UiA University of Agder

Pose Estimation of Drilling Pipe Using a 3D-Sensor Network

Erlend Buan

Supervisors:
Atle Aalerud
Joacim Dybedal

University of Agder
Department of Engineering Sciences
Spring 2019

Abstract

In this thesis, a pose estimation system for drill pipe-ends has been made. The system utilizes 6 sensor nodes containing a Kinect v2 RGB-D camera in addition to a Jetson TX2 computer. Pipe detection is done on 2D images from each camera using Tiny-YOLOv3. A pair of point clouds for the box-end and pin-end are made based on these detections, in addition to the depth map which the Kinect provides. Further segmentation is done, and pose estimation using ICP is performed.
When the pipes are placed on the ground and placed vertical at a minimum distance of 5.5m away from the sensors, the system achieves a mean positional accuracy of $\approx 2.6cm$, $\approx 2.0cm$ and $\approx 6.5cm$. Further, the mean rotational deviation is $\approx 0.8^\circ$, $\approx 3.5^\circ$ around the x- and y-axis.

Preface

This Master's thesis was done as a part of the Mechatronics Master's Programme at the University of Agder. Work was conducted in the period of January 7 up to May 24 in 2019. This has been a challenging and very interesting project. Prior to this thesis, I had a basic understanding of C++ and Python, including next to no experience with computer vision. Work done through the thesis has given me a much deeper understanding and great interest for computer vision systems, in addition to programming in general.

I would like to thank my supervisors Joacim Dybedal and Atle Aalerud for advising me through the project, who also reviewed my report prior to the delivery. An extra thanks to Atle, who had to endure a lot of questions during a change of direction in the thesis. I am also thankful for the university who provided the necessary equipment in addition to the opportunity to work with computer vision.

Erlend Buan

Grimstad
May 23, 2019

Contents

List of Figures

List of Tables

Abbreviations

$AMCW - ToF$	Amplitude Modulated Continuous Wave Time-Of-Flight
CAD	Computer Aided Design
CNN	Convolutional Neural Network
FCL	Fully Connected Layers
$FIFO$	First In, First Out
$FPFH$	Fast Point Feature Histogram
ICP	Iterative Closest Point
IOU	Intersection Over Union
IRL	Industrial Robotics Lab
MLP	Multi-Layer Perceptron
PCL	Point Cloud Library
PFH	Point Feature Histogram
$RANSAC$	RANdom SAmple Consensus
RGB	Red Green Blue
ROS	Robot Operating System
$SPFH$	Simplified Point Feature Histogram
SVD	Singular Value Decomposition
ToF	Time-Of-Flight
$YOLO$	You Only Look Once

1 Introduction

In this thesis, a pose estimation system for drill pipes is presented. This chapter will introduce the reader to the current state of pipe handling on offshore drilling ships, as well as present an area of improvement. Following, the requirements and limitations of the system, in addition to the report structure is shown.

1.1 Background and Motivation

Computer vision is an important part in many fully automated systems. This may be pose estimation for objects that are to be manipulated by a robot, visual inspection of products during production, and localization of a mobile robot in a scene.
As with many other industries, the oil sector aims to automate its processes to increase the efficiency. This is also the case for offshore drilling, where most of the drilling process is automated. On modern drilling ships, the drilling and storage/handling of the pipes are done by large machinery, in an almost fully automated process. The pipes are stored in large vertical stores and transported by a catwalk, a large horizontal machine shuttling the pipes to and from the storage(can be seen on Fig. 2). The Hydraracker is responsible for handling the pipe between the catwalk and pipe assembly/disassembly.
On Fig. 2, such a process is shown, although in disassembly mode. The Hydraracker lifts the pipe segment after an iron roughneck(see Fig. 1b) has detached it from the string, and lays it down onto the catwalk which transports it to an storage.
However, there are still some parts of the process that requires manual input from the workers. Currently, the worker has to confirm the position of the pipe before the HydraRacker picks it up. Similarly, when a pipe section is ready to be attached/detached to/from the drill string, the worker has to confirm the position of the pipes before the iron roughneck proceeds with connecting/disconnecting.

There is a desire to remove this manual position verification from the loop, and have a computer vision system to accurately estimate the pipe pose, for handling and assembly/disassembly of the pipe string.

(a) Traditional roughneck connecting the box-end(lower part of the pipe joint) and the pin-end(upper part of the pipe joint).
Source: Image from [16]

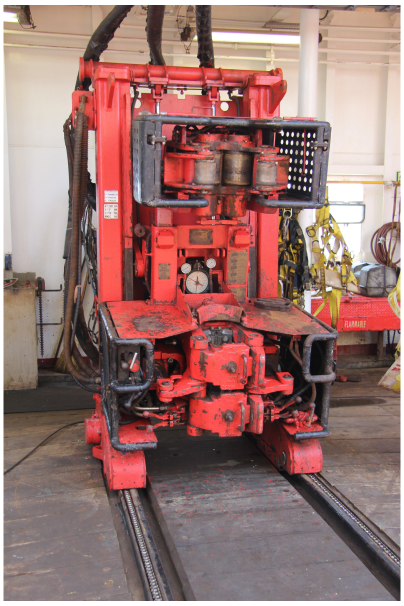

(b) Iron roughneck, used for connecting/disconnecting drill pipes
Source: Image from [17]

Figure 1: Traditional roughneck and iron roughneck

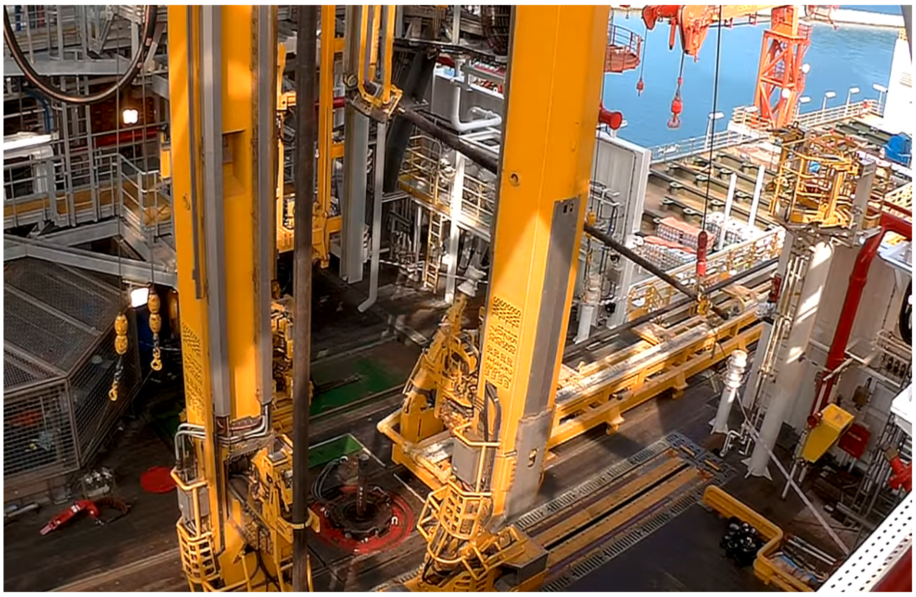


Figure 2: Hydraracker(large vertical machine) placing the pipe segment onto the catwalk after being disconnected from the drill string by an iron roughneck. Source: Image from [1].

1.2 Problem Statement and Limitations

The objective of this thesis is to create a ROS node for automatic detection and pose estimation of the pipe-ends(see Fig. 1a for the pipe-ends), bypassing the need for visual confirmation of an operator. The master thesis problem statement are as follows:

1. Evaluate different detection techniques.
2. Create a ROS node that locates and publishes the pose of the pipe-ends at a medium distance.
3. Demonstrate the system in the Mechatronics Innovation Lab(MIL).

Additional requirements of the system is to use a sensor node network. A total of six sensor nodes are provided, where each node contains a Jetson TX2 computer in addition to a Kinectv2 RGB-D camera. A proof of concept system is to made with pose estimation of stationary pipe-ends.

1.3 Structure of the Report

- In Chapter 2, different state-of-the-art machine learning and model estimator algorithms are presented. Further, theory behind the algorithms and techniques used in the system is explained. This includes basic communication between the cameras and machines using ROS, machine learning detectors using YOLOv3, point cloud creation and alignment using RANSAC and ICP.
- Chapter 3 shows the implementation of the system. Here different aspects of the system are presented which includes: lab-setup, communication between the cameras and computers, the YOLOv3 object detector, point cloud creation, segmentation and pose estimation using ICP.
- Chapter 4 presents the experimental results of the pose estimator and the object detector in two different test scenarios.
- In Chapter 5, there is a discussion regarding the accuracy of the pose estimation, system speed, and shortcomings of the system.
- In Chapter 6, a conclusion of the project is held, along with different proposals that improve the system's speed and robustness.

2 Background Theory

In this chapter a theoretical background for the techniques used in this thesis is presented. A fundamental overview of ROS is first shown, followed by machine learning with convolutional neural network and YOLO, at last point cloud construction and pose estimation techniques are presented.

2.1 Robot Operating System

Robot operating system(ROS) is framework for writing robot software across different robotic platforms. Due to varying tasks and environments, the task of making general-purpose software is hard. ROS aims to solve this and to make collaboration between different robotic disciplines easier. This section will not delve deep into everything ROS encompasses, but will focus on the parts used in this thesis.

2.1.1 Nodes and Master

Every robotic system using ROS contains something called nodes. A node is a fundamental building block of ROS and multiple nodes build up the ROS network. Every node contains some code to perform a task of the user's desire. The nodes can communicate with each other by sending/receiving messages to another node. [18]
However, in order for the nodes to receive or send messages, a master is required. The master has to register the publishing(sending) and subscribing(receiving) nodes in addition to the message type. Once the nodes are registered with the master, the information transfer can happen. [19]

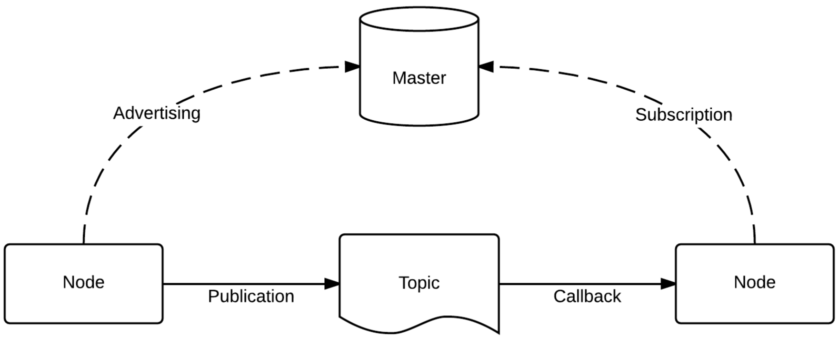


Figure 3: Interaction between master and nodes in message transfers.
Source: Image from [2]

2.1.2 Messages and Topics

A ROS message is a data structure that contains the information to be exchanged between the different nodes. Messages can be user-defined, or pre-defined, obtained from one of the many message libraries which exists. In ROS, sending and receiving messages are done by publishing the message on named buses called topics. The publishing node notifies the master that a message is to be published onto a topic. The receiving node then gets notified by the master that a message on the topic is ready to be subscribed, in which case the node will subscribe to the topic and receive the message. [20][21]

2.1.3 Publishing

After the publishing node notifies the master, it connects both the sending and receiving nodes and creates a peer-to-peer connection. A publishing queue has to be specified in order for the node to publish. When the publishing node is sending messages to a topic, the messages are put into a publisher queue, and is then sent to the subscriber for processing. If the publishing node is sending faster than the subscribing node is able to receive, the publisher will drop the oldest messages in the queue, in a FIFO(first in first out) manner. [22]

2.1.4 Subscribing and Synchronization

Subscribing in ROS is done through a callback function and a spinner. A spinner will call all the callbacks contained in a callback queue, which in this case is a queue of incoming messages. The callback function is the message handler and is attached to the subscriber. The callback function is called each time the spinner spins, and receives the incoming messages in addition to performing tasks set by the user. When subscribing to multiple topics, a callback function has to be created for each of the topics. Unfortunately, this is impractical and can be solved by using a synchronizer to create a common callback function for all the subscribers. Depending on the data used in the system, a synchronization policy has to be chosen. The messages may contain a timestamp that indicates when it was published, and this is used when choosing the synchronization type. In ROS there are two types of synchronization policy, ExactTime and ApproximateTime. The first policy requires the messages to have equal timestamps and is useful for systems that requires synchronized data from multiple sensors. The latter policy allow for a small deviation in timestamp value when receiving the messages.[23]

2.1.5 Spinners

The function of a spinner in ROS is to call the callback function. The spinner can be considered as a loop and will continuously call for the callback function. In ROS there are multiple spinners depending on the system requirement. The simplest spinner is a single threaded process, which means everything in the program has to be run inside the callback function on a single thread. For more demanding systems, a single thread is not sufficient. Therefore, ROS provides a solution for multi-threaded applications. One of these methods is something called an Asynchronous Spinner. This method allows for the use of multiple threads, where a single thread is reserved for the callback function. [24]

2.2 Machine Learning

Machine learning is an emerging tool for various disciplines, and especially computer vision. This section will focus on some of the fundamentals in machine learning followed by general Convolutional Neural Network using images. Further, different state-of-the-art machine learning networks for point clouds and 2D images are presented. Lastly, a detailed explanation of YOLOv3 will be held.

2.2.1 Basics

Machine learning revolves around estimating a function that predicts based on the input data. This fundamental predictive function can be estimated fairly well, although there will always be an error due to all the variables our data-set cannot capture. The procedure of mapping the input to the output is as follows,

$$Y = f(x) + \varepsilon \tag{1}$$

where,
Y is the prediction.
f is the predictive funtion.
x is the input data.
ε is the irreducible error.

Although the prediction process always has an error due to the limited span of training data, there is a lot to gain by improving the function estimation based on the available data. [25]
All neural networks are built up by artificial neurons. An artificial neuron is a mathematical model based on a biological neuron. The biological neuron will not be explained here, but the mathematical model describing an artificial neuron is as follows:

$$y_k = \varphi \sum_{i=1}^{m} w_i x_i \tag{2}$$

Where,
w_i is the weight.
x_i is the input signal.
φ is the activation function.

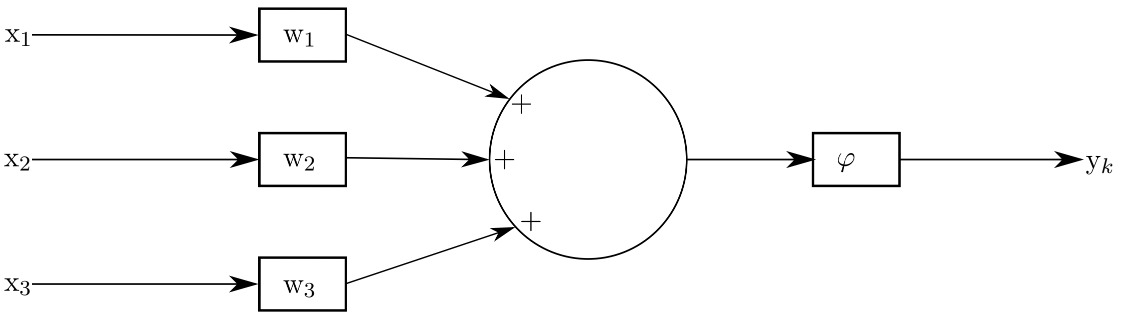


Figure 4: Artificial neuron.

Although this particular neuron shown in Fig. 4 only has three inputs, the number of inputs can vary. [26]

After the inputs are multiplied with their respective weights and summed up, the output is passed to an activation function. There are several different types of activation functions: linear, step, rectifier as well as nonlinear functions. The linear activation function is just equal to unity and is not suitable for complex tasks using data types such as images and audio due to the non-linear nature of these data types.
The step function is useful in binary classification due to its two values: 0, 1. If the input value is over a threshold, the output value is 1, and 0 if it is under. However this activation function limits the classification down to two classes. In addition, the step function as well as the linear type can not be used with backpropagation, an important learning technique in machine learning, which makes these activation functions impractical.
The rectified linear unit(ReLu) is the most common activation function used in modern neural networks. Because of its non-linearity, it is better fitted for image detection and other complex tasks. The function is $R(x) = max(0, x)$, if $x < 0 \Rightarrow R(x) = 0$ and if $x > 0 \Rightarrow R(x) = 0$.
One important property of this function is that it avoids the vanishing gradient problem. [27]

A simple neural network can be a collection of neurons placed in different layers called fully connected layers(FCL). Each neuron in the layer are connected to the neurons in the preceding and succeeding layer. In Fig. 5 it is shown a network with an input and output layer, in addition to a single hidden layer. Often there are multiple hidden layers in the network. The data is fed into the input layer, while the output layer presents the predictions. [28]

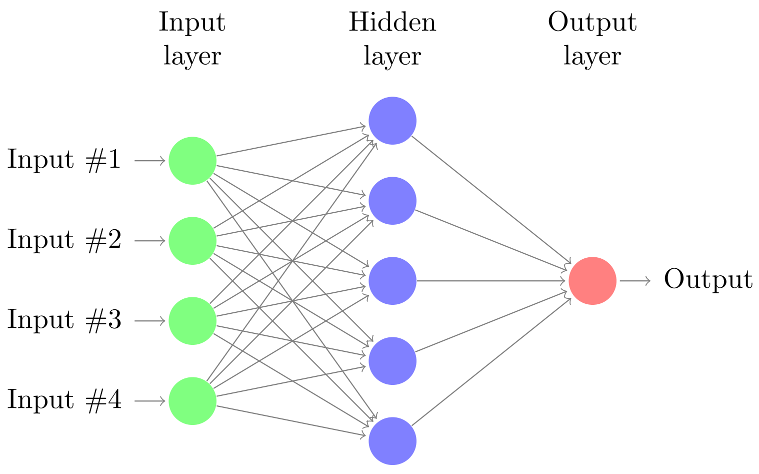


Figure 5: Fully connected layers.
Source: Image from [3]

When the computer estimates the predictive function, it uses a loss function to determine its performance. Gradient descent is then used to reduce the error and improving the predictive model. When optimizing with gradient descent, partial derivatives for all the layers are calculated by backpropagation which gives the direction to the loss function's nearest local minimum. The weights are then updated and the loss is found yet again. This is continued until the network is trained, hopefully giving good predictions.

2.2.2 Convolutional Neural Network

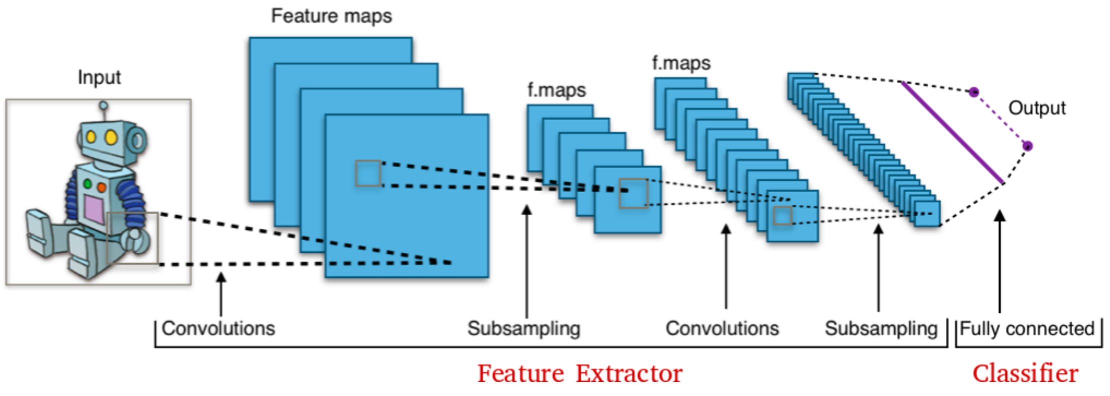


Figure 6: Convolutional Neural Network.
Source: Adapted from [4]

Convolutional neural network(CNN) is a powerful and versatile tool used in many different fields, however this section will focus on CNNs in context of image classification and localization. In short, the network extracts features in an image by using convolution and passes these feature maps to a classifier, which often is built up by a series of fully connected layers(FCL). The classifier learns the features of the objects of interest in the image, and does the classification and localization during inference. In Fig. 6 a typical CNN is shown, with several convolutional layers followed by a set of fully connected layers.

