy.init_node('camera_service_node', anonymous=True)
52          self.srv = rospy.Service("/camera_service",image,self.callback)
53          rospy.loginfo('Service is running')
54          rospy.spin()
55
56  if __name__ == '__main__':
57      try:
58          camera_service = Camera_service()
59          camera_service.main()
60      except rospy.ROSInterruptException:
61          cv2.destroyAllWindows(0)
62          pass
```

A.2.2 Edged image processing service node

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Service
"""

import rospy
import cv2
import numpy as np
from cv_bridge import CvBridge
from custom_srvs.srv import image, imageResponse

class Edged_image_processing_service():

    def callback(self, request): # request is just a string from client to initialize service
        while not rospy.is_shutdown():
            # Do stuff to the info
            img_in = self.imgur
            img = img_in
            # Blurring tuning:
            [sigma,kernel] = [2,3]
            # applying gausian filter to the image
            GaussianBlur = cv2.GaussianBlur(img,(kernel,kernel),sigma)
            # Edge detector tuning
            cnt = 0
            canny_sum = 0
            [start,stop,step] = [80,240,20]
            for i in range(start,stop,step):
                canny1 = i
                print('Loading:',canny1,'out of',stop)
                for j in range(start,stop,step):
                    cnt += 1
                    canny2 = j
                    canny = cv2.Canny(GaussianBlur,canny1,canny2)
                    canny_int32 = np.int32(canny)
                    canny_sum += canny_int32
            print('cnt',cnt)
            canny_avg = canny_sum/cnt
            print ('canny avg' , canny_avg)
            canny_avg = canny_avg.astype(np.uint8)
            ret,canny_thr = cv2.threshold(canny_avg,130,255,cv2.THRESH_BINARY)# Thresholding
            print('ret',ret)
            cv2.imwrite(self.path+self.savefile,canny_thr)# Saving generated canny image
            canny_path = self.path+self.savefile
            print(canny_path)
            fin_img = cv2.imread(canny_path,0)

            img = self.bridge.cv2_to_imgmsg(fin_img,"mono8")
            print('Sending image...')
            response = imageResponse(img.width,img.height,img.is_bigendian,img.encoding,img.data)
            print ('Image sent')

```

```
53              return response
54
55      def main(self):
56          print('inside main edged image processing service')
57          self.path = '/home/kristoffer/catkin_ws/src/Thesis/pipe_images/'
58          self.savefile = 'saved_canny_avg.png'
59          self.bridge = CvBridge()
60
61          print "Waiting for camera service"
62          rospy.wait_for_service("/camera_service")
63          try:
64              info = rospy.ServiceProxy("/camera_service",image)
65              response = info("file")
66              print (response.encoding)
67              self.imgur = self.bridge.imgmsg_to_cv2(response,
    desired_encoding="passthrough")
68          except rospy.ServiceException, e:
69              print "Service call failed: %s"%e
70
71          rospy.init_node('edged_image_processing_node', anonymous=True)
72          # Sending result to callback
73          self.srv = rospy.Service("/edges_image_processing_service",image,
    self.callback)
74          rospy.loginfo('Edges Service is running')
75          rospy.spin()
76
77  if __name__ == '__main__':
78      try:
79          edged_image_processing_service = Edged_image_processing_service()
80          edged_image_processing_service.main()
81      except rospy.ROSInterruptException:
82          cv2.destroyAllWindows(0)
83          pass
```

A.2.3 Crop image service node

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import rospy
import cv2
import numpy as np
import math
from cv_bridge import CvBridge
from custom_srvs.srv import image, imageResponse
from custom_srvs.srv import two_int, two_intResponse

class Crop_image_service():

    def __init__(self):
        self.bridge = CvBridge()

    def neck(self,image,minValue,maxValue):
        print('Inside Neck detector')
        # Copy canny image to overlay colored lines
        copy_canny = cv2.cvtColor(image,cv2.COLOR_GRAY2BGR)
        # Detect lines
        lines = cv2.HoughLinesP(image,1,1*np.pi/180,1,None,50,10)
        x1_list = []
        y1_list = []
        x2_list = []
        y2_list = []
        u, v = image.shape[::-1]
        if lines is not None:
            for i in range(0,len(lines)):
                l = lines[i][0]
                print(l)
                delta_u = float(np.abs(l[2]-l[0]))
                delta_v = float(np.abs(l[3]-l[1]))
                radangle = math.atan2(delta_v,delta_u)
                print radangle
                degangle = radangle*180/np.pi
                print degangle
                if np.abs(degangle) < 35 and np.abs(degangle) > 17:
                    # Draw lines
                    print('approved')
                    print(l)
                    print(degangle,'deg')

                    x1_list.append(l[0])
                    y1_list.append(l[1])
                    x2_list.append(l[2])
                    y2_list.append(l[3])
                    self.d_u = 2*(np.abs(l[0]-l[2]))
                    self.d_v = 2*(np.abs(l[1]-l[3]))
                    if l[1]<v/2 and l[3]<v/2:
                        cv2.line(copy_canny,(l[0],l[1]),(l[2],l[3])
    ,(0,0,255),3)
                        cv2.line(copy_canny,(l[0]-self.d_u,l[1]-self.d_v),(
    l[2],l[3]),(255,0,0),2)
                        self.X2 = l[2]
                        self.Y2 = l[3]
```