Convolutional layer
In the convolutional layer, filters are applied to each image. The filter can be thought of as a neuron. It is multiplying its weights with a spatial area of the input image, followed by summation and added non-linearities with an activation function, similarly to the neuron explained in Sec. 2.2.1. [29]
This filter can either be a convolution or a correlation filter. Contrary to the name, most CNNs uses correlation instead of convolution, although both give the same result in the end. In image correlation, each element of the filter is multiplied with the corresponding element of the image and summed up. This operation does not work at the edges of the image, due to the filter has to cover the image with all its elements. This results in a reduction of dimensions.

$$\underset{\text{Image}}{\begin{array}{|c|c|c|c|c|c|}\hline 0&0&1&1&1&1\\\hline 1&0&1&0&1&1\\\hline 1&1&1&0&1&0\\\hline 1&1&1&1&1&0\\\hline 0&1&1&0&0&0\\\hline 1&1&1&0&1&1\\\hline\end{array}} + \underset{\text{Filter}}{\begin{array}{|c|c|c|}\hline 2&0&3\\\hline 0&1&1\\\hline 0&0&2\\\hline\end{array}} \longrightarrow \begin{array}{|c|c|c|c|c|c|}\hline 0&0&1&1_2&1_0&1_3\\\hline 1&0&1&0_0&1_1&1_1\\\hline 1&1&1&0_0&1_0&0_2\\\hline 1&1&1&1&1&0\\\hline 0&1&1&0&0&0\\\hline 1&1&1&0&1&1\\\hline\end{array} \longrightarrow \underset{\text{Filtered Image}}{\begin{array}{|c|c|c|c|}\hline 6&4&8&7\\\hline ..&..&..&..\\\hline ..&..&..&..\\\hline ..&..&..&..\\\hline\end{array}}$$

Figure 7: Filtering an image using correlation.

Fig. 7 illustrates image correlation on a simplified image(black and white pixels) with one depth channel and a filter. In the case of RGB pictures, the filter and image has three depth channels, making the image and filter of dimension 6x6x3 and 3x3x3. It should be noted that by filtering in this way, the dimension reduction leads to loss of valuable information in the picture. Therefore, in a lot of CNNs, padding is applied to the image beforehand. Padding increases the dimensions of the original image equal the dimensions removed by the filter. This results in a filtered image of the same size as the original image, and all the information is preserved.
The convolutional layers utilizes these techniques in order to extract features from the images. In practice multiple learnable filters are applied to each image which in turn creates multiple feature maps. After several convolutional and subsampling layers, the feature maps become more numerous and smaller as can be seen in Fig. 6. At first, the filters capture low level features such as edges and orientation of those edges. The deeper the convolutional layers are, the more high level features are captured such as a human face, bird etc. [29]

After each filtering operation, an activation function and batch normalization are applied. Batch normalization is used in order to help against internal covariance shift, a problem that makes it harder to train the network. Internal covariance shift is a problem due to the distribution of the current layers' input data changes when the parameters of the previous layer are updated. The algorithm works by calculating the mean and variance of the batch, then normalize it and at last the data is scaled and shifted. [30]

Subsampling
In the subsampling step, the feature maps coming from the convolutional layer are downsampled, which creates images with lower spatial dimensions. This greatly reduces the computational cost, while retaining the most important features of the incoming feature maps. By making the features more abstract, this layer also helps against overfitting, a problem when the network is too specialized on the training images and in turn performs poorly on general pictures. [31]

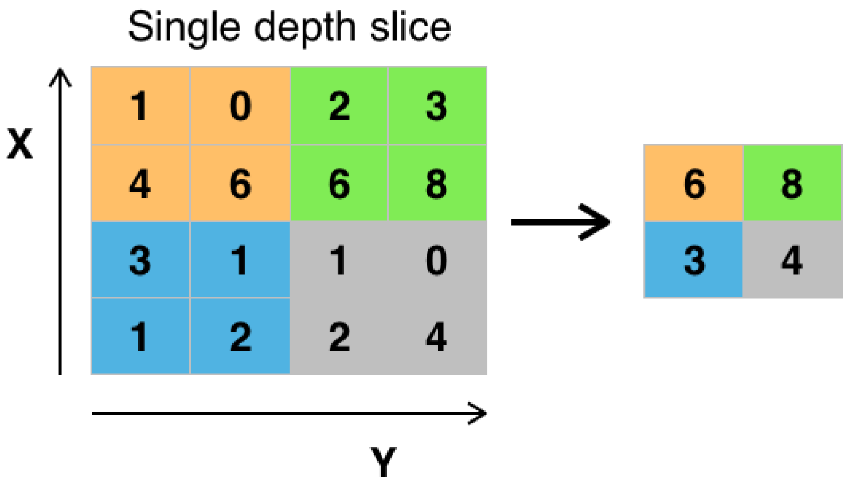


Figure 8: Max pooling.
Source: Image by Aphex34[5]

There are several pooling types, the most common is max pooling which is shown in Fig. 8. In max pooling, a fixed size region is sliding over the feature map, choosing the the element with the highest value. In this case the max pooling layer has a stride of two, which means the pooling region steps two elements for each iteration.[29]
Some CNNs abandons the use of pooling altogether, and instead opt for convolutional filters with strides higher than one. This is due to networks that uses convolution for downsampling, can at times match and even outperform networks using pooling[32].

Fully connected layer and loss layer
After several layers of convolution and subsampling, the extracted features of the original image is in a vector form, and is then passed into the fully connected layers(FCL). The fully connected layers consists of multiple layers of neurons, where each neuron in the layer is connected to the activations from the previous neurons, in additions to the neurons in the next layers. [28]
In the fully connected layer, the previous extracted features are used to learn during training, as well as classification during inference.
After the FCL, there is normally a loss layer that determines how the system penalizes error. This layer depends on what the system is predicting. In the case of images with multiple classes, sigmoid cross-entropy is used[33].

2.2.3 State-of-the-Art

Use of deep learning in computer vision has increased greatly in the last decade. Although deep learning through functioning convolutional neural networks(CNN) has existed since the 90s, it has been restricted by the lack of computing power. With the growth of computing power through faster CPUs and emergence of GPUs, the use of neural networks has become viable[34]. Especially in computer vision(CV), the use of CNNs on 2D images has become a tool that surpasses traditionally hand-crafted CV systems in performance. In addition, new network architectures designed for 3D data has emerged in the recent years. This section presents different state of the art deep learning networks, which utilizes 2D images or 3D data for object detection.

YOLO(You only look once)[6] is a fairly new CNN which at the time of paper release was state of the art for object detection on 2D images. The network uses a single convolutional neural network to predict bounding boxes and class probabilities. Through the improved versions YOLOv2[35] and YOLOv3[10] it has further improved its performance and the latest version is lagging a bit behind in term of accuracy compared to other state-of-the-art CNNs such as RetinaNet, but at significantly higher processing speed[10]. First the image is divided into a SxS grid(see Fig. 9), where each grid cell is checked for an object. If a grid cell contains an object, then an anchor box is associated with the object. Objects may span over multiple grid cells, which may result in multiple boxes associated with the same object. Threshhold for class prediction is then used to remove any detected objects with low confidence. To solve the issue with multiple bounding boxes for the same object, non-max suppression is used. The box with highest object detection confidence is chosen, and boxes with high Intersection over Union(IoU) with the chosen box is removed. Everything is done with one execution of the convolutional network, which makes it fast.

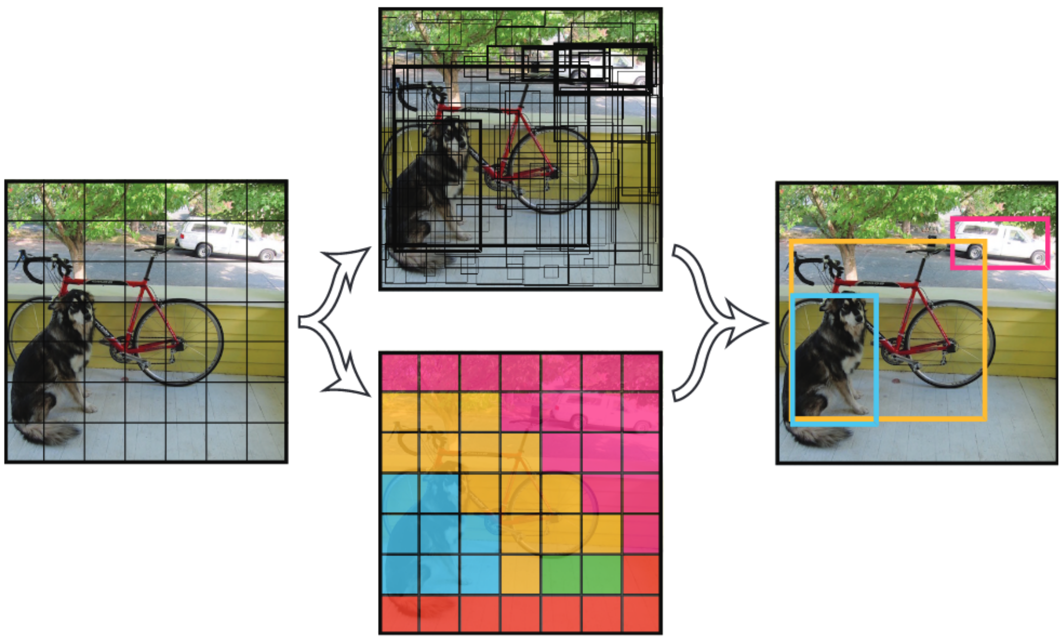


Figure 9: YOLO
Source: Image from [6]

YOLO is mostly used for object detection on 2D images alone, but paired with a depth sensor like Kinect, further information can be extracted from the detected object. An example is the use of YOLO and Kinect to detect robots in RoboCub MSL[36]. YOLO is used to detect the robots from Kinects RGB images, and the robots bounding box parameters are used to extract depth data in the region of the bounding box.

Another approach to object detection with 3D data is to utilize point cloud in machine learning. Machine learning with point clouds is a technique which has emerged recently, mainly due to research into autonomic vehicles with LiDARs.
This approach presents some new challenges: Point clouds do not have the advantage of the spatial relationship which the pixels have on 2D images. In contrast a point in a point cloud can exist at any location, and its positional data is encoded explicitly along other information. [37]

Some methods convert the point clouds to a voxel grid to solve these issues. One network that does this is VoxelNet[7]. This network utilizes LiDAR with around 100 000 points. As seen in Fig. 10, the point cloud is sent to a Feature Learning Network(FLN). Due to memory/processing issues of a point cloud of this size, the network samples points at random. This does not only reduce the required computational power, but also removes any potential bias because of the variable density of points in the point cloud. The sampled point cloud is then sent to a voxel feature encoder(VFE) which extracts and combines point-wise features and local aggregated features. Voxel-wise feature tensor is extracted through a fully connected layer and max pooling. After the FLN, convolutional middle layers extracts high level features by using 3D convolution. A region proposal network then outputs a regression map for 3D bounding boxes and a probability map containing the probability scores for the different classes.

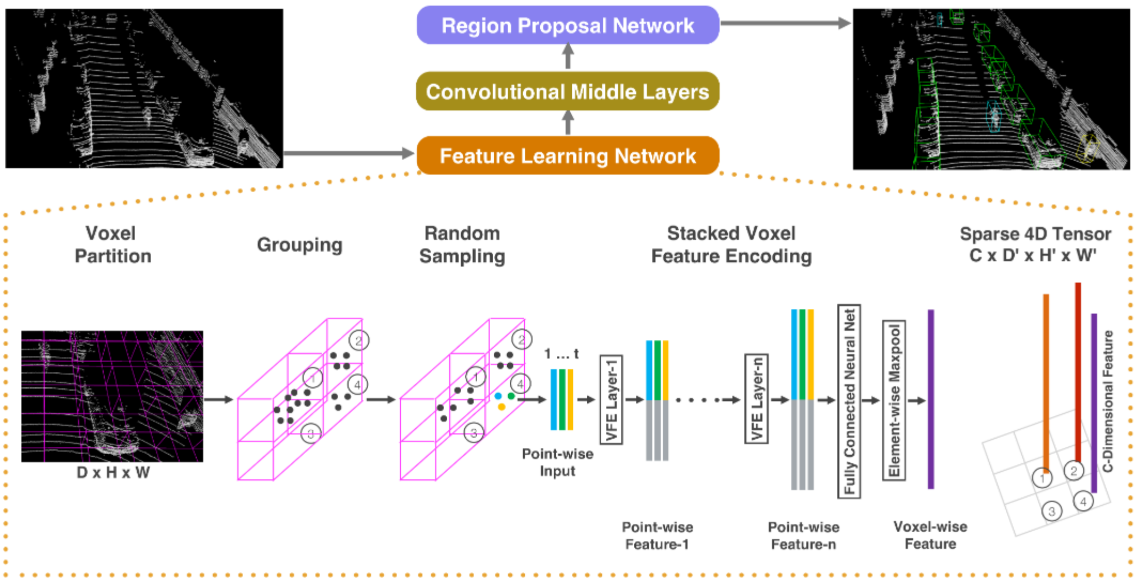


Figure 10: VoxelNet architecture
Source: Image from [7]

One end-to-end network utilizing point cloud is PointNet[38] and its successor

PointNet++[39]. Because of the unordered structure of point clouds, the network has to learn unique features while being invariant to permutations of the point cloud. To solve this, PointNet is using a symmetric function.

$$f(x_1, \ldots, x_n) = \gamma \odot g(h(x_1), \ldots, h(x_n)) \tag{3}$$

where f is a symmetric function which transforms the points $[x_1, x_n]$ to a k-dimensional vector(k being the number of object classes), containing the scores for classification. h is a multilayer perceptron(MLP), which consists of multiple FCLs that maps the input points into a latent space. g a symmetric function, in this case a max pooling function which aggregates the learned features into a global feature vector. This vector is then passed to γ(another MLP), and at last the predicted class score is outputted. As the feature learning should be invariant to geometric transformation, PointNet uses something called T-net. T-net transforms the input point cloud into fixed, canonical form which makes the network robust to small variations of the input cloud.

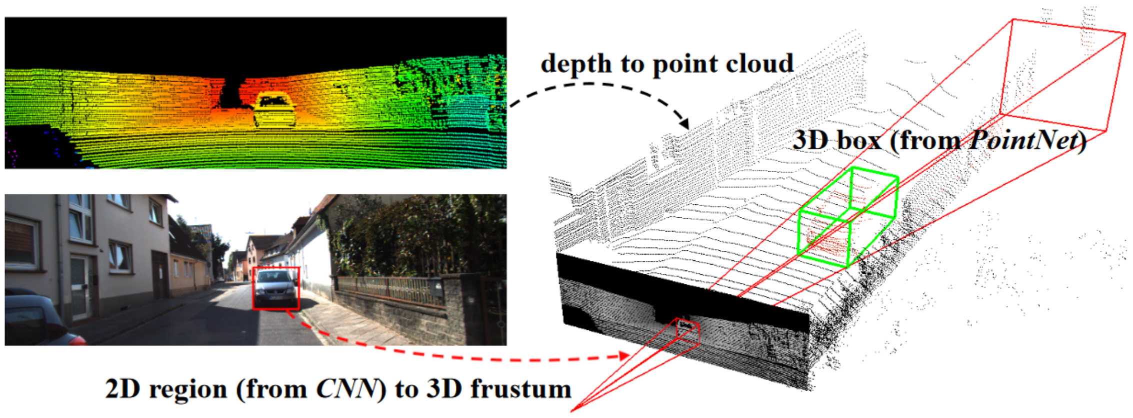


Figure 11: Frustum PointNets approach to object detection and segmentation.
Source: Image from [8]

Frustum PointNets[8] is a further extension of PointNet. As seen on Fig. 11 it uses a CNN on 2D images to detect and create a bounding box of an object. From the 2D bounding box it then creates a corresponding 3D frustum. The 3D frustum contains the point cloud data, which is fed to PointNet which performs the detection. PointNet then returns an oriented amodal 3D bounding box around the object. The network is also capable of doing semantic segmentation, where each of the points belonging to the object are grouped together. It does both operations in one go in contrast to a more conventional sliding window approach, which makes Frustum PointNets fast.

2.2.4 YOLOv3

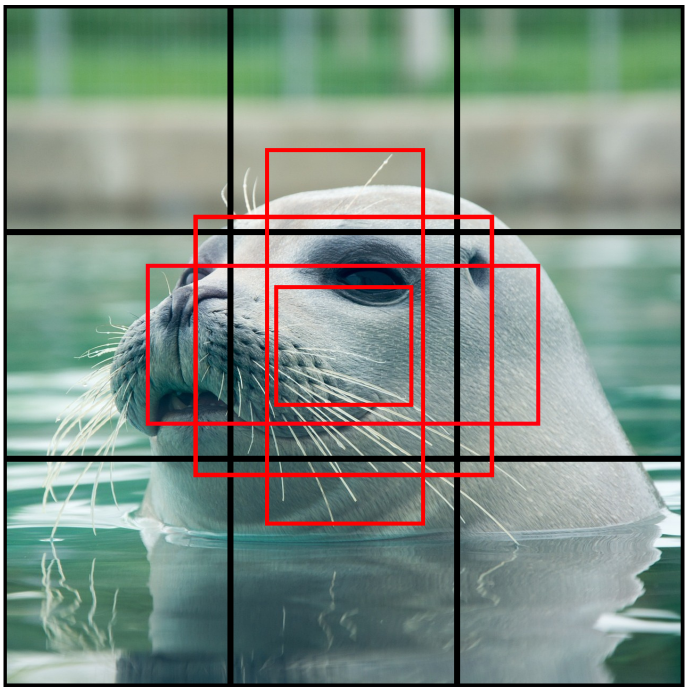
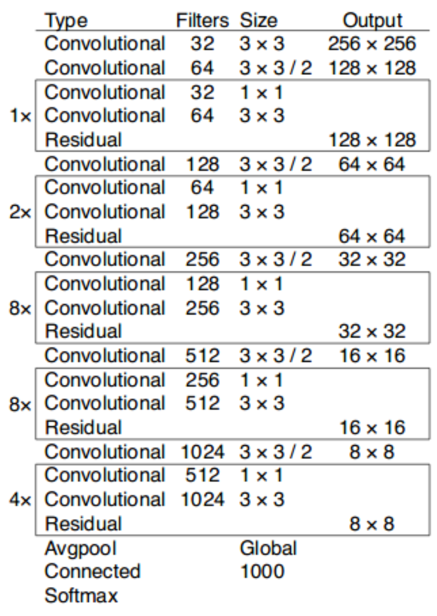


Figure 12: Grid cells(black) and anchor boxes(red).
Source: Adapted from [9]

Several of the 3D machine learning networks mentioned in the previous section requires powerful hardware to run. These networks are made for LiDARs or in the case of PointNet, smaller point clouds. Because the point cloud density of the Kinects are significantly higher than the mentioned point cloud sources, it was decided to go for YOLOv3, and combine this detection with more traditional point cloud algorithms. This network is fast, making it possible to run on the Jetson computers used in this thesis.

Object detection is done using grid cells, where each grid cell can detect a number of objects. The grid cell containing the object's center, is responsible for prediction. In addition, YOLOv3 uses something called anchor boxes. The anchor boxes is of predefined aspect ratio, and is optimized by using K-means clustering. The fixed aspect ratio of the anchor boxes makes predictions faster, relative to YOLOv1 where the aspect ratio among other parameters had to be optimized. To match the bounding box with the object, the offset from the anchor box to the objects center is precisely calculated and corrected, as well as scaled in order to match the object. The anchor box with the highest intersection over union(IOU) with the object is chosen. However, objects larger or smaller than the anchor boxes will remain undetected. Moreover, the network cannot handle cases when there are multiple objects assigned to the same anchor box. The network solves this by applying the same approach on three different scales. YOLOv3 uses grid resolution of 13x13, 26x26 and 52x52. A somewhat simplified version of the grid cells and anchor boxes can be seen on Fig. 12. The predictions on the three feature maps are then merged together in a Feature Pyramid Network(FPN) fashion. [10][40]



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 13: Darknet-53.
Source: Image from [10]

Feature extractor
YOLOv3 utilizes the feature extractor Darknet-53, which can be seen in Fig. 13. Instead of max pooling, the network uses a filter with a stride of two, in order to reduce the image size by half. YOLOv3 also utilizes something called residuals. Residual networks work by feed-forwarding the output from the previous layer and add this to the output of the current layer. The more layers added to a neural network, the more accurate it tends to be. However, previously very deep neural networks were limited due to the vanishing gradient problem. Residuals resolves this problem, allowing for deeper neural networks in addition to reducing the required computational power[41].

Feature Pyramid Network

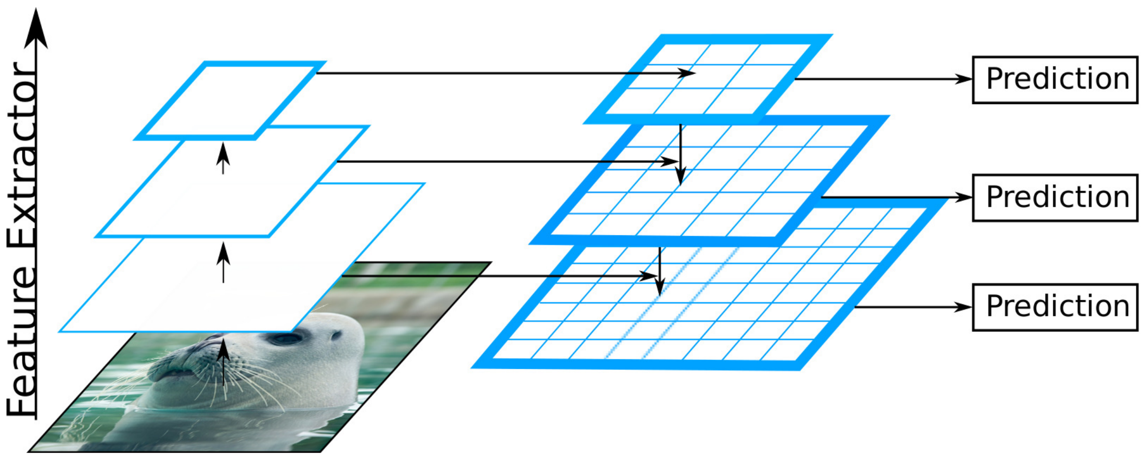


Figure 14: YOLOv3 algorithm.
Source: Adapted from [11][9]

YOLOv3 utilizes something called Feature Pyramid Network(FPN)[11] where the feature maps in the convolutional layers are extracted on different levels and merged to get good predictions of small and big objects. While larger objects are retained deep in the convolutional layers, the smaller objects vanish. Therefore predicting on multiple scales are beneficial for good detection of small and large objects. YOLOv3 takes feature maps at different levels of the feature extractor, passes it through additional convolutional layers and outputs the prediction based on grid network as illustrated on Fig. 14. On each feature level, the predictions at each grid cell uses three anchor boxes, making nine the total number of different anchor boxes used in Yolov3. The feature maps from deep in the feature extractor are also upsampled and concatenated to the feature maps extracted from more shallow layers. This is to be able to detect smaller objects, as these features vanish during downsampling. The predictions is then outputted from the three different scales as a 3D-tensor containing the objectness, bounding boxes and classes for each grid cell. [40]

Prediction 3D tensor:

$$N \times N \times (A \cdot (b + P + C)) \tag{4}$$

Where,

N is the grid resolution.
A is the number of anchor boxes.
$b = 4$, is the bounding box parameters: x and y position, width and height.
$P = 1$, is the objectness score.
C is the number of classes.

Tiny-Yolov3, the version used in this thesis, is a faster variant of the algorithm explained. It has its feature extractor trimmed down, having less layers. This version only predicts on two different scales, abandoning the layer responsible for detecting the smallest objects in favour of being faster.

2.3 Point Cloud

This section starts by presenting several model estimators that was considered in this thesis. In addition, the theory behind various different techniques ranging from point cloud construction to filtering, object detection and alignment is explained.

2.3.1 State-of-the-Art

RANSAC(Random Sample Consensus) is an algorithm to estimate parameters of a mathematical model[42]. The method is iterative, where model hypotheses are generated, and each hypothesis is then going through a verification process to check if the model is good. First m random points are selected from the input data, where m is the minimum amount of data points required for the model estimation, and the model parameters are computed based on the these points. Then the whole point space is considered, where the each point is checked against the model. If the point is within a set distance threshold it is considered an inlier, if the point is outside this threshold, it is an outlier. The threshold value is dependent on the application and dataset, however the number of required iterations, k, can be determined.

$$k = \frac{log(1-p)}{log(1-w^m)} \tag{5}$$

p is the probability of choosing an inlier when choosing a point. w is the inlier to total point data ratio. The number of iteration is required to get below the threshold $(1-p)$, that is, the probability of getting a sample with at least one outlier.

Further development has been done in order to improve computational speed, which is mainly done in two ways, either improve the generation of hypotheses or improve the model verification. Randomized RANSAC[43] is a method which does the latter. R-RANSAC does a preliminary test of the hypothesis, in order to disregard bad hypotheses early. By randomly selecting a small subset of the point cloud and test these against the hypothesis, bad hypotheses can be disregarded with high confidence. This improves the computational speed as the algorithm avoids the full model verification of hypotheses that are unlikely to give a good solution.
Progressive Sample Consensus (PROSAC)[44] is an algorithm which aims to improve the generation of hypothesis. Instead of picking points at random, the hypothesis generation is instead based on points with higher similarities. First the points are grouped based on similarity, then the largest of these groups are used to generate hypothesis.
The previously mentioned RANSAC-based algorithms do not have an upper time limit. Adaptive Real-Time Random Sample Consensus(ARRSAC)[45] aim to solve this problem. Due to an upper time limit when running in real time, there has to be a limit for the amount of hypotheses generated. The algorithm is generating different hypotheses and evaluates them on a subset of data points. The bad hypotheses are thrown away after evaluation, and good hypothesis provides an inlier ratio. When calculating the inlier ratio on a subset of the original data-set, the inlier ratio may not be correct. Therefore the algorithm allows for hypothesis generation in the evaluation stage. The good hypotheses are then further evaluated, returning the model.

A promising approach to pose estimation with point clouds is the algorithm proposed in [46]. The algorithm uses something called point pair features. A point pair feature(PPF) describes the relative position and orientation for two points. A set of reference points in the scene are selected, then the rest of the points in the scene are matched up to these reference points to create point pair features. Point pair features are not discriminative enough to detect planar and self-symmetric object, therefore [46] incorporates the color vectors of the two points into the PPF, resulting in color point pair feature(CPPF). This algorithm calculates all the possible CPPFs of the model. Then the set is stored in a hash table, and corresponding keys for the hash table are created. The data is then passed to a voting function where CPPFs of the model are compared to the scene point cloud, and if there is a match it computes a set of votes. These votes are then used to calculate the pose of the object.

2.3.2 Time-of-flight Depth Sensors and Kinect v2

RGB-D cameras provides depth measurement in real-time at a low budget, although only in close range. The specified operating range of Kinect v2 is 0.5m-4.5m[47]. Kinect v2 is a RGB-D camera which uses a technique called time-of-flight(ToF) to create a range image. A ToF sensor measures the time taken from emitting of light until it returns to the camera. In the case of the Kinect v2 it uses an infrared emitter and a sensor to detect the IR-light. The Kinect v2 uses a ToF technique called amplitude modulated continuous wave ToF(AMCW-ToF). The scene is continuously illuminated by a periodic intensity modulated infrared light. The sensor then measures the incoming light and senses a phase-shift depending on the distance the light has travelled[48].

2.3.3 Point Cloud Construction

Because the intrinsic matrix is needed when creating the point cloud, a short explanation describing the camera matrix, will he held. However, since camera calibration is outside the scope of this thesis, the calibration process is left out.

Pinhole Model

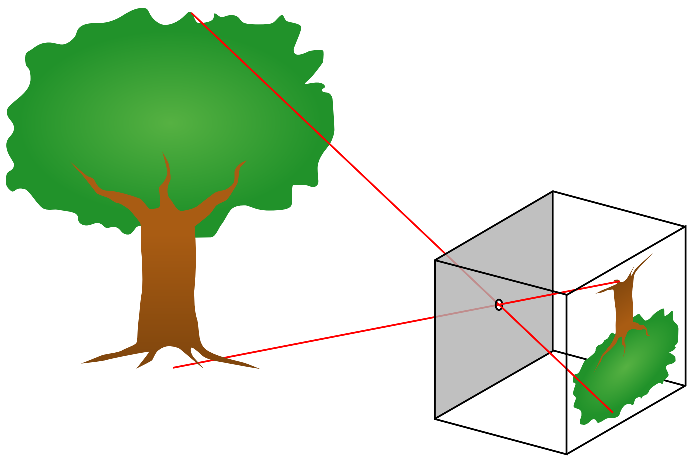


Figure 15: Pinhole camera model.
Source: Image from [12]

The most common camera model is the pinhole camera. As illustrated on Fig. 15, the model assumes the light from a 3D scene is projected through a single point, or pinhole, onto the 2D plane, otherwise called the image plane. The mapping from 3D world coordinates to 2D pixel coordinates can be represented as a 3×4 matrix.

$$\begin{aligned} P &= K \begin{bmatrix} R & T \end{bmatrix} \\ &= \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \end{aligned}$$

Where,
c_x [pixels], principal offset in x direction
c_y [pixels], principal offset in y direction
f_x [pixels], pixel focal length in x direction
f_y [pixels], pixel focal length in y direction
s [-], skew coefficient

The rotational matrix, R and translation vector, T describes the point's rigid transformation from the 3D world frame to the 3D camera frame[49].
The intrinsic camera matrix projects the point from the camera frame into 2D pixel coordinates. The principal offset describes the image center. It is a point on the image plane that is formed by a perpendicular line on the image plane that intersects through the pinhole. The focal length, F can be explained as the distance in world units between the pinhole and image plane. Often this is divided into two components $f_x = F/p_x$ and $f_y = F/p_y$, where each describes its relation between the focal length and physical pixel width/height respectively. Ideally $f_x = f_y$, meaning the pixels are perfectly squared, in practice this is not the case which results in distorted images. [50]

In fact, the camera matrix P, maps the world to an undistorted image[51]. Therefore, in addition to the camera matrix, finding the distortion coefficients that remove distortion effects such as pincushion/barrel are essential to get a good image.

Depth Map to Point Cloud

With a calibrated sensor, the point cloud can be made. This is a simple mapping procedure from the depth space to world space. This can be done with look-up tables as shown in [52], where a mapping value for each pixel in the depth image is made. The calculation shown below, can be done during run-time, but calculating a look-up table at the beginning saves this computation, making the program more effective.

$$u = (u_i - c_x) \cdot f_x \tag{6}$$

$$v = (v_i - c_y) \cdot f_y \tag{7}$$

Where,

u is the look-up value in the x direction of the image.
v is the look-up value in the y direction of the image.
u_i and v_i are in the range of [0, width] and [0, height] respectively.

Finally the 3D points are made by using the depth map and the generated look-up tables, to create all the points corresponding to the depth map elements.

$$x = u \cdot d \tag{8}$$

$$y = v \cdot d \tag{9}$$

$$z = d \tag{10}$$

Where,

d, is the depth value from the depth map.

2.3.4 Statistical Outlier Removal

Measurements from a ToF depth sensor are not perfect and there are several different sources of errors. For example, darker objects are more inaccurate at greater distances, resulting in depth values have larger variance and offset. The sensors are also vulnerable to multiple light returns from the scene to a single pixel on the sensor, giving an erroneous depth value. [53]
Because of these error sources among others, the point cloud become noisy. Such noise can be seen in Fig. 16. Therefore filtering is essential to get a clean point cloud to work with. A statistical outlier filter is a useful tool to remove these unwanted points. The following paragraphs will present a filter proposed in [13].

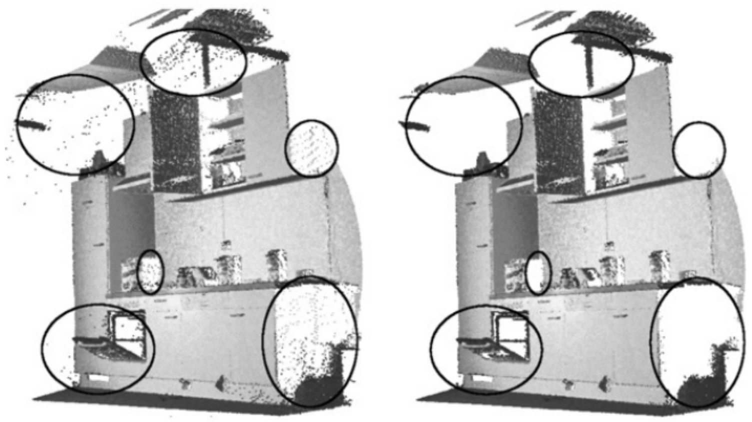


Figure 16: Noise filtering by statistical outlier rejection
Source: Image from [13]

The algorithm works by dividing the point clouds into several subsets. In each subset a point is chosen, and the mean distance μ from the point to its K neighbouring points is calculated. The algorithm assumes a Gaussian distribution and calculates the standard deviation, σ, as well.

It then iterates through each point again and checks if the neighbouring point is within a desired distance threshold, t, and rejects the points outside this threshold. The standard deviation multiplier α is set based upon the quality of the point cloud. On a rough and irregular point cloud, the standard deviation is high, therefore the point cloud may benefit by setting the standard deviation multiplier low. The standard deviation of an accurate and smooth point cloud is low, in that case the multiplier should not be set too low, as this will remove desired points.

$$t = \mu \pm \alpha\sigma \tag{11}$$

Where,

μ is the mean distance value between the chosen point and its K nearest neighbors.
α is the standard deviation multiplier.
σ is the calculated standard deviation of the distance from the chosen point.

2.3.5 Iterative Closest Point

Iterative closest point(ICP) is an algorithm used to align two point clouds. It is an important tool when reconstructing scenes, or pose estimating an object. Multiple algorithms have been developed to solve this problem: an iterative algorithm is described in [54], a non-iterative version based on quaternions[55]. However the algorithm used in this thesis is a version based on Singular value decomposition(SVD). In the case of many point correspondences, it is shown in [56] that SVD achieves faster computation speed in comparison to the other methods.

The main goal of the mentioned algorithms is to find a translation vector, t, and rotation matrix, R, to minimize the sum of the squared distance error between the two clouds. The error formula is as follows:

$$E(R,t) = \frac{1}{N_p}\sum_{i=1}^{N_p} ||x_i - Rp_i - t||^2 \tag{12}$$

where,
x_i and p_i is the corresponding model and measured data points.
N_p is the number of points contained in the model.
R is the rotation matrix.
t is the translation vector.

Finding the right point correspondences in the measured and model point sets are challenging, and will not be explained here. It is therefore assumed that the point correspondences are already found.

Before finding the transformation, the point sets are centered together. The center of mass for both the model and measured data are:

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \qquad \mu_p = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i \tag{13}$$

where,
μ_x, μ_p are the center of mass for the measured and model point sets.
N_x, N_p are the number of points contained in the measured and model point sets.

The center of mass are then subtracted from the measured point set X and model point set P:

$$(P - \mu_p) = P' \tag{14}$$

$$(X - \mu_x) = X' \tag{15}$$

The matching rotation around the point sets' center is found by using Singular value decomposition(SVD) on the matrix formed by the matrix product of P' and X'.

$$W = P'X'^T \tag{16}$$

SVD is used to factorize the matrix W into the matrices U, Σ, and V:

$$W = U\Sigma V^T \tag{17}$$

As shown in [56], the rotation matrix that gives the least error is as follows.

$$R = UV^T \tag{18}$$

According to [54] the optimal translation vector is

$$t = \mu_x - R\mu_p \tag{19}$$

After finding the most optimal rotation matrix and translation vector, the model cloud is then transformed. If desired, this process can be repeated a set number of times until the minimum least squares error is found.

2.3.6 Standard RANSAC

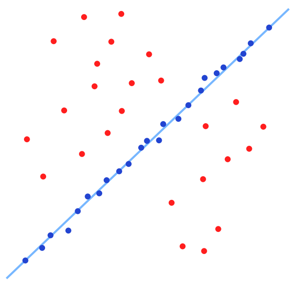


Figure 17: RANSAC model estimation in noisy point clouds.
Source: Image from [14]

RANSAC is a versatile tool used for alignment and recognition of primitive shapes. Point clouds from many modern day sensors are often riddled with noisy points called outliers. These points will negatively affect the algorithms using least squares regression as optimization technique. Instead of using a least squares loss function, it bases its model proposals on points that are within a certain threshold, i.e. inliers, and the model with the most inliers is kept.

The standard RANSAC algorithm proposed by Fischler and Bolles[42] is essentially a two step process:

1. From the point set, choose a number of points randomly and create a model/hypothesis based on these points. The number of points are based on the minimum amount of points required to determine the models parameters.

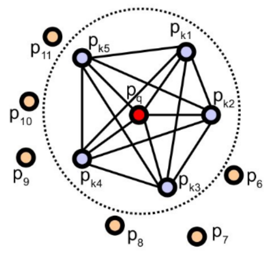
2. Check the hypothesis against the whole point set. Count the points within a set threshold.

This process is repeated a fixed amount of times, returning the model with the highest inlier ratio. As shown in Fig. 17 the algorithm can be used to estimate lines, in highly noisy data. More useful though, is the ability to segment other primitive shapes such as planes, cylinders, spheres and more.

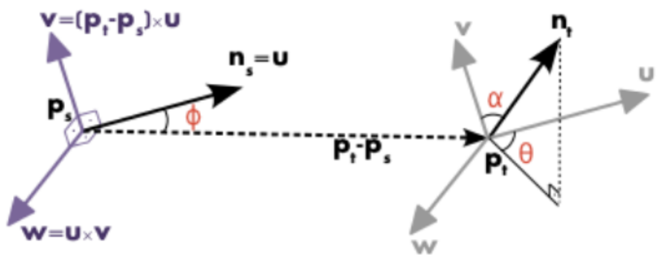
2.3.7 Prerejective RANSAC with Feature Descriptors

This section will describe a prerejective RANSAC algorithm that uses Fast Point Feature Histograms(FPFH) descriptors to match the corresponding features of the scene and model. This version is significant faster than the standard RANSAC algorithm, and therefore more suitable for real-time application.

Feature Descriptors



(a) PFH
Source: Image from [15]



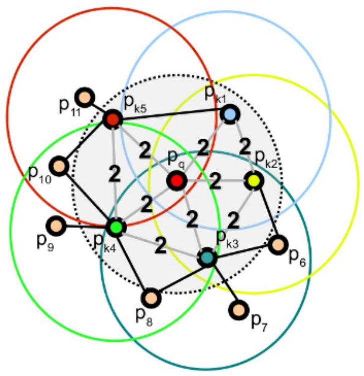
(b) Source: Image from [57]

Figure 18: Left: Spherical point neighborhood. Right: Relationship between two points.