```
55                      else:
56                          cv2.line(copy_canny,(l[0],l[1]),(l[2],l[3])
    ,(0,0,255),3)
57                          cv2.line(copy_canny,(l[0]-self.d_u,l[1]+self.d_v),(
    l[2],l[3]),(255,0,0),2)
58                          pass
59                  else:
60                      print('neck pass')
61                      pass
62
63      def minimum(self, aList):
64          minPos = aList.index(min(aList))
65          maxPos = aList.index(max(aList))
66          return minPos,maxPos
67
68      print('neck lines:')
69      print('-_-_-_-_-_-_-_-_-_'*3)
70      def tool_joint_top(self, image):
71          # Copy canny image to overlay colored lines
72          copy_canny = cv2.cvtColor(image,cv2.COLOR_GRAY2BGR)
73          print('Inside Tool Joint detector')
74          # Detect lines
75          lines = cv2.HoughLinesP(image,1,1*np.pi/180,100,None,80,40)
76          x1_list = []
77          y1_list = []
78          x2_list = []
79          y2_list = []
80          u, v = image.shape[::-1]
81          if lines is not None:
82              for i in range(0,len(lines)):
83                  l = lines[i][0]
84                  degangle = np.arctan([np.abs(l[3]-l[1]),np.abs(l[2]-l[0])])
    *180/np.pi
85                  if np.abs(degangle[1])<0.5:
86                      print('pass')
87                      pass
88                  else:# Draw lines
89                      print('x1',l[0],'y1',l[1],'x2',l[2],'y2',l[3])
90                      print (degangle[0], degangle[1])
91                      x1_list.append(l[0])
92                      y1_list.append(l[1])
93                      x2_list.append(l[2])
94                      y2_list.append(l[3])
95              minPos,maxPos = self.minimum(y1_list)
96              print('Minimum element location',minPos)
97              self.minValue = y1_list[minPos]
98              print('minValue',self.minValue)
99              self.maxValue = y1_list[maxPos]
100             print('maxValue',self.maxValue)
101
102             cv2.line(copy_canny,(0,y1_list[minPos]),(u,y2_list[minPos])
    ,(0,255,0),1)
103             self.resize_copy_canny = cv2.resize(copy_canny,(1366,768))
104             print("end of tool joint top")
105
106
107     def ignorator(self,image):
108         print ("Inside ignorator")
```

```
109         print(type(image))
110         self.tool_joint_top(image)
111         print("Going for neck!")
112         self.neck(image,self.minValue,self.maxValue)
113         print(self.d_u,self.d_v,self.X2,self.Y2)
114         self.d_v = float(self.d_v)
115         self.d_u = float(self.d_u)
116         a = (((self.d_v/self.d_u)*self.X2)-self.Y2)
117         b = float(self.d_v/self.d_u)
118         print('a',a,'b',b)
119         self.X = (a+self.minValue)/b
120         self.ign_u = self.X
121         self.tj = self.minValue
122         print('X',self.X)
123 
124     def callback0(self,request):
125         final_canny_avg = self.bridge.imgmsg_to_cv2(self.edge_resp)
126         print (type(final_canny_avg))
127         self.ignorator(final_canny_avg)
128         crop_img = final_canny_avg[self.tj-150:self.tj+150,int(self.ign_u)
    -1000:int(self.ign_u)]
129         img = self.bridge.cv2_to_imgmsg(crop_img, "mono8")
130         print('Sending image...')
131         crop_resp = imageResponse(img.width,img.height,img.is_bigendian,img
    .encoding,img.data)
132         print ('Image sent')
133         org_res = self.bridge.imgmsg_to_cv2(self.org_resp)
134         print("after org res")
135         crop_img1 = org_res[self.tj-150:self.tj+150,int(self.ign_u)-1000:
    int(self.ign_u)]
136         self.im = self.bridge.cv2_to_imgmsg(crop_img1,"bgr8")
137         print('end of callback0')
138 
139         return crop_resp
140 
141     def callback1(self,request):
142         print('Sending image...')
143         crop_resp_col = imageResponse(self.im.width,self.im.height,self.im.
    is_bigendian,self.im.encoding,self.im.data)
144         print ('Image sent')
145         return crop_resp_col
146 
147     def callback2(self,request):
148         print('sending tj and ign_u')
149         two_int_resp = two_intResponse(self.tj, self.ign_u)
150         print("tj and ign_u sent!")
151         return two_int_resp
152 
153     def main(self):
154         print('inside main crop image service')
155         #Insert how to get the info
156         print "Waiting for edges image service"
157         rospy.wait_for_service("/edges_image_processing_service")
158         try:
159             print('Found edge service')
160             edge_client_node = rospy.ServiceProxy("/
    edges_image_processing_service",image)
161             print("Request sent")
```

```
162             self.edge_resp = edge_client_node("Run Forrest, Run!")
163             print("Server responded")
164 
165 #             edge_img = self.bridge.imgmsg_to_cv2(self.edge_resp,
    desired_encoding="passthrough")
166 #             r_edge_img = cv2.resize(edge_img,(1366,768))
167 #             cv2.imshow("service image",r_edge_img)
168 #             cv2.waitKey()
169 #             cv2.destroyAllWindows()
170             print"does stuff!"
171 #             return self.edge_resp
172         except rospy.ServiceException, e:
173             print "Service call failed: %s"%e
174         # TEST Below
175         print "Waiting for original image service"
176         rospy.wait_for_service("/camera_service")
177         try:
178             print('Found original service')
179             cam_client = rospy.ServiceProxy("/camera_service",image)
180             print("Request sent")
181             self.org_resp = cam_client("file")
182             print("Server responded")
183 
184 #             org_img = self.bridge.imgmsg_to_cv2(self.org_resp,
    desired_encoding="passthrough")
185 #             r_org_img = cv2.resize(org_img,(1366,768))
186 #             cv2.imshow("original image",r_org_img)
187 #             cv2.waitKey()
188 #             cv2.destroyAllWindows()
189             print"does stuff!"
190 #             return self.edge_resp
191         except rospy.ServiceException, e:
192             print "Service call failed: %s"%e
193         # TEST Above
194         # end new info
195         print("create node")
196         rospy.init_node('crop_image_node', anonymous=True)
197         # Sending result to callback
198         self.s = rospy.Service("/crop_image_service",image,self.callback0)
199         rospy.loginfo('Crop image Service is running')
200 
201         self.sr = rospy.Service("/crop_col_image_service",image,self.
    callback1)
202         rospy.loginfo('Crop color image Service is running')
203 
204         self.sr = rospy.Service("/tj_ign_u",two_int,self.callback2)
205         rospy.loginfo('Tool joint and ign_u Service is running')
206 
207         rospy.spin()
208 
209 if __name__ == '__main__':
210     try:
211         crop_image_service = Crop_image_service()
212         crop_image_service.main()
213     except rospy.ROSInterruptException:
214         cv2.destroyAllWindows(0)
215         pass
```

A.2.4 Feature detection service node

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-


import rospy
import cv2
import numpy as np
from os import listdir
from cv_bridge import CvBridge
from custom_srvs.srv import image, imageResponse
from custom_srvs.srv import two_int
from custom_srvs.srv import img_coord, img_coordResponse

class Feature_detector_service():

    def __init__(self):
        print("creating node")
        rospy.init_node('feature_detector_node', anonymous=True)
        self.bridge = CvBridge()
        self.path = '/home/kristoffer/catkin_ws/src/Thesis/pipe_images/'
        self.savefile = 'saved_canny_avg.png'
    def loadImages(self,path):
        # returns an array of images
        self.imagesList = listdir(path)
        self.loadedImages = []
        for item in self.imagesList:
            item = path+item
            img = cv2.imread(item,0)
            self.loadedImages.append(img)

    # Detecting the start of weld
    def detectorstart(self,crop_img, crop_img1, imgs, imge, V0, U1, ign_u,
    tj, color):
        # Import image to search in
        print('Searching in crop_img: ')
        img_gray = crop_img
        cnt = 0

        for img in imgs:
            X_0 = img
            cnt +=1
            print('DETECT START:',cnt)

            X_0_Blur = cv2.GaussianBlur(X_0,(1,1),2)
            # applying canny edge detection to the blured image
            canny2 = cv2.Canny(X_0_Blur,100,200)
            X_0 = canny2
            # Get the size of image
            w, h = X_0.shape[::-1]
            w1, h1 = img_gray.shape[::-1]
            threshold = np.linspace(0.7,0.4,num=20)
            res = cv2.matchTemplate(crop_img,X_0,cv2.TM_CCOEFF_NORMED)

            for element in threshold:
                location = np.where(res>= element)
                # Note the if statment ignoring false points, limits depend
```

```
    on image
                t,p_u,p_v = [0,0,0]
                for p in zip(*location[::-1]):
                    if p[1]<V0+20 and p[1]>V0-20: # choses which points to
ignore
                        pass# ignore
                    else:
                        print('Feature detected at pixel coordinates: ')
                        print('u = ',p[0])
                        print('v = ',p[1])
                        t += 1
                        p_u += p[0]
                        p_v += p[1]
                        # While loop stopper
                key = cv2.waitKey(1)
                if key == 27:
                    print('while loop engaged')
                    cv2.destroyAllWindows()
                if t == 0:
                    Found = False
                    pass
                else:
                    print('Detected at:',element)
                    U_s = int(p_u/t)
                    V_s = int(p_v/t)

                    [U_e,V_e,we,he]=self.detectorend(crop_img,crop_img1,
imge,(0,0,255),U_s,V_s)
                    if U_e == 1 and V_e == 1:
                        Found = False
                        print('Not found')
                        break
                    else:
                        Found = True
                        cv2.rectangle(crop_img1,(U_s,V_s),(U_s+w,V_s+h),(
color),3)
                        self.height.append([h,he])
                        self.width.append([w,we])
                        break
                if Found == True:
                    print('t',t)
                    break
            if Found == True: # stopper    leite gjennom flere bilder
                break
        try:
            return U_s,V_s,U_e,V_e,self.width,self.height,Found
        except:
            print('except start')
            a,b,c,d = [1,1,1,1]
            return a,b,c,d,self.width,self.height,True

    # Detecting the end of weld
    def detectorend(self,crop_img,crop_img1,imge,color,u,v):
        # Import image to search in
        print('Searching in: crop_img')
        img_gray = crop_img
        # Processing detection image
        cnt = 0
```

```
110         for img in imge:
111             cnt +=1
112             print('DETECT END:',cnt)
113             X_0 = img
114             X_0_Blur = cv2.GaussianBlur(X_0,(1,1),2)
115             X_0 = cv2.Canny(X_0_Blur,100,200)
116             w, h = X_0.shape[::-1] # Get the size of image
117             w1, h1 = img_gray.shape[::-1]
118             threshold = np.linspace(0.7,0.3,num=20)
119             res = cv2.matchTemplate(crop_img,X_0,cv2.TM_CCOEFF_NORMED)
120             for element in threshold:
121                 location = np.where(res>= element)
122                 t,p_u,p_v = [0,0,0]
123                 for p in zip(*location[::-1]):
124                     if p[1] > v+5 or p[1]<v-5 or u-p[0]<20: # choses which
    points to ignore
125                         pass
126                     else:
127                         print('Feature detected at pixel coordinates: ')
128                         print('u = ',p[0])
129                         print('v = ',p[1])
130                         t = t+1
131                         p_u = p_u + p[0]
132                         p_v = p_v + p[1]
133                 key = cv2.waitKey(1)
134                 if key == 27:
135                     print('while loop engaged')
136 #                    cv2.destroyAllWindows()
137                 if t == 0:
138                     pass
139                 else:
140                     U_e = int(p_u/t)
141                     V_e = int(p_v/t)
142                     cv2.rectangle(crop_img1,(U_e,V_e),(U_e+w,V_e+h),(color)
    ,3)
143 #                    cv2.imshow('detected', crop_img1)
144                     self.img_res = crop_img1
145                     break