Point Feature Histograms(PFH) represents the relationship between a number of points, giving a pose invariant description of the underlying surface. It is based on geometric properties between a point p and neighbour points k, that are contained within a sphere. This spherical neighbourhood can be seen in Fig. 18a. The properties are precomputed surface normals and position of the points. For each point pair combination in the sphere, the angle difference between the normals of the points are calculated, as illustrated on Fig. 18b. The angles are combined with the distance between the two points, d, forming a quadruplet which is then fed into a histogram.[58]

$$\alpha = v \cdot n_t$$
$$\phi = u \cdot \frac{p_t - p_s}{d}$$
$$\theta = tan^{-1}\frac{w \cdot n_t}{u \cdot n_t}$$

Where,
n_s and n_t are the normal vectors of the points p_s and p_t respectively.
d, is the distance between the two points.
u, w and v are the coordinate frame based on p_s.



Unfortunately, because the relationship between every point pair is calculated, PFH is not suitable for real-time applications. Fast Point Feature Histogram(FPFH) is a solution that runs in real-time while retaining most of the discriminatory capability of PFH. Instead of calculating the relationship between every point in the neighborhood, only the relationship between the point and its closest neighbors are calculated(see Fig. 19). In addi-

Figure 19: FPFH
Source: Image from [15]

tion, FPFH discards the distance parameter, as it was found that it did not enhance the algorithm's robustness. [15]

The algorithm is a two step process:

1. Calculate the geometric relationship between the main point p_q and its neighbors, p_k(though not between the neighbors), resulting in a Simplified Point Feature Histogram(SPFH).

2. For each of the main point's neighbors, p_k, calculate the SPFH for its own neighbors. Then the resulting histogram is a sum of the main point's SPFH and a weighted sum of the neighbors' SPFH.

$$FPFH(p) = SPF(p) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot SPF(p_k) \tag{20}$$

Where,
p, is the main point.
k, is the number of neighboring points.
ω_k, weight based on the distance between p and p_k.

Prerejective RANSAC
The RANSAC algorithm presented is based on [59]. This version improves a common RANSAC algorithm by introducing a prerejective step where the feature correspondences are compared, and rejected if they are too dissimilar. The algorithm can use other 3D feature descriptors, but in this case FPFH descriptors was chosen due to its real-time capability.
The algorithm works as follows:

1. Find $n > 3$ random points in the model and the corresponding points in the scene by using feature descriptor matching.

2. Calculate the dissimilarity vector, δ. If $||\delta|| < t_{poly}$ continue, if not, go to step 1.

3. Estimate a hypothesis transformation matrix, $\hat{T}$, based on the n sampled point correspondences.

4. Transform the model cloud with the estimated matrix.

5. From the transformed model, find the amount of inliers at a set Euclidean distance threshold. If the inliers are less than a set limit, go to step 1.

6. Re-estimate a hypothesis based on the inlier point correspondences.

7. Check the least squares distance error between the corresponding points of model and scene. If it is the lowest so far, keep the transformation matrix.

Step 2 uses the fact that during rigid transformation the lengths between every pair of points are preserved. The algorithm calculates the edge lengths between each point in the model, $d_{p,i} = ||p_{i+1modn} - p_i||$ and similarly for the scene, $p_i, q_i \in \{1, ...n\}$. Then the corresponding edge lengths are compared and put in a dissimilarity vector, δ. If $||\delta|| < t_{poly}$ then the correspondences are accepted.

$$\delta = \left[\frac{|d_{p,1} - d_{q,1}|}{max(d_{p,1}, d_{q,1})} \cdots \frac{|d_{p,n} - d_{q,n}|}{max(d_{p,n}, d_{q,n})}\right] \tag{21}$$

Where
    d_p and d_q are the polygon edge lengths.

Ideally, $||\delta|| = 0$ if there is a perfect match between the model and the corresponding area in the scene. This may be the case if the two point clouds are artificially generated by computers. In practice, the clouds contain inaccuracies because of the sensors, resulting in edge differences of the corresponding edges. Therefore a threshold, t_{poly}, has to be set, and the more inaccurate the point cloud is, the higher the threshold.

2.4 Rigid Transformation

The rigid transformation matrix describes the change in orientation and position of an object. In addition, it preserves the geometric properties, meaning the shape of the object is preserved during transformation. It consists of two parts, a rotation matrix and a translation vector. In the case of 3D space, the rotational matrix is of the dimension 3x3 and consists of the elements R_{11} to R_{33} which can be seen in eq. 22. The translation movement is described by the row vector and in the case of 3D, contains the elements T_x, T_y and T_z.

$$A = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{22}$$

The translation parameters can be easily read from the transformation matrix, however finding the Euler angles from the matrix, is slightly more difficult. It should be noted that working with euler angles in a system is not preferable due to no unique solutions, but because of the intuitive nature of these angles it can be in some cases preferred.

Rotation
The rotation matrix in 3D space is a combination of rotations around the x, y and z axes.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta_x) & -sin(\theta_x) \\ 0 & sin(\theta_x) & cos(\theta_x) \end{bmatrix} \qquad R_y = \begin{bmatrix} cos(\theta_y) & 0 & sin(\theta_y) \\ 0 & 1 & 0 \\ -sin(\theta_y) & 0 & cos(\theta_y) \end{bmatrix} \qquad R_z = \begin{bmatrix} cos(\theta_z) & -sin(\theta_z) & 0 \\ sin(\theta_z) & cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A standard way of expressing the total rotation matrix is in the following order.

$$R = R_z(\theta_z)R_y(\theta_y)R_x(\theta_x) \tag{23}$$

The order of these matrices does matter, and reordering will change the output orientation. In this case, if read from right to left, the rotation is in the order x-y-z around a fixed coordinate system. When multiplied together, the resulting rotation matrix becomes:

$$\begin{aligned} R &= \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \\ &= \begin{bmatrix} cos(\theta_y)cos(\theta_z) & sin(\theta_x)sin(\theta_y)cos(\theta_z) - cos(\theta_x)sin(\theta_z) & cos(\theta_x)sin(\theta_y)cos(\theta_z) + sin(\theta_x)sin(\theta_z) \\ cos(\theta_y)sin(\theta_z) & sin(\theta_x)sin(\theta_y)sin(\theta_z) + cos(\theta_x)cos(\theta_z) & cos(\theta_x)sin(\theta_y)sin(\theta_z) - sin(\theta_x)cos(\theta_z) \\ -sin(\theta_y) & sin(\theta_x)cos(\theta_y) & cos(\theta_x)cos(\theta_y) \end{bmatrix} \end{aligned}$$

From this matrix the Euler angles can be found. Looking at element R_{31} the angle θ_y can be derived as shown:

$$R_{31} = -sin(\theta_y) \tag{24}$$

Unfortunately, there are two solutions which satisfies eq. 24. This results in two valid sets of euler angles.

$$\begin{aligned} \theta_{y1} &= -sin(R_{31}) \\ \theta_{y2} &= \pi - \theta_{y1} = \pi + sin(R_{31}) \end{aligned}$$

The rotation around the x-axis:

$$\frac{R_{32}}{R_{33}} = \frac{sin(\theta_x)cos(\theta_y)}{cos(\theta_x)cos(\theta_y)} = tan(\theta_x) \Rightarrow \theta_x = tan^{-1}\frac{R_{32}}{R_{33}} = atan2(R_{32}, R_{33}) \tag{25}$$

In many programming languages, the function atan2(y,x) is used. It takes into account both signs of the arguments in order to determine which quadrant the results lies in. Lastly, the rotation around z-axis is as follows:

$$\frac{R_{21}}{R_{11}} = tan(\theta_z) \Rightarrow \theta_z = tan^{-1}\frac{R_{21}}{R_{11}} = atan2(R_{21}, R_{11}) \tag{26}$$

These equations work if $\theta_y \neq \pm\frac{\pi}{2}$. If $\theta_y = \pm\frac{\pi}{2}$, then $cos(\theta) = 0$. This results in the elements R_{11}, R_{21}, R_{32} and R_{33} will all be zero. Therefore other elements in the rotation matrix has to be chosen in order to find θ_x and θ_y.

In the case of $\theta_y = \frac{\pi}{2}$, θ_x and θ_z can be found by using the matrix elements R_{12}, R_{13}, R_{22} and R_{23}.

$$\begin{aligned} R_{12} &= s_x s_y c_z - c_x s_z \\ R_{13} &= c_x s_y c_z + s_x s_z \\ R_{22} &= s_x s_y s_z + c_x c_z \\ R_{23} &= c_x s_y s_z - s_x c_z \end{aligned} \quad \xrightarrow{\theta_y = \frac{\pi}{2}} \quad \begin{aligned} R_{12} &= s_x c_z - c_x s_z = sin(\theta_x - \theta_z) \\ R_{13} &= c_x c_z + s_x s_z = cos(\theta_x - \theta_z) \\ R_{22} &= s_x s_z + c_x c_z = cos(\theta_x - \theta_z) = R_{13} \\ R_{23} &= c_x s_z - s_x c_z = -sin(\theta_x - \theta_z) = -R_{12} \end{aligned}$$

Unfortunately, this gives us an equation with no unique solutions for θ_x and θ_z.

$$\frac{R_{12}}{R_{13}} = \frac{sin(\theta_x - \theta_z)}{cos(\theta_x - \theta_z)} = tan(\theta_x - \theta_z) \Longrightarrow \theta_x = \theta_z + atan2(R_{12}, R_{13}) \tag{27}$$

Therefore either of the two angles, can be arbitrarily chosen in order to find the other angle. In the case of $\theta_y = -\frac{\pi}{2}$, the two angles can be found in a similar fashion[60]:

$$\begin{aligned} R_{12} &= s_x s_y c_z - c_x s_z \\ R_{13} &= c_x s_y c_z + s_x s_z \\ R_{22} &= s_x s_y s_z + c_x c_z \\ R_{23} &= c_x s_y s_z - s_x c_z \end{aligned} \quad \xrightarrow{\theta_y = -\frac{\pi}{2}} \quad \begin{aligned} R_{12} &= -s_x c_z - c_x s_z = -sin(\theta_x + \theta_z) \\ R_{13} &= -c_x c_z + s_x s_z = -cos(\theta_x + \theta_z) \\ R_{22} &= -s_x s_z + c_x c_z = cos(\theta_x + \theta_z) = -R_{13} \\ R_{23} &= -c_x s_z - s_x c_z = -sin(\theta_x + \theta_z) = R_{12} \end{aligned}$$

$$\frac{R_{12}}{R_{13}} = tan(\theta_x + \theta_z) \Longrightarrow \theta_x = atan2(R_{12}, R_{13}) - \theta_z \tag{28}$$

3 Implementation

This section will present how the system was implemented. First, a system utilizing a single Kinect is shown, then a system containg a network of six sensor nodes. For the source code, visit my Github repositories [61] for the pose estimator node, [62] for the ROS YOLO detector and [63] for the annotation tools used for YOLO. The system is based on libfreenect2[64], to retrieve data from the Kinects. In addition, iai_kinect2 [52] was used in order to get the data into an ROS environment. The system utilizes the point cloud library(PCL) implementations of point cloud algorithms such as ICP, RANSAC and statistical outlier filter among others.

3.1 System Overview

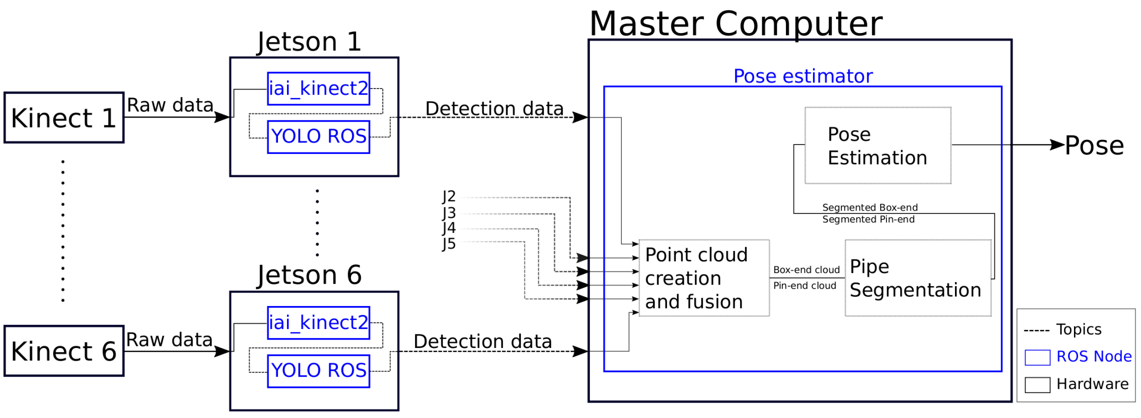


Figure 20: Camera network diagram

The Kinect network system works as follows:

1. Data from Kinect is captured using iai_kinect2, which publishes the data on multiple topics.
2. QHD image, SD depth map, and intrinsic parameters are retrieved from iai_kinect2, and sent to YOLO ROS. QHD image is used by the Tiny-YOLOv3 detector, which outputs the bounding box coordinates in its own topic, as well as passing on the depth map and intrinsic parameters.
3. The pose estimator node fetches the topics from Jetson, creating point clouds based on the depth map, intrinsic parameters and bounding boxes. These point clouds are then fused together based on the pose of the different Kinects, creating a complete point cloud for the box-end and pin-end in the world coordinate system.
4. The two point clouds are sent to a custom made class that segment and filter the clouds.
5. In the same class, the models are aligned with the two point clouds in order to get the pose.

The system with one camera works similar, except the camera is connected directly to computer, using iai_kinect2 and a YOLO node(although a different implementation) to get the detection data.

3.2 Lab Setup

The Industrial Robotics Lab(IRL) at the University of Agder, seen in Fig. 21, was used as a testing ground. It is equipped with six sensor nodes placed on the walls around 5 meters above the floor. The working area of the sensors is approximately 10x10x4m. The sensor node setup is imitating industrial sites one can find on offshore drilling ships, among others. Each sensor node contains a Jetson TX2 computer and a Kinect v2 RGB-D camera placed in a waterproof box. The information each sensor node provides is sent by Gigabit Ethernet to a central computer. [65]

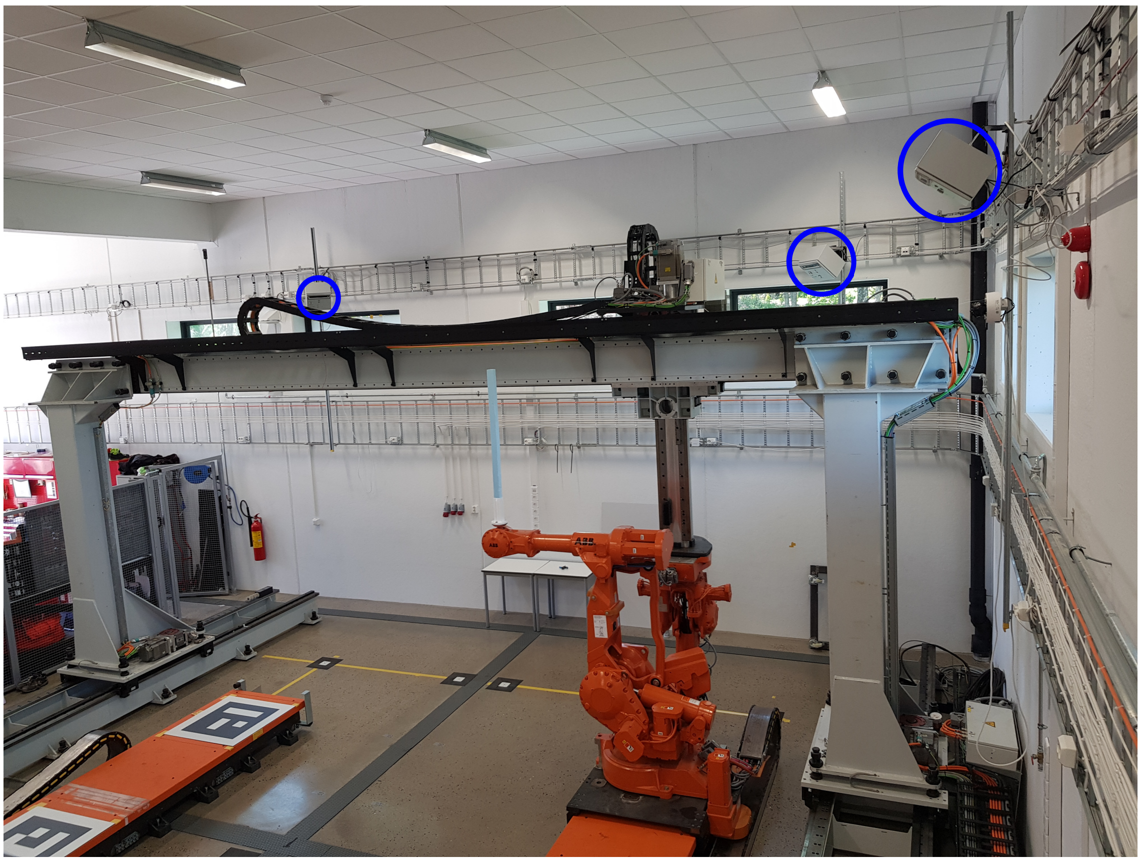


Figure 21: The Industrial Robotics Lab. Three of six sensor nodes shown in blue.

The PC running the pose estimator ROS node in this setup is equipped with an Intel i5-8300h CPU, a Nvidia GTX 1050 Ti(mobile) GPU and 8 GB of RAM. The sensor network was calibrated prior to this thesis, therefore no calibration is presented in this paper. The calibration procedure achieved an accuracy with an average euclidean distance error of 3cm at up to 9.45m. For more information on this procedure see [65].

In addition to the IRL, a set of truncated pipe-ends(see Fig. 22) were provided to do the testing. Ideally it would be interesting to create the system with pipes of full length, but each pipe segment is $\approx 6m$ at minimum, making it impractical. In practice, the texture and colour of the pipes is a combination of steel, mud and rust. However these are painted black, but was later changed to a lighter colour as is explained in Sec. 3.4.1.

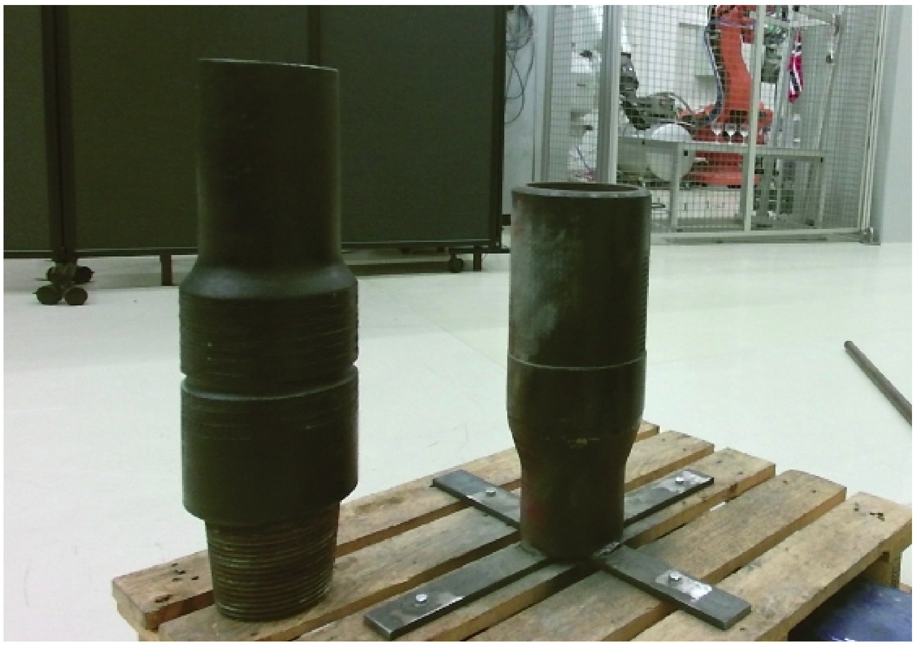
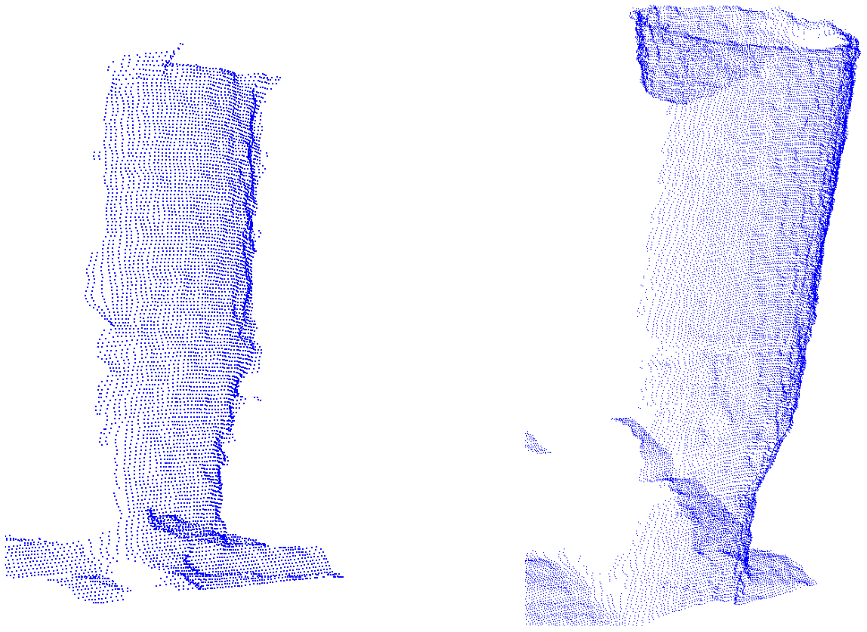


Figure 22: The Pin-end and Box-end.