146             if t != 0:
147                 break
148         try:
149             return U_e,V_e,w,h
150         except:
151             print('except')
152             a,b,c,d = [1,1,1,1]
153             return a,b,c,d
154 
155     def rename(self,U_start1,V_start1,U_stop1,V_stop1,U_start2,V_start2,
    U_stop2,V_stop2,w,h):
156         if V_start1 < V_start2:
157             # first detected start is lower
158             x1,y1,x2,y2 = [U_start1,V_start1,U_stop1,V_stop1]
159             U_start1,V_start1,U_stop1,V_stop1 = [U_start2,V_start2,U_stop2,
    V_stop2]
160             U_start2,V_start2,U_stop2,V_stop2 = [x1,y1,x2,y2]
161             U = list([U_start1,U_stop1,U_start2,U_stop2])
162             V = list([V_start1,V_stop1,V_start2,V_stop2])
163             if V_stop1 == 1 or V_stop2 == 1:
```

```
164                 W = w
165                 H = h
166             else:
167                 [w0,w1],[w2,w3] = w
168                 W = list([w2,w3,w0,w1])
169                 [h0,h1],[h2,h3] = h
170                 H = list([h2,h3,h0,h1])
171         else:
172             U = list([self.U0, self.U1, self.U2, self.U3])
173             V = list([self.V0, self.V1, self.V2, self.V3])
174             if V_stop1 == 1 or V_stop2 == 1:
175                 W = w
176                 H = h
177             else:
178                 [w0,w1],[w2,w3] = w
179                 W = list([w0,w1,w2,w3])
180                 [h0,h1],[h2,h3] = h
181                 H = list([h0,h1,h2,h3])
182             pass
183         return U,V,W,H
184 
185     def translator(self,U,V,tj,ign_u):
186         if self.Found == True:
187 #             U = U + self.ign_u-1000
188             w = self.ign_u-1000
189             U = [int(U[0]+w),int(U[1]+w),int(U[2]+w),int(U[3]+w)]
190             print U
191             q = self.tj-150
192 #             V = V + tj-150
193             V = [int(V[0]+q),int(V[1]+q),int(V[2]+q),int(V[3]+q)]
194             print V
195         else:
196 
197             U2 = int(U[2] + self.ign_u - 1000)
198             U3 = int(U[3] + self.ign_u - 1000)
199             V2 = V[2] + tj - 150
200             V3 = V[3] + tj - 150
201             U = [int(U[0]),int(U[1]),U2,U3]
202             V = [V[0],V[1],V2,V3]
203         return U, V
204 
205 
206     # Insert code above
207     def callback1(self,request):
208 
209         start = '/home/kristoffer/catkin_ws/src/wcm/Master/Code/start/'
210         self.loadImages(start)
211         imgs = self.loadedImages
212         end = '/home/kristoffer/catkin_ws/src/wcm/Master/Code/end/'
213         self.loadImages(end)
214         imge = self.loadedImages
215         # end move to detector part
216 
217         self.width = []
218         self.height = []
219 
220         [self.U0,self.V0,self.U1,self.V1,self.w,self.h,self.Found]=self.
    detectorstart(self.gray_img,self.col_img,imgs,imge,10,1,self.ign_u,self.
```

```
    tj,(0,255,0))
221         print('width',self.w,'height',self.h)
222 
223         print('Detecting second round: ')
224         [self.U2,self.V2,self.U3,self.V3,self.w,self.h,self.Found]=self.
    detectorstart(self.gray_img,self.col_img,imgs,imge,self.V0,self.U1,self.
    ign_u,self.tj,(0,255,0))
225         print('width',type(self.w),'height',self.h)
226 
227         if np.abs(self.U0-self.U2)<2 and np.abs(self.U1-self.U3)<2:
228             self.V_stop1 = 1
229 
230         U,V,W,H = self.rename(self.U0,self.V0,self.U1,self.V1,self.U2,self.
    V2,self.U3,self.V3,self.w,self.h)
231         print('U',U)
232         U,V = self.translator(U,V,self.tj,self.ign_u)
233         print('U',U)
234         self.U_0,self.U_1,self.U_2,self.U_3 = U[0],U[1],U[2],U[3]
235         self.V_0,self.V_1,self.V_2,self.V_3 = V[0],V[1],V[2],V[3]
236 
237         print('First weldgun position from: [',U[0],',',V[0],']to: [',U[1],
    ',',V[1],']')
238         print('Second weldgun position from: [',U[2],',',V[2],']to: [',U
    [3],',',V[3],']')
239 #         cv2.waitKey(0) &0xFF
240 #         cv2.destroyAllWindows()
241         self.im = self.bridge.cv2_to_imgmsg(self.img_res, "bgr8")
242         print('Sending image...')
243         crop_resp_col = imageResponse(self.im.width,self.im.height,self.im.
    is_bigendian,self.im.encoding,self.im.data)
244         print ('Image sent')
245         return crop_resp_col
246 
247 
248     def callback0(self,request):
249         print('Sending coordinates...')
250         print self.U_0
251         image_coordinates = img_coordResponse(self.U_0,self.U_1,self.U_2,
    self.U_3,self.V_0,self.V_1,self.V_2,self.V_3)
252         print ('Coordinates sent')
253         print('end of callback0')
254         return image_coordinates
255     def main(self):
256         ############################ Calling Crop Gray
    ################################
257         print "Waiting for crop image service"
258         rospy.wait_for_service("/crop_image_service")
259         try:
260             print('Found crop image service')
261             edge_client_node = rospy.ServiceProxy("/crop_image_service",
    image)
262             print("Request sent")
263             self.gray_resp = edge_client_node("Run Forrest, Run!")
264             print("Server responded")
265             self.gray_img = self.bridge.imgmsg_to_cv2(self.gray_resp,
    desired_encoding="passthrough")
266 #             cv2.imshow("service image",self.gray_img)
267 #             cv2.waitKey()
```

```
#             cv2.destroyAllWindows()
                print"does stuff!"
            except rospy.ServiceException, e:
                print "Service call failed: %s"%e
            ############################ Calling Crop Color
    ##############################
            print "Waiting for crop color image service"
            rospy.wait_for_service("/crop_col_image_service")
            try:
                print('Found crop color image service')
                crop_col_client = rospy.ServiceProxy("/crop_col_image_service",
    image)
                print("Request sent")
                self.col_resp = crop_col_client("file")
                print("Server responded")
                self.col_img = self.bridge.imgmsg_to_cv2(self.col_resp,
    desired_encoding="passthrough")
#             cv2.imshow("original image",self.col_img)
#             cv2.waitKey()
#             cv2.destroyAllWindows()
                print"does stuff!"
            except rospy.ServiceException, e:
                print "Service call failed: %s"%e

            ############################ Calling Crop Color
    ##############################
            print "Waiting tj_ign_u service"
            rospy.wait_for_service("/tj_ign_u")
            try:
                print('Found tj and ign_u service')
                twoint = rospy.ServiceProxy("/tj_ign_u",two_int)
                print("Request sent")
                tj_ign_u_resp = twoint()
                print("Server responded")
                self.tj = tj_ign_u_resp.tj
                print(self.tj)
                self.ign_u = tj_ign_u_resp.ign_u
                print(self.ign_u)
                print"does stuff!"
            except rospy.ServiceException, e:
                print "Service call failed: %s"%e
            ############################ Activating service
    ##############################
            # Sending image.srv result from client to service with callback
    function
            self.s = rospy.Service("/detected_image",image,self.callback1)
            rospy.loginfo('Feature detection Service is running')

            # Sending image.srv result from client to service with callback
    function
            self.sr = rospy.Service("/image_coord",img_coord,self.callback0)
            rospy.loginfo('Image coordinate service is running')

            rospy.spin()

if __name__ == '__main__':
    try:
            feature_detector_service = Feature_detector_service()
```

```
319         feature_detector_service.main()
320     except rospy.ROSInterruptException:
321         cv2.destroyAllWindows(0)
322         pass
```