3.3 Single Camera



(a) Box-end, roughly 2 m away from camera. (b) Box-end, under 1m away from the camera.

Figure 23: Box-end pipe cloud.

Originally, the main idea in this thesis was to mount a single Kinect camera on the iron roughneck. In this configuration the camera is close to the pipes which results in high resolution, capturing the features of the pipes. This can be seen on Fig. 23, where the point cloud closest to the camera is the most detailed. In addition, the point cloud is less affected by noise. The system uses Tiny-YOLOv3 to detect the pipes and make cutouts in the point clouds. This will be explained further in Sec. 3.4. From this cutout point cloud, a full segmentation was obtained by using a standard version of RANSAC to remove planes and extract cylinders in order to remove the leftover unwanted points.

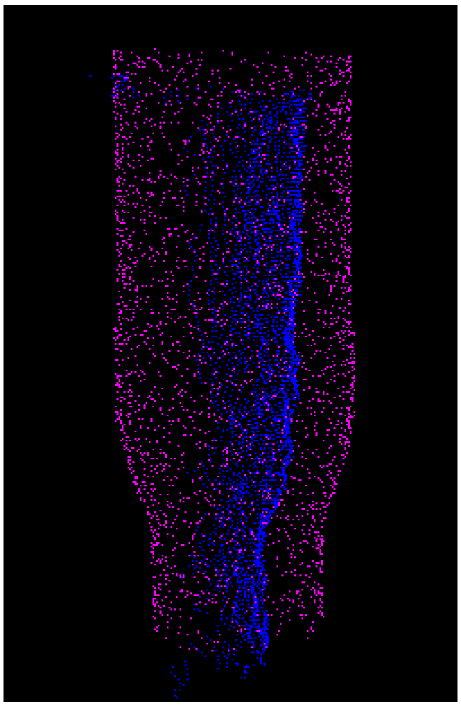
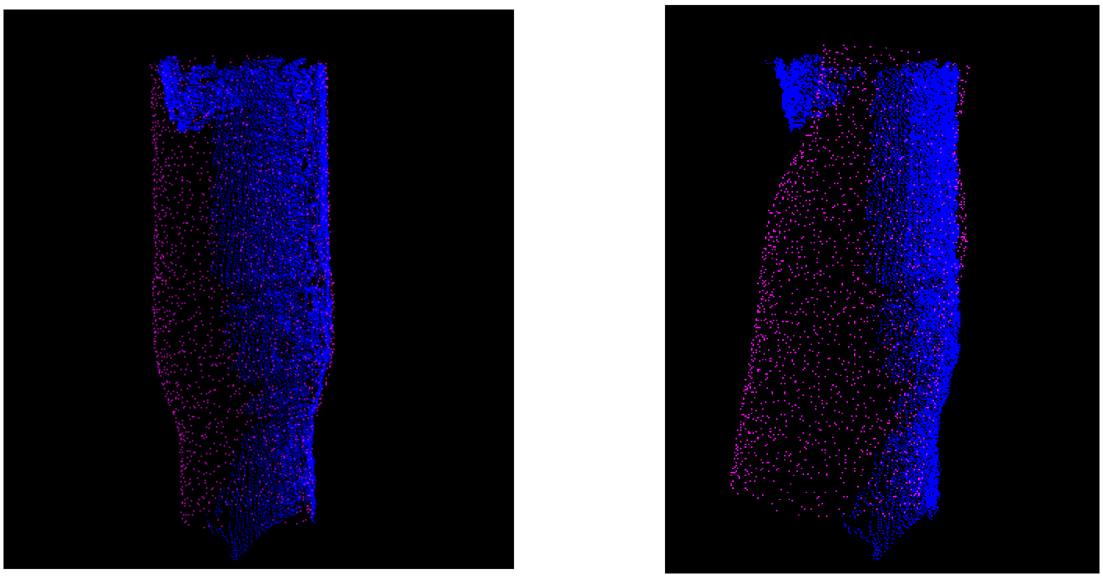


Figure 24: ICP alignment with model(purple) and scene(blue), using a single camera.

When using a single camera the point cloud of the pipe is incomplete. This affects what kind of alignment technique can be used. There was an attempt to use two different techniques, a prerejective version of RANSAC and ICP. When using ICP, the model is incorrectly aligned to the scene cloud, as can be seen on Fig. 24. The model's orientation follows the pipe cloud pretty well, however the vertical center of the pipes are not aligned properly, resulting in an offset. Therefore ICP was found to be unsuitable for pose alignment in a single camera setup.

Another approach for pose alignment was tried as well, where a prerejective RANSAC algorithm[59] was used. As seen in Fig. 25a, the algorithm performed better. The scene's and the model's vertical center are aligned properly. However due to the random nature of RANSAC, this method resulted in unreliable pose estimating. In some cases the algorithm would estimate the model to be upside down compared to the scene cloud, other times the orientation was incorrect as can be seen on Fig. 25b. In addition, the run-time of the RANSAC algorithm implemented in PCL was slow and fluctuating. Because of these problems, the RANSAC algorithm for pose estimating was abandoned as well. It should been mentioned that the parameters used when testing may have been suboptimal. Some of the unreliability issues can be prevented by reducing the prerejection of the algorithm. However, this results in a significant increase of computing time with minimal improvement to alignment.



(a) Proper alignment

(b) Misalignment

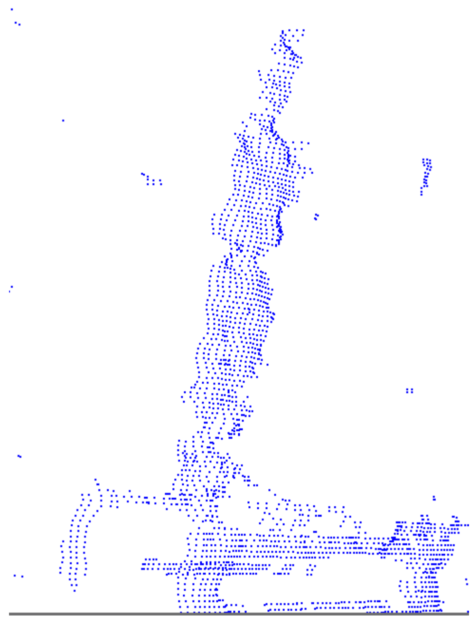
Figure 25: Alignment using RANSAC.

3.4 Kinect Network

Although the thesis requires a sensor node network, it was decided to attempt the single camera system due to a simple and less costly system. However, because of the unreliability of the pose estimation explained in 3.3 and the project requirements, it was decided to adapt the system. While a single camera mounted close to the object may capture the features of the object better, a network of cameras will capture the overall shape of the object better, hopefully improve the pose estimation.

3.4.1 IR Camera Limitation

Unfortunately, having cameras at a distance results in a lower resolution of points describing the pipes, in addition to the lower depth accuracy. Even worse, the dark colouring of the pipes makes the details deteriorate more with larger distances. The IR light emitted from the IR camera is absorbed by the pipes and the quality of the point cloud is therefore reduced. Shown in Fig. 26 is the pipe connection, at roughly the distance of 6 meters. Looking at the point cloud in Fig. 26a, it is hard to recognize the pipe connection. In the mentioned figure the pipe is stained with dried dirt that gives it a lighter colour, which in turn gives better point clouds. However the problem is worse if the colour is dark rusty, or black. This case is shown in Fig. 27a, where very little of the details gets captured on the depth map, and in turn in the point cloud.


(a)

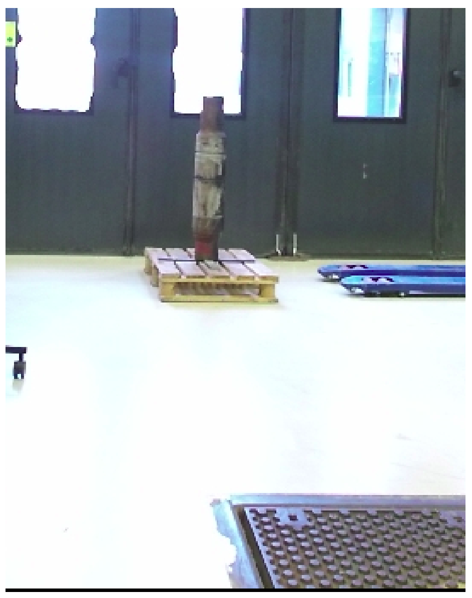

(b)

Figure 26: Pipe-connection, around 6 m away from camera.

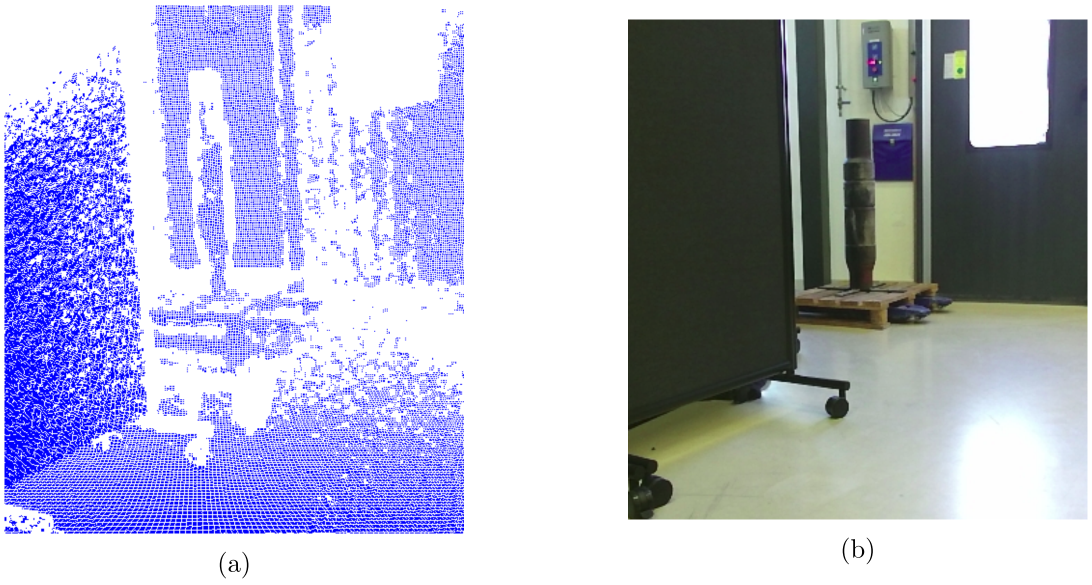

(a) (b)

Figure 27: Pipe-connection, around 6 m away from camera.

This is a fundamental problem when trying to pose estimate based on point clouds on larger distances. As this thesis is focusing on developing a pose estimating system based on point clouds, the camera limitation is circumvented by painting the pipes in yellow.

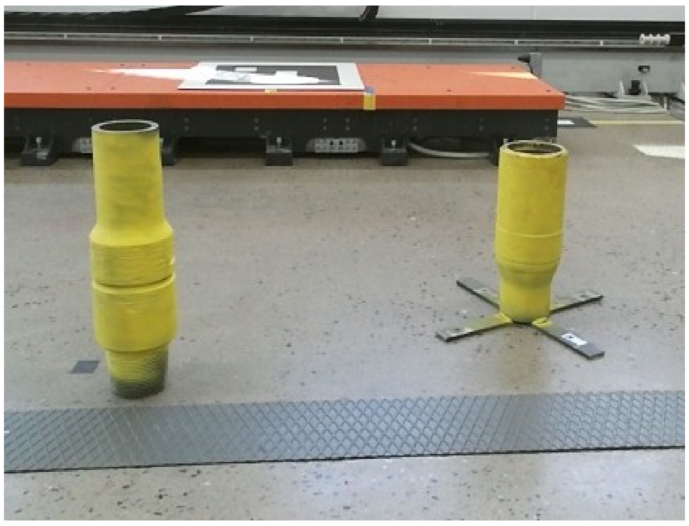
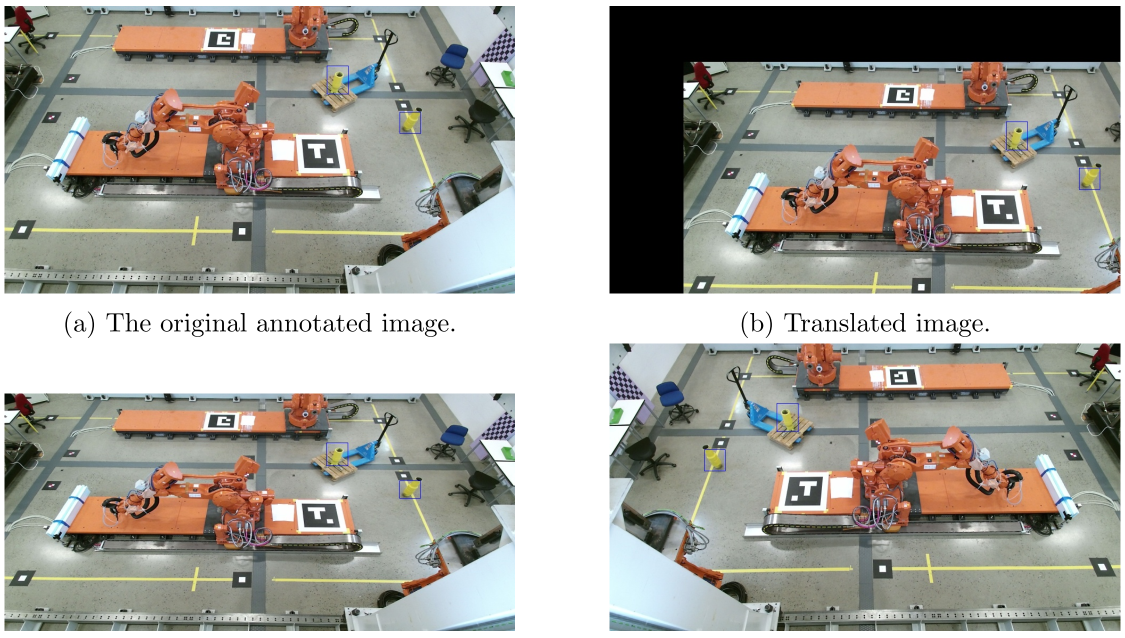


Figure 28: Repainted pipe-ends.

3.4.2 YOLO Object Detector

The system utilizes the YOLO implementation in ROS by Marko Bjelonic[66]. The only modification of the code is change of input topics and changing the network weights to a version that detects box-end and pin-ends. The network utilizes QHD(960x540 resolution) images, which are downsampled from the Kinect's HD camera. Libfreenect and iai_kinect2 provides SD images as well, which gives faster run-time at the cost of accuracy. However, the provided SD image is created by merging the depth-map pixels with the corresponding RGB pixels that results in a noisy picture. Therefore it was chosen to use pure QHD RGB images for the detector, where the details are better captured. The training images for the object detector were captured by using the Kinects connected to each Jetson computer. The captured pictures were then annotated using a self-developed tool.
The annotation tool outputs the annotated data in VOC dataset style, although it can convert the annotation files into the format YOLO uses. There are also implemented some simple data augmentation techniques: translation, scaling, and horizontal mirroring/flipping(flip around the vertical axis). The scaling occurs in both x and y direction, although it is not as visible on Fig. 29c. The blue rectangles shown in the images are for illustrative purposes, and do not appear on the images used for training.


(a) The original annotated image.

(b) Translated image.

(c) Scaled image.

(d) Mirrored/flipped image.

Figure 29: Annotated images.

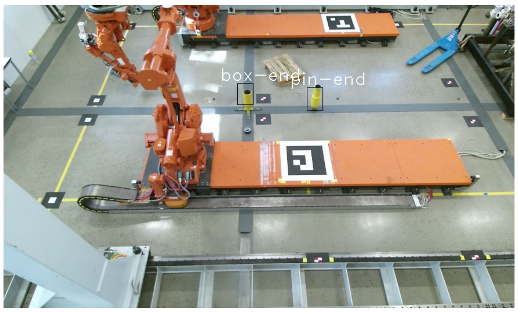
Due to the limited computing power of the Jetson TX2, it was chosen to use the YOLOv3-tiny version. This variant of YOLO requires less powerful hardware than the regular YOLOv3 version, although with less accuracy. To train the network, the implementation by AlexeyAB[67] was adopted. To reduce training time, transfer learning was utilized by utilizing pre-trained weights as a starting point for training. These are weights that are trained on many classes from different datasets, where some om these learned low-/mid-level features might be useful for recognizing the pipes. This significantly reduces the training time when training for the classes utilized in this thesis. In addition the implementation uses standard anchor sizes for Tiny-Yolov3, i.e. two sets of 3 anchor boxes for detection on two feature maps.

In the case of multiple detections of the same pipe-end class(see 40d and 40e), a function was implemented that discounts the detection with the lowest confidence. This limits the detector system to only a single set of pipe-ends.

The object detector is then implemented on each Jetson computer. To start the object detectors from a local PC, a ROS-launch file was created which communicates with the Jetsons over the network using the SSH protocol. The run-time speed of the detectors are roughly 13-14 FPS, which is fast relative to the rest of the system.
This detector is trained on fairly stable lighting condition. The ceiling lamps provide constant illumination, and only light coming through the outer windows affects the lighting. However on a drilling platform, the light will greatly vary. These are conditions the detector has to be trained for, as well as using unpainted pipes. In addition, the combination of bad lighting and the darker color of the actual drilling pipes will likely affect the performance of the detector.

3.4.3 Point Cloud Fusion

After the pipe detection, the information is sent to a master computer for point cloud creation and pose estimate. Each Jetson publishes the topics containing the bounding box coordinates from the object detector, as well as the corresponding depth-map and intrinsic camera parameters. In order to receive this information, the subscriber utilized an approximate synchronizing policy because of the asynchronous publishing between the different Jetson computers. This asynchronous data transportation results in erroneous point clouds when the pipes are moving. However, the Jetsons publish at a rate of $\approx 14FPS$, which means the pipes has to move fast for this to be a problem. In addition, due to pose estimation on moving pipes is not a requirement, it was chosen to continue with this. To speed up the system a bit, an asynchronous spinner was used to handle the callback function on a separate processor thread.
Based on the depth map, intrinsic parameters and bounding boxes, two cut-out point clouds are created containing the pin-end and box-end as can be seen on Fig. 30. It should be noted that the point cloud in Fig. 30b is transformed to the world frame, which is not the case when the cut-out is performed.



(a) Detected objects.

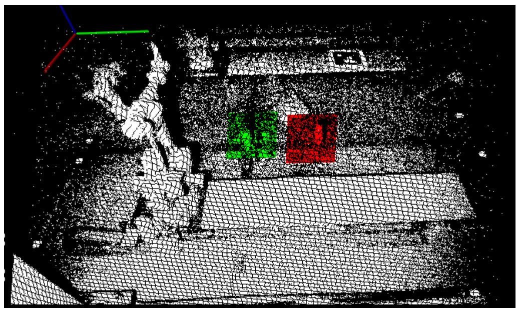

(b) Point cloud of box(green) and pin(red).

Figure 30: The corresponding QHD image and point cloud from Jetson3

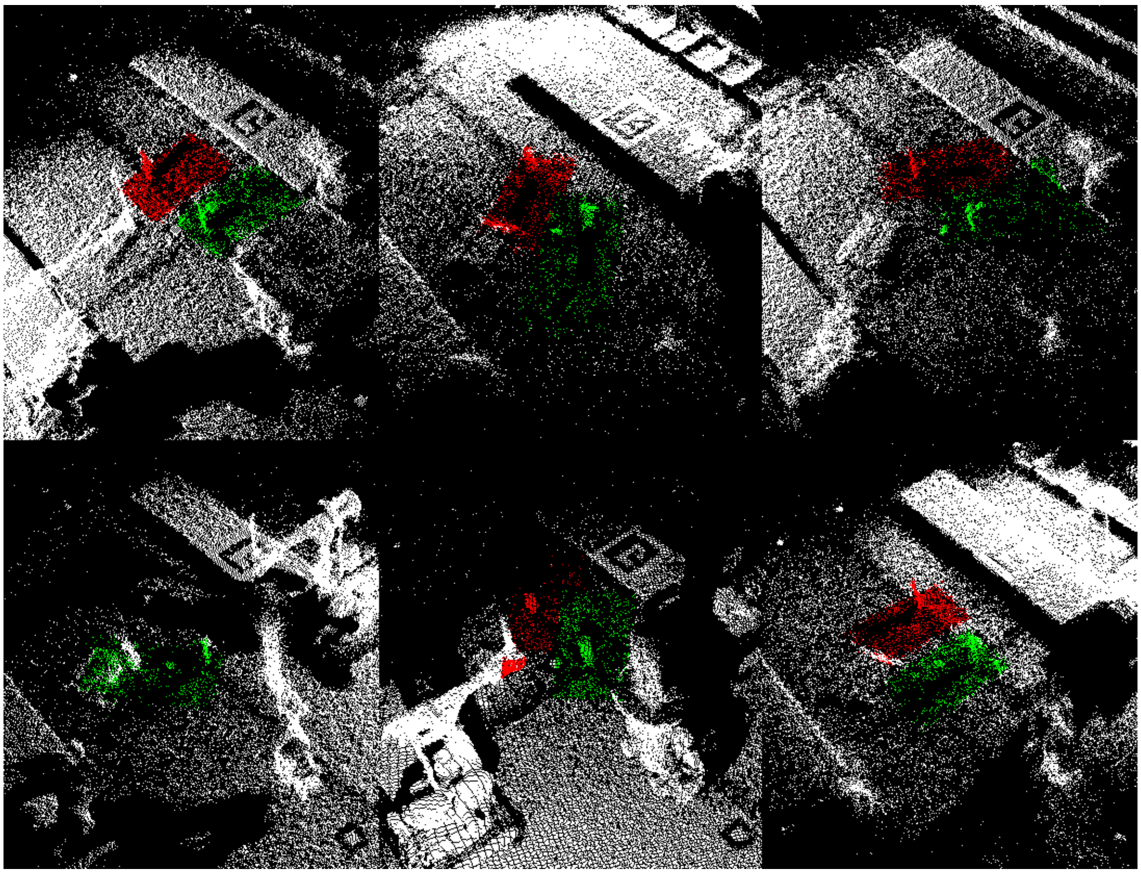
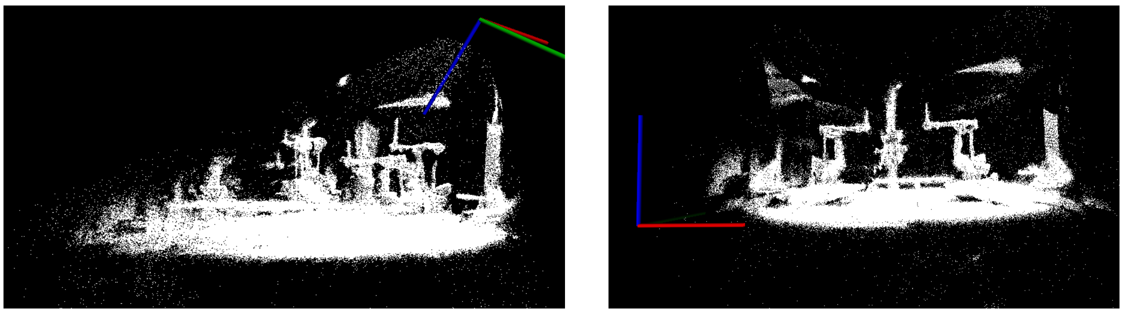


Figure 31: Point cloud cut-out of pin-end(red) and box-end(green) from the different cameras. Where the white points are the full view of the camera.

Because the bounding box coordinates are in the QHD frame, the coordinates have to be converted to the depth-map frame. This is a registration problem, finding the corresponding RGB pixel coordinate and depth-map pixel coordinate. Since this system is a proof of concept, finding exact RGB and depth-map pixel correspondences were not prioritized. In this case the corresponding bounding box between the RGB image and depth-map was found empirically, ensuring the bounding box of the depth-map covers each pipe. This was done by roughly finding the x-valued pixel on RGB that corresponds to the zero valued pixel on the depth map.

Due to the RGB camera having a wider field of view than the depth map, objects detected on the far sides has to be rejected. The depth map also has a higher field of view than the RGB, making the area limited by the RGB height and IR width, the effective usable area. The rough cut-out, will affect the segmentation negatively as will be described later.

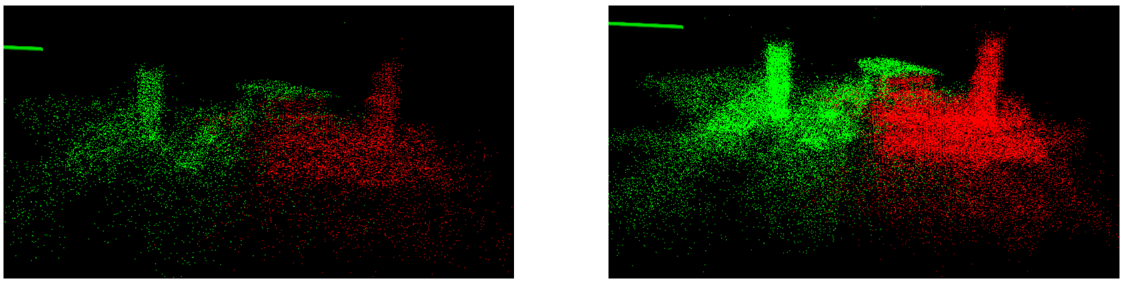
Since the point cloud from each camera is created in its own locale frame, the different point clouds have to be transformed. The parameters for the different transformation matrices was given. On Fig. 32 this transformation can be seen.


(a) Point clouds from the different cameras in a common camera frame.

(b) Point clouds transformed into the global frame.

Figure 32: Transforming the point clouds to the global frame.

After transforming the point clouds to a global frame, they were combined. Even though the camera network gives a more complete point cloud of the scene, the resolution provided by the Kinect is good enough to capture the shape of the small pipes. In order to get a better point cloud to work with, multiple layers of point clouds are appended to each other. On Fig. 33a there is only one layer, but on Fig. 33b five layers have been joined together. This creates a better point cloud to work with, capturing the pipe shape better although the fine details are still left out.


(a) Pin-end and box-end using one point cloud layer

(b) Pin-end and box-end using five point cloud layers

Figure 33: Point cloud of pipe-ends using concatenated layers.

3.4.4 Pipe Segmentation

Before pose estimation, it is important to separate the pipe points in order to make the pipe alignment more efficient. Two methods were evaluated, the first method uses a square around the pipe to exclude all points outside. The other method is segmenting the pipe by using RANSAC to recognize primitive shapes, which are then excluded or retained.

The first method assumes the pipes are placed on the ground, in a vertical/angled orientation. The aim is to find a square that is parallel to the ground and is centered around the pipes. To calculate the square's position, the x and y coordinate of pipe-cloud's center has to be calculated. This is done by excluding every point outside the height range of 0.45m to 1.5m and the rest points are used for calculating x and y position of the pipe. In this case the square used for excluding the unnecessary points is of the dimension 0.20x0.20m, a little wider than the pipes' diameter. This is not a good method for segmenting the pipe cloud, and will not work if there are other objects close to the pipe in the said height range. In that case, the square will most likely be placed outside the pipe and exclude the pipe itself. One option to solve this is to improve the cut-out of pin-end and box-end to only cover the pipe-ends when creating the point cloud. Although, if the perfect cut-outs contain any obstructions close to the pipes, this method will not work.

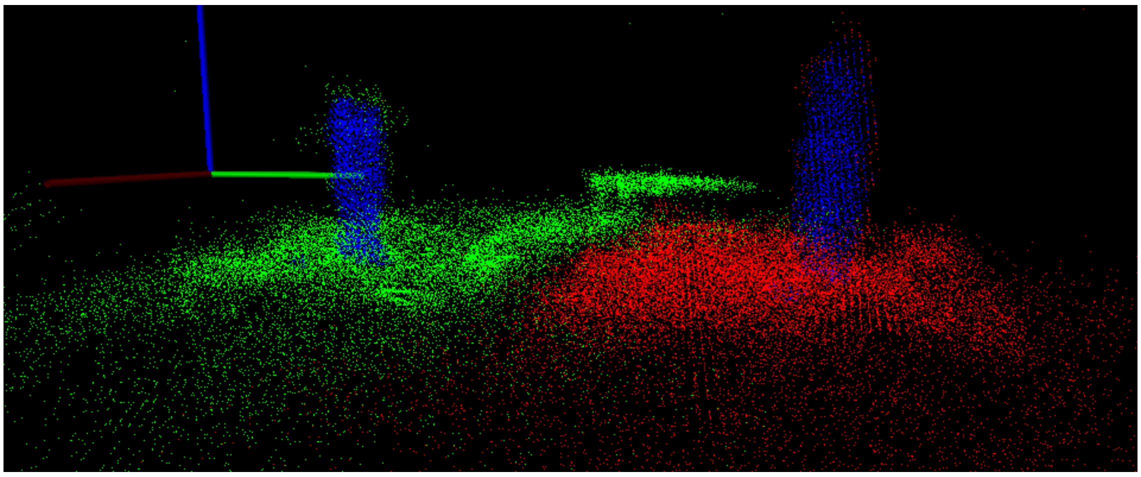
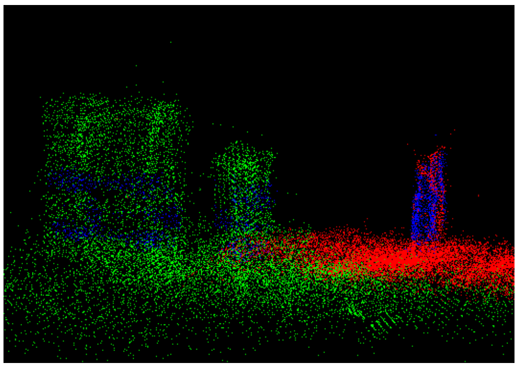


Figure 34: Segmented and filtered box-end(blue) and pin-end(blue), using a square exclusion method.

The second method was aimed to solve the aforementioned problems of the square exclusion approach. The idea is to use RANSAC to find planes, and then exclude the points found in these plane. Unfortunately, because of the inaccurate points, removal of planes leaves behind a substantial amount of scarce points. To solve this, a statistical outlier removal filter was applied. This will remove the remaining points, as well as refine the pipe clouds. Due to the wide spread of points, the standard deviation is high. Therefore the standard deviation multiplier α was set to a value of $\alpha = 0.6$.
Planes are not the only obstacles in the scene, therefore this operation is followed by utilizing RANSAC to find cylinders, and keep these points. The reason being that the pipes have a cylindrical geometry. Sadly this approach did not perform much better. Because of the inaccurate point cloud, RANSAC struggles to recognize cylinders correctly. As can be seen on Fig. 35a, the algorithm removes the ground plane, but the cylinder segmentation is incorrect if there are any larger objects close to the pipes. This method is also slower than the other approach. Although it is not a big improvement in avoiding obstacles, it solves the ground placement limitation.


(a) Segmented points(blue).

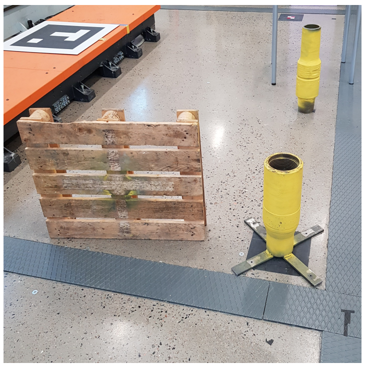

(b) Corresponding image.

Figure 35: RANSAC segmentation

Even though the RANSAC segmentation works slightly better, it was chosen to proceed with the square exclusion method. Due to the fact that the latter preserves most of the pipe points, and will work better with the ICP algorithm as well.

3.4.5 Pose Estimation

With the pipe fully segmented, it is ready to be used for pose estimation. The models are created in Solidworks, based upon roughly measured dimensions of the pin-end and box-end.

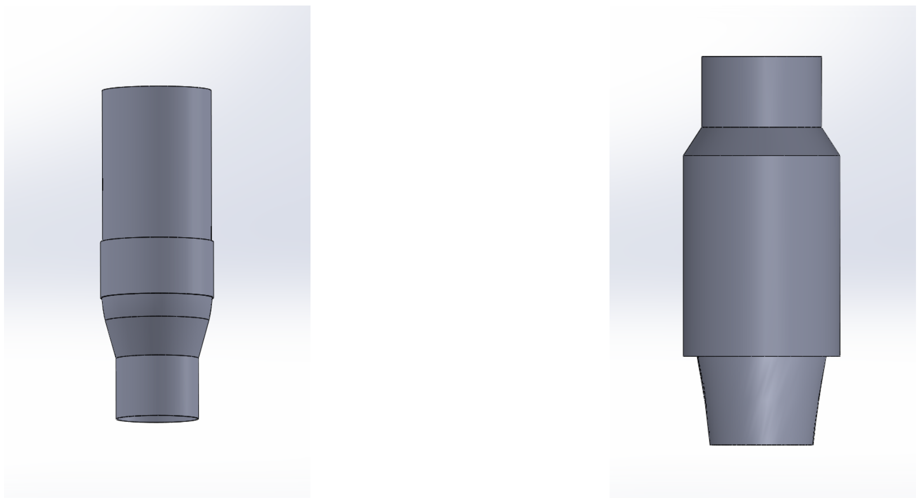

(a) Box-end CAD model (b) Pin-end CAD model

Figure 36: Solidworks models.

The CAD models then had to be converted to point clouds. This was done by using CloudCompare to create uniform distributed point clouds of the models.

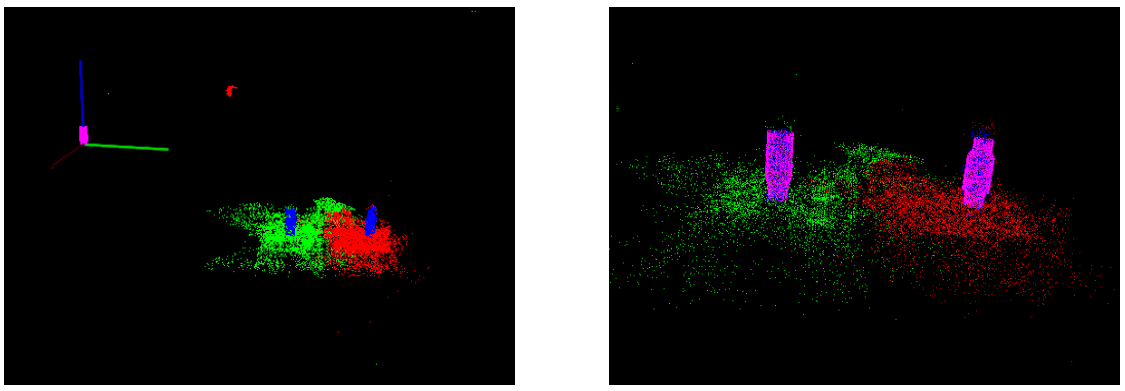
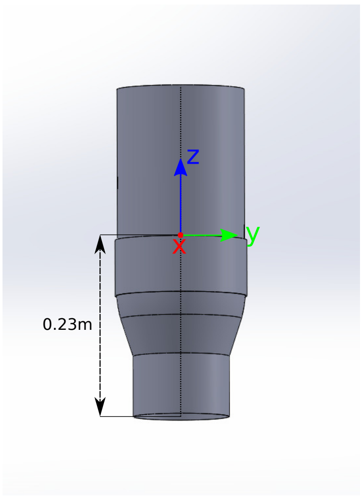

(a) Models(purple) loaded at origin before ICP alignment.
(b) Models(purple) aligned with the segmented point clouds using ICP.

Figure 37: Model alignment using ICP.

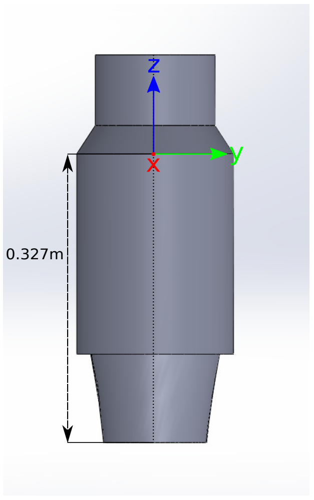
The models are then loaded into the program at the origin as can be seen on Fig. 37a. Before running the ICP algorithm, the models are pre-aligned with their respective point cloud. This is done by calculating the average x, y and z value of the segmented pin-end and box-end point cloud, and then use the corresponding transformation matrix to translate the models to these position. The prealignment is done if the models are more than a set distance away from the segmented clouds. Afterwards, ICP is used to properly align the models with the clouds. Maximum iterations were set to 200, and during each program cycle, the ICP algorithm was applied 10 times in order to get a good fit.

3.5 Test Procedure

To test the system's performance a set of ground truths were measured, and the pipe-ends were places at these coordinates. The coordinate systems seen in Fig. 38, is set to match the global coordinate system, meaning when the pipes are in a vertical position, the pipe rotations in x and y are zero.



(a)



(b)

Figure 38: Pipe-ends' coordinate system.

For the box-end, the position was chosen to be above the neck, through the longitudinal center of the pipe. Similarly, the end of the neck was chosen for the pin-end as well. These positions were chosen because they tend to be constant for the pipes. Due to wear, the pipes has to be grinded and re-threaded which results in varying length of the pipe-ends. The positions given in the following plots is therefore referring to these positions.

Two cases are considered, vertical and angled. The system was left to run for $\approx 15min$ to get multiple estimations, capturing 109 data samples for the vertical case and 95 samples for the angled case. The orientation data was captured by converting the resulting rotation matrix to Euler angles. Due to the master computer's RAM limitation, it was chosen to use three concatenated layers of point clouds in the following results. It was also found that increasing the layers beyond three, did not increase the accuracy of the estimation in any significant way.