A.2.5 Feature detection client

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  import rospy
5  import cv2
6  from cv_bridge import CvBridge
7  from custom_srvs.srv import image
8  from custom_srvs.srv import img_coord
9
10 class Detect_image_client():
11     def __init__(self):
12         self.bridge = CvBridge()
13
14     def main(self):
15         print "Waiting for detect service"
16         rospy.wait_for_service("/detected_image")
17         try:
18             print('Found service')
19             detected_image = rospy.ServiceProxy("/detected_image",image)
20             self.detect_resp = detected_image("Run Forrest, Run!")
21             print("Request sent")
22             self.detect_resp = detected_image("Run Forrest, Run!")
23             print("Server responded")
24             img = self.bridge.imgmsg_to_cv2(self.detect_resp,
    desired_encoding="passthrough")
25             cv2.imshow("service image",img)
26             cv2.waitKey()
27             cv2.destroyAllWindows()
28             print"does stuff!"
29 #            return self.crop_resp
30         except rospy.ServiceException, e:
31             print "Service call failed: %s"%e
32
33             print "Waiting for image_coord service"
34         rospy.wait_for_service("/image_coord")
35         try:
36             print('Found service')
37             detected_position = rospy.ServiceProxy("/image_coord",img_coord
    )
38             print("Request sent")
39             self.pos_resp = detected_position()
40             print("Server responded")
41             print self.pos_resp
42             print"does stuff!"
43 #            return self.crop_col_resp
44         except rospy.ServiceException, e:
45             print "Service call failed: %s"%e
46
47 if __name__ == '__main__':
48     try:
```

```
49          detect_image_client = Detect_image_client()
50          detect_image_client.main()
51      except rospy.ROSInterruptException:
52          cv2.destroyAllWindows(0)
53          pass
```

A.3 CMakeList for services

```
1  cmake_minimum_required(VERSION 2.8.3)
2  project(feature_detector)
3
4  ## Compile as C++11, supported in ROS Kinetic and newer
5  # add_compile_options(-std=c++11)
6
7  ## Find catkin macros and libraries
8  ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9  ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   OpenCV
12   custom_srvs
13   cv_bridge
14   image_transport
15   message_generation
16   rospy
17   sensor_msgs
18   std_msgs
19 )
20
21 ## System dependencies are found with CMake's conventions
22 # find_package(Boost REQUIRED COMPONENTS system)
23
24
25 ## Uncomment this if the package has a setup.py. This macro ensures
26 ## modules and global scripts declared therein get installed
27 ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
28 # catkin_python_setup()
29
30 ################################################
31 ## Declare ROS messages, services and actions ##
32 ################################################
33
34 ## To declare and build messages, services or actions from within this
35 ## package, follow these steps:
36 ## * Let MSG_DEP_SET be the set of packages whose message types you use in
37 ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
38 ## * In the file package.xml:
39 ##   * add a build_depend tag for "message_generation"
40 ##   * add a build_depend and a exec_depend tag for each package in
   MSG_DEP_SET
41 ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled
   in
42 ##     but can be declared for certainty nonetheless:
43 ##     * add a exec_depend tag for "message_runtime"
44 ## * In this file (CMakeLists.txt):
45 ##   * add "message_generation" and every package in MSG_DEP_SET to
46 ##     find_package(catkin REQUIRED COMPONENTS ...)
47 ##   * add "message_runtime" and every package in MSG_DEP_SET to
```

```
48 ##      catkin_package(CATKIN_DEPENDS ...)
49 ##    * uncomment the add_*_files sections below as needed
50 ##      and list every .msg/.srv/.action file to be processed
51 ##    * uncomment the generate_messages entry below
52 ##    * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES
       ...)
53
54 ## Generate messages in the 'msg' folder
55 # add_message_files(
56 #   FILES
57 #   Message1.msg
58 #   Message2.msg
59 # )
60
61 ## Generate services in the 'srv' folder
62 # add_service_files(
63 #   FILES
64 #   Service1.srv
65 #   Service2.srv
66 # )
67
68 ## Generate actions in the 'action' folder
69 # add_action_files(
70 #   FILES
71 #   Action1.action
72 #   Action2.action
73 # )
74
75 ## Generate added messages and services with any dependencies listed here
76 # generate_messages(
77 #   DEPENDENCIES
78 #   std_msgs  # Or other packages containing msgs
79 # )
80
81 ################################################
82 ## Declare ROS dynamic reconfigure parameters ##
83 ################################################
84
85 ## To declare and build dynamic reconfigure parameters within this
86 ## package, follow these steps:
87 ## * In the file package.xml:
88 ##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
89 ## * In this file (CMakeLists.txt):
90 ##   * add "dynamic_reconfigure" to
91 ##     find_package(catkin REQUIRED COMPONENTS ...)
92 ##   * uncomment the "generate_dynamic_reconfigure_options" section below
93 ##     and list every .cfg file to be processed
94
95 ## Generate dynamic reconfigure parameters in the 'cfg' folder
96 # generate_dynamic_reconfigure_options(
97 #   cfg/DynReconf1.cfg
98 #   cfg/DynReconf2.cfg
99 # )
100
101 ##################################
102 ## catkin specific configuration ##
103 ##################################
104 ## The catkin_package macro generates cmake config files for your package
```