4 Results

4.1 Case 1: Vertical

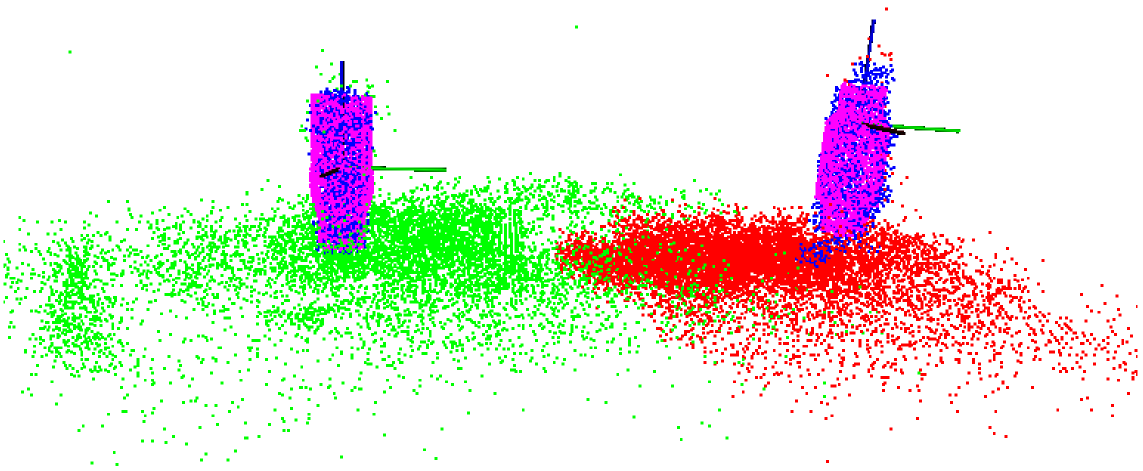


Figure 39: Models(purple) aligned with the segmented points(blue). Points made bigger for illustrative purposes.

	$x[m]$	$y[m]$	$z[m]$	θ_x[deg]	$\theta_y[deg]$	$\theta_z[deg]$
Box-end	4.694	5.3	0.23	0	0	0
Pin-end	4.694	6.8	0.327	0	0	0

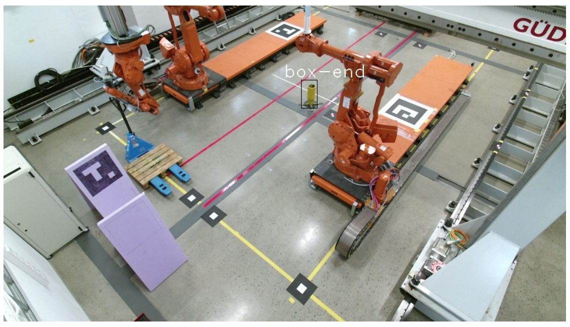
Table 1: Ground truth pose.

Box-end				
	Position μ [m]	Position σ [mm]	Rotation μ[deg]	Rotation σ[deg]
x-axis	4.720	2.1	-4.55	1.31
y-axis	5.320	2.2	-1.50	1.59
z-axis	0.295	3.6	31.2	11.33

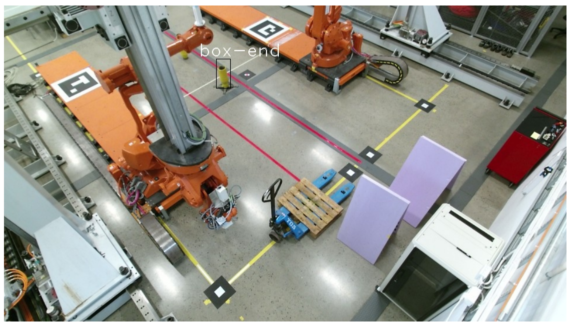
Table 2: Mean μ and standard deviation σ values for position and rotation.

Pin-end				
	Position μ [m]	Position σ [mm]	Rotation μ[deg]	Rotation σ[deg]
x-axis	4.733	4.5	0.80	0.93
y-axis	6.824	4.4	3.52	0.99
z-axis	0.393	6.8	20.89	5.26

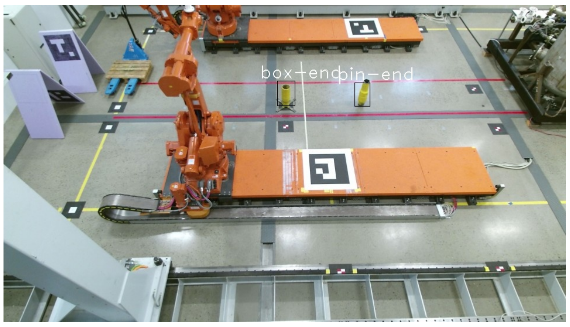
Table 3: Mean μ and standard deviation σ values for position and rotation.



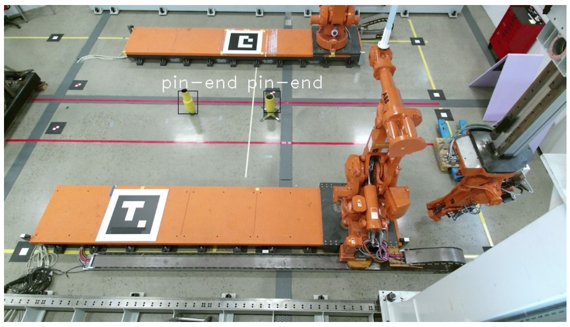
(a) Jetson1 detected objects.



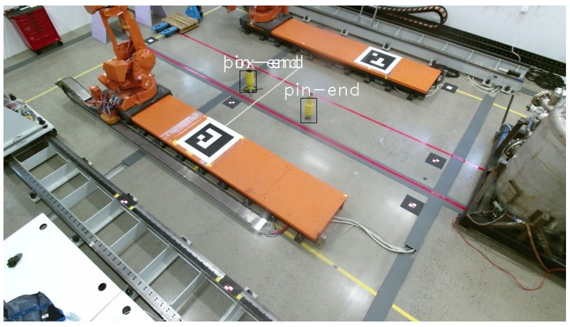
(b) Jetson2 detected objects.



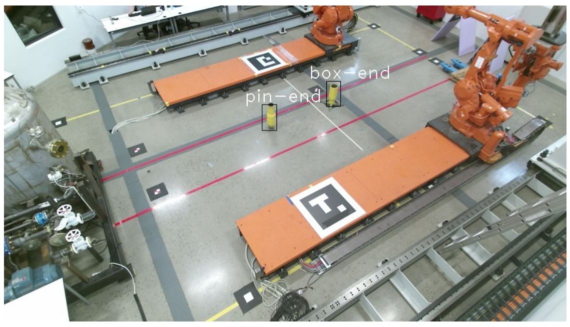
(c) Jetson3 detected objects.



(d) Jetson4 detected objects.

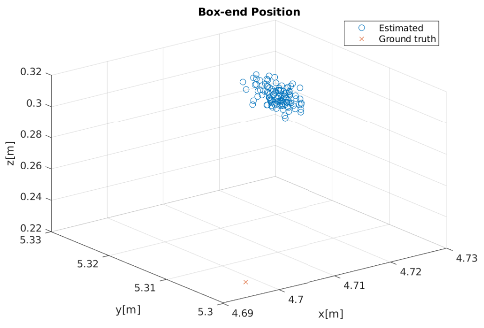


(e) Jetson5 detected objects.

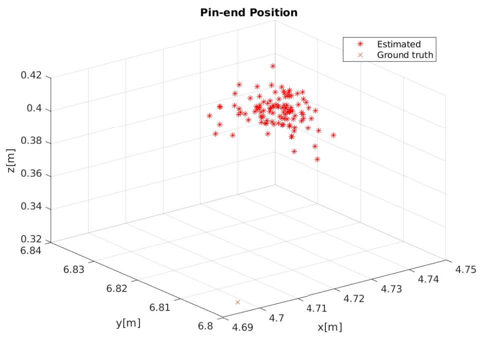


(f) Jetson6 detected objects.

Figure 40: YOLO detections

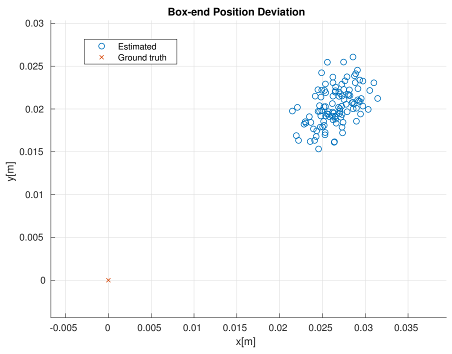


(a)

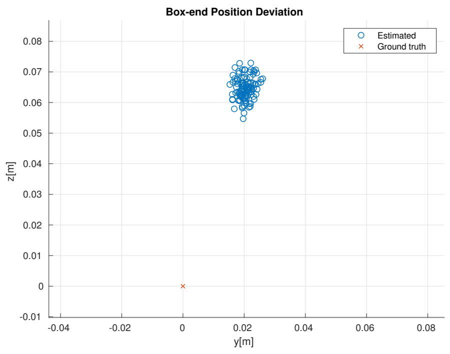


(b)

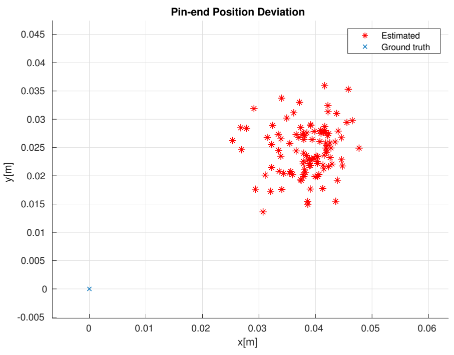
Figure 41: Pipe positions.



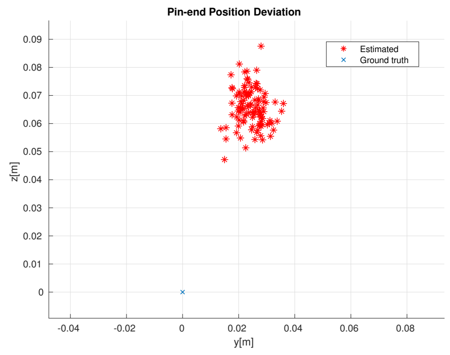
(a)



(b)

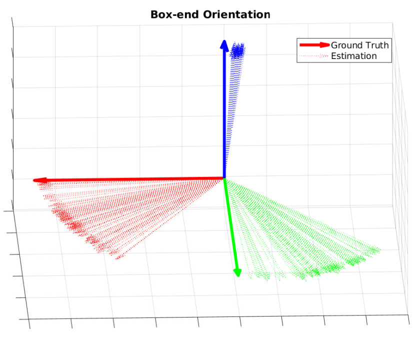


(c)

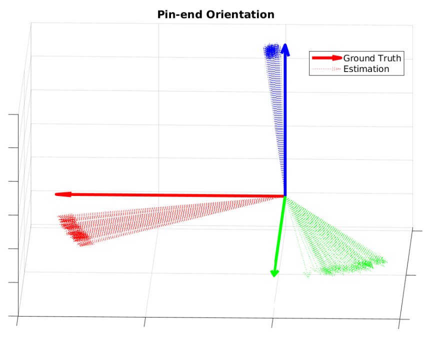


(d)

Figure 42: Estimated position error.

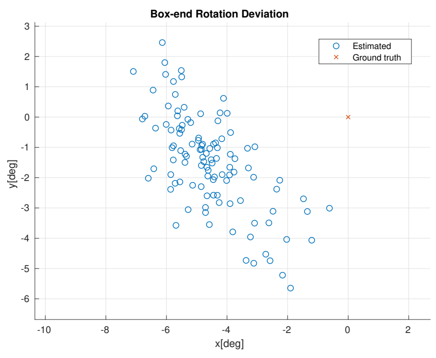


(a) Box-end orientation.

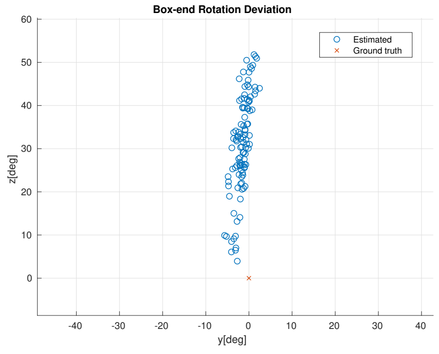


(b) Pin-end orientation.

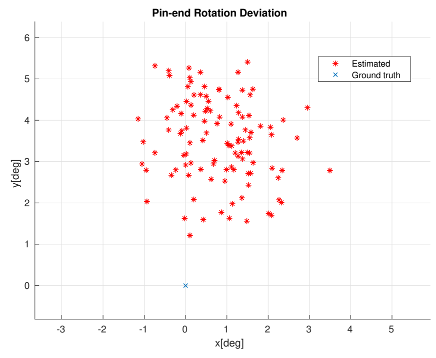
Figure 43: Pipe orientation relative to ground truth.



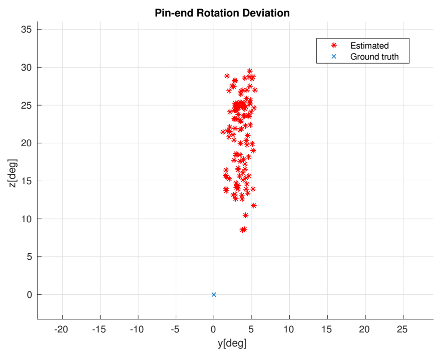
(a)



(b)



(c)



(d)

Figure 44: Estimated pipe orientation error.

4.2 Case 2: Angled

Due to practical reasons, only the box-end was tested in an angled orientation.

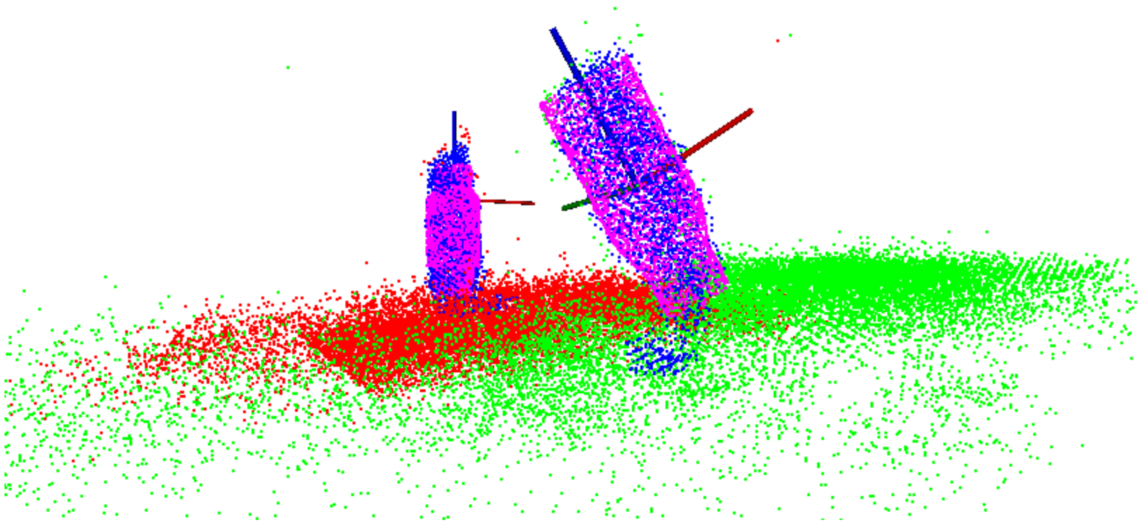


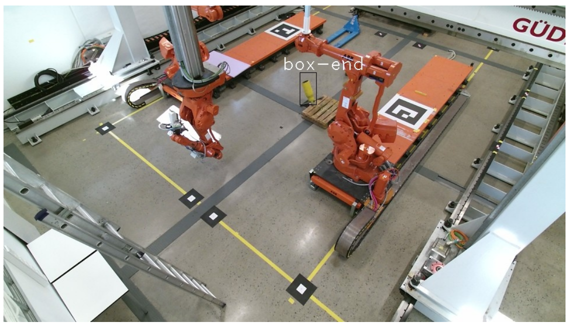
Figure 45: The box-end model(purple) aligned with the segmented point cloud(blue). Points made bigger for illustrative purposes.

	$x[m]$	$y[m]$	$z[m]$	θ_x[deg]	$\theta_y[deg]$	$\theta_z[deg]$
Box-end	4.694	5.3	0.31	0	$\approx$ -30.6	0

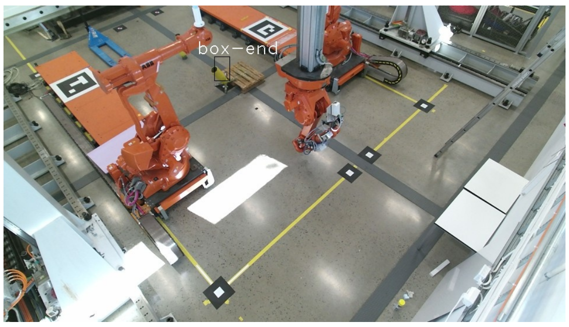
Table 4: Ground truth pose.

Box-end				
	Position μ [m]	Position σ [mm]	Rotation μ[deg]	Rotation σ[deg]
x-axis	4.643	4.7	-1.21	5.73
y-axis	5.321	6.4	-25.51	1.29
z-axis	0.358	3.8	2.06	6.09

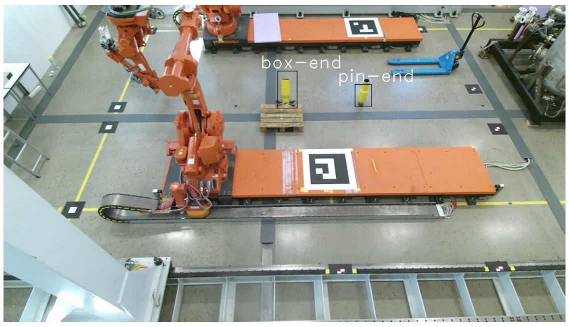
Table 5: Mean μ and standard deviation σ values for position and rotation.



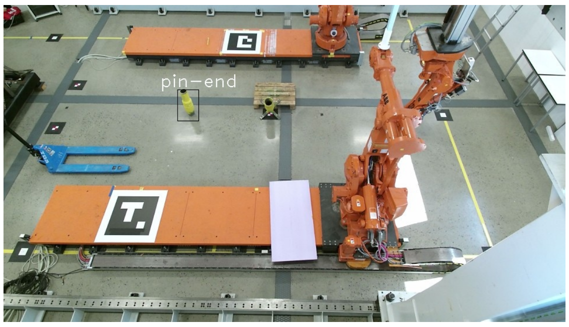
(a) Jetson1 detected objects.



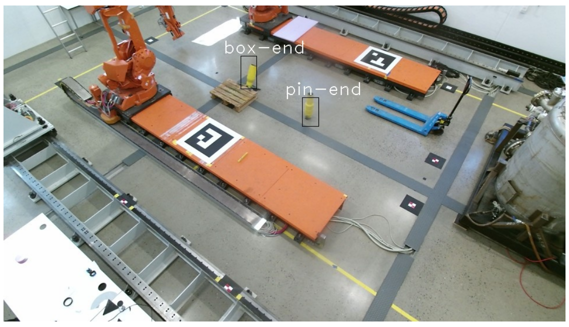
(b) Jetson2 detected objects.



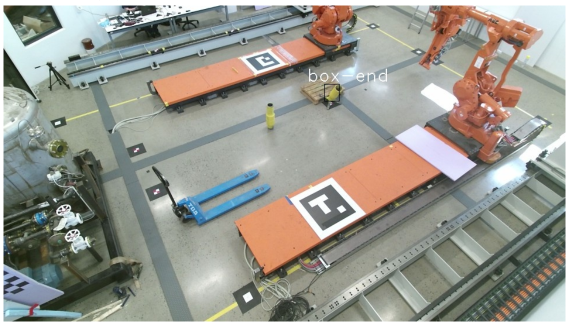
(c) Jetson3 detected objects.



(d) Jetson4 detected objects.



(e) Jetson5 detected objects.



(f) Jetson6 detected objects.

Figure 46: YOLO detections.