```
105 ## Declare things to be passed to dependent projects
106 ## INCLUDE_DIRS: uncomment this if your package contains header files
107 ## LIBRARIES: libraries you create in this project that dependent projects
    also need
108 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
109 ## DEPENDS: system dependencies of this project that dependent projects
    also need
110 catkin_package(
111 #  INCLUDE_DIRS include
112 #  LIBRARIES feature_detector
113   CATKIN_DEPENDS
114   cv_bridge
115   rospy
116   sensor_msgs
117   std_msgs
118   custom_srvs
119   #  DEPENDS system_lib
120 )
121
122 ###########
123 ## Build ##
124 ###########
125
126 ## Specify additional locations of header files
127 ## Your package locations should be listed before other locations
128 include_directories(
129 # include
130   ${catkin_INCLUDE_DIRS}
131 )
132
133 ## Declare a C++ library
134 # add_library(${PROJECT_NAME}
135 #   src/${PROJECT_NAME}/feature_detector.cpp
136 # )
137
138 ## Add cmake target dependencies of the library
139 ## as an example, code may need to be generated before libraries
140 ## either from message generation or dynamic reconfigure
141 # add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})
142
143 ## Declare a C++ executable
144 ## With catkin_make all packages are built within a single CMake context
145 ## The recommended prefix ensures that target names across packages don't
    collide
146 # add_executable(${PROJECT_NAME}_node src/feature_detector_node.cpp)
147
148 ## Rename C++ executable without prefix
149 ## The above recommended prefix causes long target names, the following
    renames the
150 ## target back to the shorter version for ease of user use
151 ## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg
    someones_pkg_node"
152 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node
    PREFIX "")
153
154 ## Add cmake target dependencies of the executable
155 ## same as for the library above
```

```
156 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
        ${catkin_EXPORTED_TARGETS})
157
158 ## Specify libraries to link a library or executable target against
159 # target_link_libraries(${PROJECT_NAME}_node
160 #   ${catkin_LIBRARIES}
161 # )
162
163 #############
164 ## Install ##
165 #############
166
167 # all install targets should use catkin DESTINATION variables
168 # See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html
169
170 ## Mark executable scripts (Python etc.) for installation
171 ## in contrast to setup.py, you can choose the destination
172 # install(PROGRAMS
173 #   scripts/my_python_script
174 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
175 # )
176
177 ## Mark executables and/or libraries for installation
178 # install(TARGETS ${PROJECT_NAME} ${PROJECT_NAME}_node
179 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
180 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
181 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
182 # )
183
184 ## Mark cpp header files for installation
185 # install(DIRECTORY include/${PROJECT_NAME}/
186 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
187 #   FILES_MATCHING PATTERN "*.h"
188 #   PATTERN ".svn" EXCLUDE
189 # )
190
191 ## Mark other files for installation (e.g. launch and bag files, etc.)
192 # install(FILES
193 #   # myfile1
194 #   # myfile2
195 #   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
196 # )
197
198 #############
199 ## Testing ##
200 #############
201
202 ## Add gtest based cpp test target and link libraries
203 # catkin_add_gtest(${PROJECT_NAME}-test test/test_feature_detector.cpp)
204 # if(TARGET ${PROJECT_NAME}-test)
205 #   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
206 # endif()
207
208 ## Add folders to be run by python nosetests
209 # catkin_add_nosetests(test)
```

A.4 package file services

```
<?xml version="1.0"?>
<package format="2">
  <name>feature_detector</name>
  <version>0.0.0</version>
  <description>The feature_detector package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="kristoffer@todo.todo">kristoffer</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/feature_detector</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use exec_depend for packages you need at runtime: -->
  <!--   <exec_depend>message_runtime</exec_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <!-- Use doc_depend for packages you need only for building documentation: -->
```

```
  <!--   <doc_depend>doxygen</doc_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>cv_bridge</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>custom_srvs</build_depend>
  <build_export_depend>cv_bridge</build_export_depend>
  <build_export_depend>rospy</build_export_depend>
  <build_export_depend>sensor_msgs</build_export_depend>
  <build_export_depend>custom_srvs</build_export_depend>
  <exec_depend>cv_bridge</exec_depend>
  <exec_depend>rospy</exec_depend>
  <exec_depend>sensor_msgs</exec_depend>
  <exec_depend>std_msgs</exec_depend>
  <exec_depend>message_runtime</exec_depend>


  <!-- The export tag contains other, unspecified, tags -->
  <export>
    <!-- Other tools can request additional information be placed here -->

  </export>
</package>
```

A.5 Thermal measurements with computer vision

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import cv2
import numpy as np

path = "/home/kristoffer/catkin_ws/src/Thesis/pipe_images/"
filename = "thermal.png"
impath = path+filename
thermal_img = cv2.imread(impath,0)
thermal_img = cv2.resize(thermal_img,(1366,768))
cv2.imshow('bluuh',thermal_img)
cv2.waitKey(0)&0xFF
cv2.destroyAllWindows()

delta = 200-20 # change in temperature for image
scale = float(delta)/float(255)
twenties = np.ones((768,1366))*20
imgfloat = thermal_img.astype(np.float32)
imgfloat = imgfloat * scale
imgint = imgfloat.astype(np.uint8)
imgint = imgint+twenties
imgint = imgint.astype(np.uint8) #compansated thermal image celsius
print imgint
cv2.imshow('bluuh',imgint)
cv2.waitKey(0)&0xFF
cv2.destroyAllWindows()
crop_img = imgint[300:530,400:600]
pixelsum = 0
maxvalue = 0
```

```
row = 0
for array in crop_img:
    row +=1
    col = 0
    for pixel in array:
        pixelsum += pixel
        col +=1
        if pixel > maxvalue:
            maxvalue = pixel
            maxrow = row
            maxcol = col

height, width = crop_img.shape[:2]
pixels = width*height
print maxvalue, "at location", maxrow ,"x",maxcol
print pixels
print pixelsum
avg_temp = pixelsum/pixels
print avg_temp
cv2.imshow('cropped',crop_img)
cv2.waitKey(0)&0xFF
cv2.destroyAllWindows()

bgr_img = cv2.cvtColor(thermal_img, cv2.COLOR_GRAY2BGR)
cv2.rectangle(bgr_img,(400,300),(600,530),(0,0,255),3)
cv2.rectangle(bgr_img,(400+maxcol-5,300+maxrow-5),(405+maxcol,305+maxrow)
    ,(255,0,0),3)

font = cv2.FONT_HERSHEY_SIMPLEX
u1 = 330
v1 = 610
string = str(avg_temp)
string2 = str(maxvalue)
cv2.putText(bgr_img,"Avrage temperature: "+string+" C", (100,100), font, 1,
     (0,0,255))
cv2.putText(bgr_img,"Max temperature: "+string2+" C", (100,150), font, 1,
    (255,0,0))


cv2.imshow('with rectangle',bgr_img)
cv2.waitKey(0)&0xFF
cv2.destroyAllWindows()
```

A.6 Service message files

A.6.1 image.srv file

```
string camera
---
uint32 width
uint32 height
uint8 is_bigendian
string encoding
uint8[] data
```

A.6.2 two_int.srv file

```
1
2 ---
3 uint32 tj
4 uint32 ign_u
```

A.6.3 image_coord.srv file

```
1
2 ---
3 int32 U0
4 int32 U1
5 int32 U2
6 int32 U3
7 int32 V0
8 int32 V1
9 int32 V2
10 int32 V3
```

A.7 CMakeList.txt file for custom messages

```
1 cmake_minimum_required(VERSION 2.8.3)
2 project(custom_srvs)
3
4 ## Compile as C++11, supported in ROS Kinetic and newer
5 # add_compile_options(-std=c++11)
6
7 ## Find catkin macros and libraries
8 ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9 ## is used, also find other catkin packages
10 find_package(catkin REQUIRED COMPONENTS
11   std_msgs
12   sensor_msgs
13   message_generation
14 )
15
16 ## System dependencies are found with CMake's conventions
17 # find_package(Boost REQUIRED COMPONENTS system)
18
19
20 ## Uncomment this if the package has a setup.py. This macro ensures
21 ## modules and global scripts declared therein get installed
22 ## See http://ros.org/doc/api/catkin/html/user_guide/setup_dot_py.html
23 # catkin_python_setup()
24
25 ################################################
26 ## Declare ROS messages, services and actions ##
27 ################################################
28
29 ## To declare and build messages, services or actions from within this
30 ## package, follow these steps:
31 ## * Let MSG_DEP_SET be the set of packages whose message types you use in
32 ##   your messages/services/actions (e.g. std_msgs, actionlib_msgs, ...).
33 ## * In the file package.xml:
```

```
34 ##   * add a build_depend tag for "message_generation"
35 ##   * add a build_depend and a exec_depend tag for each package in
       MSG_DEP_SET
36 ##   * If MSG_DEP_SET isn't empty the following dependency has been pulled
       in
37 ##     but can be declared for certainty nonetheless:
38 ##     * add a exec_depend tag for "message_runtime"
39 ## * In this file (CMakeLists.txt):
40 ##   * add "message_generation" and every package in MSG_DEP_SET to
41 ##     find_package(catkin REQUIRED COMPONENTS ...)
42 ##   * add "message_runtime" and every package in MSG_DEP_SET to
43 ##     catkin_package(CATKIN_DEPENDS ...)
44 ##   * uncomment the add_*_files sections below as needed
45 ##     and list every .msg/.srv/.action file to be processed
46 ##   * uncomment the generate_messages entry below
47 ##   * add every package in MSG_DEP_SET to generate_messages(DEPENDENCIES
       ...)
48
49 ## Generate messages in the 'msg' folder
50 # add_message_files(
51 #   FILES
52 #   Message1.msg
53 #   Message2.msg
54 # )
55
56 ## Generate services in the 'srv' folder
57  add_service_files(
58    FILES
59    custom_service.srv
60    image.srv
61    image_to_image.srv
62    two_int.srv
63    img_coord.srv
64  )
65
66 ## Generate actions in the 'action' folder
67 # add_action_files(
68 #   FILES
69 #   Action1.action
70 #   Action2.action
71 # )
72
73 ## Generate added messages and services with any dependencies listed here
74  generate_messages(
75    DEPENDENCIES
76    std_msgs
77    sensor_msgs
78  )
79
80 ################################################
81 ## Declare ROS dynamic reconfigure parameters ##
82 ################################################
83
84 ## To declare and build dynamic reconfigure parameters within this
85 ## package, follow these steps:
86 ## * In the file package.xml:
87 ##   * add a build_depend and a exec_depend tag for "dynamic_reconfigure"
88 ## * In this file (CMakeLists.txt):
```

```
89 ##   * add "dynamic_reconfigure" to
90 ##     find_package(catkin REQUIRED COMPONENTS ...)
91 ##   * uncomment the "generate_dynamic_reconfigure_options" section below
92 ##     and list every .cfg file to be processed
93
94 ## Generate dynamic reconfigure parameters in the 'cfg' folder
95 # generate_dynamic_reconfigure_options(
96 #   cfg/DynReconf1.cfg
97 #   cfg/DynReconf2.cfg
98 # )
99
100 ###################################
101 ## catkin specific configuration ##
102 ###################################
103 ## The catkin_package macro generates cmake config files for your package
104 ## Declare things to be passed to dependent projects
105 ## INCLUDE_DIRS: uncomment this if your package contains header files
106 ## LIBRARIES: libraries you create in this project that dependent projects
    also need
107 ## CATKIN_DEPENDS: catkin_packages dependent projects also need
108 ## DEPENDS: system dependencies of this project that dependent projects
    also need
109 catkin_package(
110 #  INCLUDE_DIRS include
111 #  LIBRARIES custom_srvs
112   CATKIN_DEPENDS rospy
113 #  DEPENDS system_lib
114 )
115
116 ###########
117 ## Build ##
118 ###########
119
120 ## Specify additional locations of header files
121 ## Your package locations should be listed before other locations
122 include_directories(
123 # include
124 # ${catkin_INCLUDE_DIRS}
125 )
126
127 ## Declare a C++ library
128 # add_library(${PROJECT_NAME}
129 #   src/${PROJECT_NAME}/custom_srvs.cpp
130 # )
131
132 ## Add cmake target dependencies of the library
133 ## as an example, code may need to be generated before libraries
134 ## either from message generation or dynamic reconfigure
135 # add_dependencies(${PROJECT_NAME} ${${PROJECT_NAME}_EXPORTED_TARGETS} ${
    catkin_EXPORTED_TARGETS})
136
137 ## Declare a C++ executable
138 ## With catkin_make all packages are built within a single CMake context
139 ## The recommended prefix ensures that target names across packages don't
    collide
140 # add_executable(${PROJECT_NAME}_node src/custom_srvs_node.cpp)
141
142 ## Rename C++ executable without prefix
```

```
143 ## The above recommended prefix causes long target names, the following
    renames the
144 ## target back to the shorter version for ease of user use
145 ## e.g. "rosrun someones_pkg node" instead of "rosrun someones_pkg
    someones_pkg_node"
146 # set_target_properties(${PROJECT_NAME}_node PROPERTIES OUTPUT_NAME node
    PREFIX "")
147
148 ## Add cmake target dependencies of the executable
149 ## same as for the library above
150 # add_dependencies(${PROJECT_NAME}_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
     ${catkin_EXPORTED_TARGETS})
151
152 ## Specify libraries to link a library or executable target against
153 # target_link_libraries(${PROJECT_NAME}_node
154 #   ${catkin_LIBRARIES}
155 # )
156
157 #############
158 ## Install ##
159 #############
160
161 # all install targets should use catkin DESTINATION variables
162 # See http://ros.org/doc/api/catkin/html/adv_user_guide/variables.html
163
164 ## Mark executable scripts (Python etc.) for installation
165 ## in contrast to setup.py, you can choose the destination
166 # install(PROGRAMS
167 #   scripts/my_python_script
168 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
169 # )
170
171 ## Mark executables and/or libraries for installation
172 # install(TARGETS ${PROJECT_NAME} ${PROJECT_NAME}_node
173 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
174 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
175 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
176 # )
177
178 ## Mark cpp header files for installation
179 # install(DIRECTORY include/${PROJECT_NAME}/
180 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
181 #   FILES_MATCHING PATTERN "*.h"
182 #   PATTERN ".svn" EXCLUDE
183 # )
184
185 ## Mark other files for installation (e.g. launch and bag files, etc.)
186 # install(FILES
187 #   # myfile1
188 #   # myfile2
189 #   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
190 # )
191
192 #############
193 ## Testing ##
194 #############
195
196 ## Add gtest based cpp test target and link libraries
```

```
# catkin_add_gtest(${PROJECT_NAME}-test test/test_custom_srvs.cpp)
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)
```

A.8 package.xml file for custom service messages

```
<?xml version="1.0"?>
<package format="2">
  <name>custom_srvs</name>
  <version>0.0.0</version>
  <description>The custom_srvs package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example: -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="kristoffer@todo.todo">kristoffer</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/custom_srvs</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use depend as a shortcut for packages that are both build and exec dependencies -->
  <!--   <depend>roscpp</depend> -->
  <!--   Note that this is equivalent to the following: -->
  <!--   <build_depend>roscpp</build_depend> -->
  <!--   <exec_depend>roscpp</exec_depend> -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use build_export_depend for packages you need in order to build against this package: -->
  <!--   <build_export_depend>message_generation</build_export_depend> -->
```

```
43   <!-- Use buildtool_depend for build tool packages: -->
44   <!--   <buildtool_depend>catkin</buildtool_depend> -->
45   <!-- Use exec_depend for packages you need at runtime: -->
46   <!--   <exec_depend>message_runtime</exec_depend> -->
47   <!-- Use test_depend for packages you need only for testing: -->
48   <!--   <test_depend>gtest</test_depend> -->
49   <!-- Use doc_depend for packages you need only for building documentation
       : -->
50   <!--   <doc_depend>doxygen</doc_depend> -->
51   <buildtool_depend>catkin</buildtool_depend>
52   <build_depend>rospy</build_depend>
53   <build_depend>sensor_msgs</build_depend>
54   <build_depend>std_msgs</build_depend>
55   <build_depend>message_generation</build_depend>
56   <exec_depend>rospy</exec_depend>
57   <exec_depend>sensor_msgs</exec_depend>
58   <exec_depend>std_msgs</exec_depend>
59   <exec_depend>message_runtime</exec_depend>
60
61
62   <!-- The export tag contains other, unspecified, tags -->
63   <export>
64     <!-- Other tools can request additional information be placed here -->
65
66   </export>
67 </package>
```

A.9 Launch files

A.9.1 launch file camera_service

```
<launch>
    <node
    pkg = "camera_service"
    type = "camera_service_node.py"
    name = "camera_service_node"
    output = "screen">
    </node>
</launch>
```

A.9.2 launch file edges image processing

```
<launch>
    <node
    pkg = "edged_image_processing"
    type = "edged_image_processing.py"
    name = "edged_image_processing_node"
    output = "screen">
    </node>
</launch>
```

A.9.3 launch file crop image

```
<launch>
    <node
    pkg = "crop_image"
    type = "crop_image.py"
    name = "crop_image_node"
    output = "screen">
    </node>
</launch>
```

A.9.4 launch file feature detection

```
<launch>
    <node
    pkg = "feature_detector"
    type = "feature_detection.py"
    name = "feature_detector_node"
    output = "screen">
    </node>

    <include
    file="/home/kristoffer/catkin_ws/src/Thesis/camera_service/launch/
    camera_service_launch.launch">
    </include>
    <include
    file="/home/kristoffer/catkin_ws/src/Thesis/edged_image_processing/
    launch/edged_image_processing.launch">
    </include>
    <include
```

```
16      file="/home/kristoffer/catkin_ws/src/Thesis/crop_image/launch/
        crop_image.launch">
17      </include>
18  </launch>
```

Kristoffer Hermansen

Automating parts of the hardbanding process using computer vision and KUS

NTNU
Kunnskap for en bedre verden