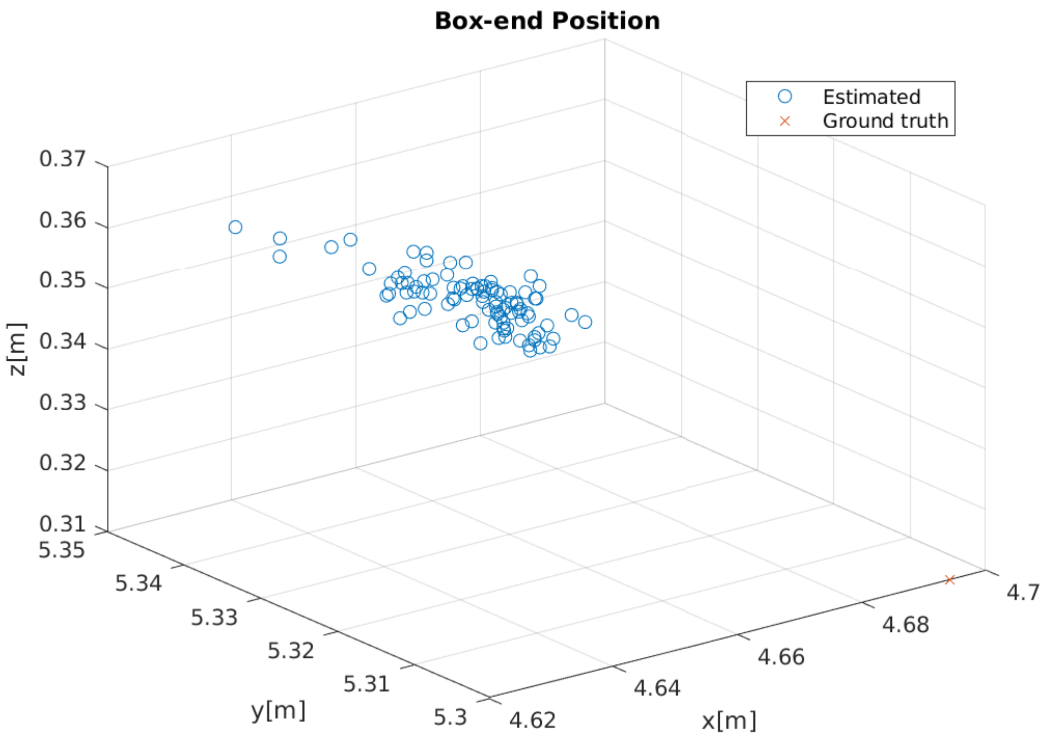
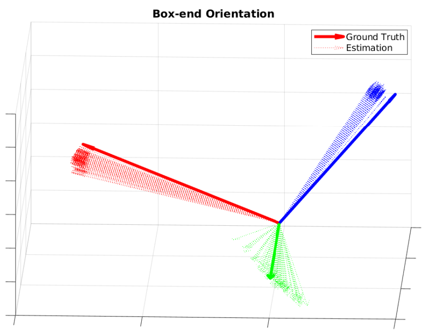
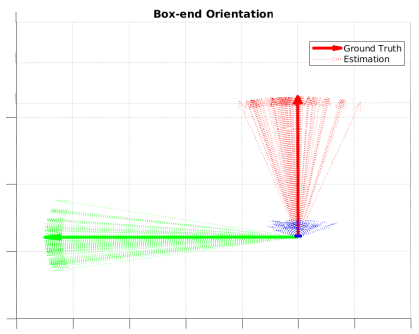


Figure 47: Box-end position.

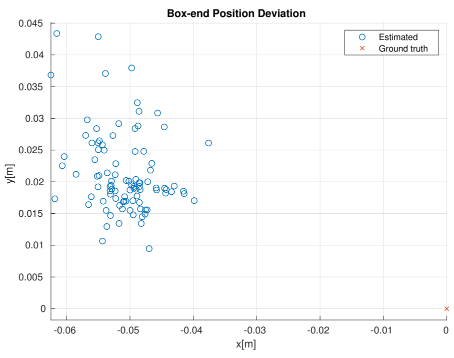


(a)

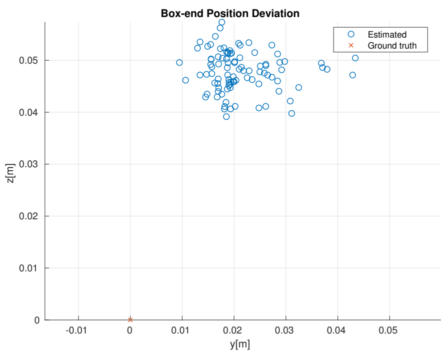


(b)

Figure 48: Box-end estimated orientation and ground truth.

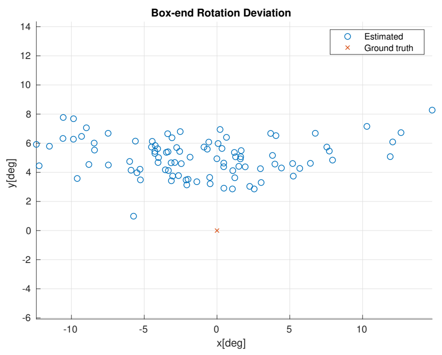


(a)

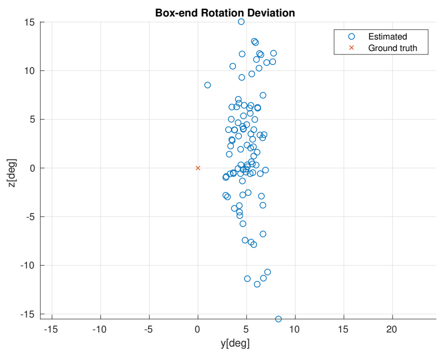


(b)

Figure 49: Estimated box-end position error.



(a)



(b)

Figure 50: Estimated box-end orientation error.

5 Discussion

5.1 Detector Performance

Fig. 40 and 46 gives an indication of the detector performance. The box-end is detected fairly well, but it struggles to detect the pin-end correctly. Unfortunately, the truncated pin-end is fairly similar to the box-end when in a vertical position at these viewpoints. In addition, due to Tiny-YOLOv3 not having the third feature map to do predictions on, the detector is likely to struggle because of the pipe-ends' small size.
As can be seen in the mentioned figures, the detector sometimes detects the box-end as a pin-end. To deal with this, a fail-safe feature was implemented to only accept the detection with the highest confidence if multiple pin-/box-ends are detected. Although, this will not work in the cases where the erroneous detection has the highest confidence. Fortunately this did not happen in the experiments, but it is something to consider. If this happens, some of the box-end would appear in the pin-end cloud, resulting in a segmentation and model alignment that would fail.

The detector only works reasonably well in vertical/angled cases, but performs poorly when the pipes are position horizontally on the ground. There was an attempt to train the network for this, however the detector did not perform well in this case, and experimentation in this case was therefore omitted. In addition, this re-training of the network reduced the performance, making detection on pipes in vertical position as seen on Fig. 40 perform worse. One reason for this performance reduction may be due to the visual similarities of the pipes when one of the them are upside down. This is the case when the pipes are laying horizontally on the ground as the pipe appear inverted for some cameras. To reduce the similarities of the box-end and pin-ends, training with whole pipes and not the truncated pipes used in this thesis, should be performed.

Another source of error that should be noted, is the annotation process. When annotating hundreds of training images, there is a chance of wrongly annotate the different classes, e.g. annotate the box-end as a pipe-end and vice versa. Unfortunately, training on wrongly annotated images decreases the detection accuracy significantly.

The detector would also perform better if there was more training data. There were roughly 500 original images used for training, although with some data augmentation this was increased to 900. When compared with classes in other datasets, e.g the Pascal's VOC datasets, which contains thousands of images per class, the amount of training data for the pipe-ends are low. Although these datasets have significant more variation in the class objects themselves in addition to the background, than the two classes used in this thesis, there is room for improvement by increasing the training data.

During the thesis, multiple weights were used. Initially the detectors were trained for the pipes placed vertical on the ground. However, there was a desire to train the detector for angled and horizontal pipes. The detector were augmented with data from these latter cases, which somewhat helped the angled cases, although it reduced the detector's performance in vertical cases. Therefore it was chosen to use the best performing weights in the vertical case as seen on Fig. 40. While the best performing detector weights were used in the angled case, as seen in Fig. 46. This explains the missing detection of the pin-end in Fig. 46f. It was chosen to use different weights in order to see how well the pose estimation performs with a functional detector.

5.2 Sensors

Due to the limitation of the depth sensors, the accuracy of the system is reduced. Even with the repainted pipes, the corresponding point clouds are not satisfyingly captured. The closest sensor is roughly 5.5m away from the pipe, but other sensors are at a distance of up to 7m. This variable distance from the pipe to the different sensors will also affect the shape of the resulting point cloud. As can be seen on Fig. 34, there are no continuous surface describing the pipes, but rather resembles two warped solid cylinders. In practice, the current system would not work due to the poor accuracy when using pipes with normal appearance. To solve this a better depth sensor is required.

5.3 Segmentation

The current segmentation procedure is maybe the biggest challenge when it comes to system robustness. Because the point cloud cut-out is done very roughly, it brings a lot of unwanted points into the clouds. Even though the square exclusion method is capable of segmenting the pipes standing on the ground with no obstructions, it is not designed for industrial application. On offshore drilling ships, the only case when the pipe is vertical on the ground, is when the box-end is coming through the platform, either going up or down through the oil-well. Otherwise, the pipes are laying on the catwalk or positioned vertically in the air when handled by the Hydraracker. For the latter cases, the square exclusion system will fail.

Segmentation using RANSAC was attempted to solve the robustness against obstruction problem, in addition to remove the ground placement limitation. Sadly, RANSAC struggles to estimate anything more than planes, when working with the inaccurate and noisy cloud. This could be solved by replacing the existing depth cameras with more accurate ones. In addition, experimentation with other RANSAC algorithms as described under Sec. 2.3.1, could give better segmentation. Although it is a significant obstacle in obtaining a robust system, a lot could be gained by improving the point cloud cut-out.

5.4 Pose Estimation

The pose estimation works fairly well when the segmentation is done properly. Looking at the Figs. 42a-42d, there is a bias on all positions. For the box-end, the mean positional error is $x_{Box,posDev} \approx 2.6cm$, $y_{Box,posDev} \approx 2.0cm$ and $z_{Box,posDev} \approx 6.5cm$ respectively, with relative little spread. Most of the detectors were able to capture the box-end, which can be seen on Fig. 40, resulting in a good point cloud. The accuracy in x and y is generally good, however the error in z tend to have a larger bias.
In the case of the pin-end, the mean positional deviation is $x_{Pin,posDev} \approx 3.9cm$, $y_{Pin,posDev} \approx 2.4cm$ and $z_{Pin,posDev} \approx 6.6cm$, with more spread than the box-end.

Some of this bias can be explained by wrongly measured ground truth coordinates. Due to practical reasons, it was hard to match the pipes to the measured coordinates correctly. A more significant error may be due to an incomplete point cloud. As shown in Fig. 24, the model centers itself around the incomplete point cloud. Although in the mentioned figure, the point cloud is worse in terms of the overall shape when compared to the point cloud of multiple cameras. However, if not the pipe is captured uniformly around the symmetric axis z, the model alignment will be biased one way or the other. In the case of box-end, this could explain the larger mean error for x when compared with y. As seen in Fig. 40d, the detector could not locate the box-end, resulting in a point cloud missing the details from this viewpoint. Similarly, this could be the case for the pin-end. Looking at the results in Fig. 42c and 42d there is a larger bias in comparison to the box-end. Some of this bias can be explained by a limited amount of detectors were able to capture the pin-end, as can be seen in Fig. 40.

As seen in Fig. 42b and 42d the bias in z for both the pipe-ends are larger when compared with x and y. The ICP algorithm struggles to align the models correctly due to the sensor inaccuracy, in addition to few points describing the pipe along its longitudinal axis. This is because the pipes' z-axes are not parallel with the sensors. It should be noted that this affects the x and y positioning as well, but the alignment in z axis is especially affected.
In the case of the pipe-end the threaded section is especially noisy. Looking at Fig. 28, the threads are not fully painted. What the figure does not show is the lack of paint on the other side. This leads to fewer and inaccurate points describing these threads, which can be seen on the pin-end cloud in Fig. 39. This results in a segmented pin-end cloud that is shifted above the ground, and in turn an increased positional deviation in z.

When it comes to orientation, the estimation is decent. Looking at the Figs. 44a-44d, the mean rotational deviation in x and y for the box-end/pin-end is $x_{Box,rotDev} \approx -4.5°$, $y_{Box,rotDev} \approx -1.5°$ and $x_{Pin,rotDev} \approx 0.8°$, $y_{Pin,rotDev} \approx 3.5°$. The rotation around z is much worse, with a mean deviation of $z_{Box,rotDev} \approx 31.2°$ and $z_{Pin,rotDev} \approx 21°$. However, due to it being a symmetrical axis, it is not relevant in this case. During alignment, ICP tend to rotate in the models around z, and the z deviation seen on Fig.44b and 44d may be an accumulative error that increases as the program is running.
Probably due to the sensors' inaccuracy, the pipe surface is warping at each iteration. This results in "wiggling", where the models have a slight variation in rotation around x and y-axis. In addition, the varying distance from the sensors to the pipes will give the pipe cloud different warping of the pipes' surface. This could explain why the orientation has the mentioned bias.

The system performs fairly well in the angled case as well, with a mean positional error of $x_{Box,posDev} \approx -5.1cm$, $y_{Box,posDev} \approx 2.1cm$ and $z_{Box,posDev} \approx 4.8cm$. Because measuring the ground truth being slightly harder, the measurement is likely to be more erroneous than the vertical case. When compared with the results in the vertical case, the positional error in x is standing out. Some of this bias can be explained by the detector in Fig. 46d is not able to locate the box-end, leading to a non-complete point cloud. In addition, because of problems locating the exact center of the pipe when the measuring ground truth value for x, it is likely to be incorrect. However, all the detectors are capturing the sides of the pipe, resulting in a small deviation for y. In addition, the positional error in z is performing better than the vertical case. This is due to the box-end is positioned almost parallel for some of the sensors, giving a better point resolution at the neck and a better alignment with the box-end's longitudinal axis. It should be noted that the variance is larger than in the vertical case, this is probably a result of the underside not being detected, in addition to the square exclusion method somewhat concatenates the cloud.

When it comes to the box-end orientation in the angled case, the mean rotational errors are $x_{Box,rotDev} \approx -1.21°$, $y_{Box,rotDev} \approx 5.09°$ and $z_{Box,rotDev} \approx 2.06°$, performing similar to vertical case when it comes to mean deviation. Looking at Fig. 50a, the rotational variance around the x-axis is significantly larger than the y-axis. Again, due to the sensor's inaccuracy, the point cloud surface appears scaled and warped when compared with the model, this may give room for "wiggling". From Fig. 48a, 48b and 50b, the rotation around z is still present, although smaller when compared with the vertical case.

It should be noted that the ICP algorithm requires a good prealignment for the pose estimation to work. This is especially the case when working with a noisy point cloud, such as the ones used in this thesis. This can be seen in Fig. 51 in Appx. A, where the box-end model has a suboptimal prealignment.

5.5 Runtime Speed

The program cannot be considered a real-time system. Although Tiny-YOLOv3 runs fast on Jetson at $\approx 14FPS$, the rest of the system is slow. Depending on the amount of concatenated layers, the point cloud construction period is long. For three layers, as used in the results, the time building the clouds are $5 - 6s$. The segmentation and pose estimation uses $\approx 5s$ each cycle, although this can be reduced. For a good fit, ICP is run multiple times to guarantee a satisfying alignment. This is redundant if the prealignment is good, but is important in the cases where the model and pipe cloud has a significant deviance in orientation and position after prealignment.
In industrial application, the hardware is likely to be more powerful than the one used in this thesis(see Sec. 3.2 for hardware), increasing the runtime speed. However, there is a lot to be gained by improving the performance of the program itself. In Sec. 6.1, several improvements are proposed, which should make the program run faster.

6 Conclusion

In this thesis, various different techniques for object detection, segmentation and alignment has been evaluated. An overview of ROS and point clouds creation are provided. In addition, a basic presentation of machine learning with CNNs has been held, along with a deeper explanation of the machine learning network YOLOv3. Furthermore, different model estimators and pose estimating algorithms for point clouds, such as ICP and various RANSAC versions are presented.

Based on these techniques, a pose estimating system for stationary pipe-ends has been made. The system utilizes a sensor node network where each node is equipped with a Jetson TX2 computer combined with a Kinect v2 sensor. For communication between the sensor nodes and the master computer, the framework ROS is used. Tiny-YOLOv3 are implemented on each sensor node which performs pipe detection on 2D images provided by the Kinect. Further, a pose estimating ROS node was created in C++. This node takes the detections from the multiple sensor nodes and uses the depth map to create point clouds cut-out of the pipes. These point cloud cut-outs from each sensor node are then combined into two point clouds, one for each pipe. Further segmentation is done in order to remove any unnecessary points, and lastly pose estimation using ICP is performed.

For the box-end, the pose estimator managed to achieve a small mean positional error of $\approx 2.6cm$, $\approx 2.0cm$ and $\approx 6.5cm$ for x,y and z respectively. The mean rotational deviation is $\approx -4.5°$, $\approx -1.5°$ around the x- and y-axis. The pin-end is slightly worse with a mean positional error of $\approx 3.9cm$, $\approx 2.4cm$ and $\approx 6.6cm$ for x, y and z. The estimated mean orientation error was found to be $\approx 0.8°$, $\approx 3.5°$ around the x- and y-axis. The box-end was also tested in an angled case, where the pose estimator achieved a mean positional error of $\approx -5.1cm$, $\approx 2.1cm$ and $\approx 4.8cm$ for x, y and z. In this case the mean rotational deviation was $\approx -1.21°$, $\approx 5.09°$ for x and y respectively. This proof of concept is not fit for industrial use, but several changes are proposed which increases the speed and accuracy.

6.1 Further Work

Although the system functions in the testing environment used in this thesis, several changes should be done in order to use it in industrial environments. This includes improving the system speed, pose estimation accuracy and adapt the detector for industrial sites.

First-most, an upgrade of the depth sensor would benefit the system's accuracy, in addition to increase segmentation methods such as RANSAC. An increased accuracy and resolution of the depth map, will give better point clouds to do the pipe alignment with. This should result in a significant improvement to pose estimation accuracy, with less variance. Further, as the sensors improves, a transition to drill pipes with natural metallic and rusty appearance should be possible.
It would be interesting to see how the system performs on newer RGB-D sensors such as the new Kinect sensor, which at the time of writing are just around the corner. This sensor provides higher resolution depth maps and better accuracy, properties which the system would greatly benefit from.

In addition, further training of the detector has to done in order to get an unified detector which performs well in different orientations and positions not used in this thesis. If the system is to be used in industrial applications, training data has to be captured from the site. Tiny-YOLOv3 can be replaced by other machine learning networks that are more accurate. Because the detector is much faster than the rest of the system, sacrificing some speed for better detections will improve the system's accuracy without slowing the system down in any significant way.

The segmentation should be improved by changing the current point cloud cut-out procedure. By constructing a correct mapping from QHD image to depth map coordinates, the cut-out becomes significantly smaller, improving both the system speed and memory usage. This will in turn help the segmentation as well, by removing a lot of unwanted objects from the point cloud. Ideally, the improved cut-out will only provide the pipe-end points, but in the case the detectors are trained for small obstructions, further segmentation has to be done. However, the reduced points will increase the segmentation speed. In addition, more accurate sensors will make the use of algorithms such as RANSAC more efficient. Further experimentation with other segmentation algorithms should also be experimented with.

In order to improve the system speed, a modification to the point cloud creation is needed. By moving the point cloud creation to the Jetson itself, the system becomes faster. The process of creating the cut-out point cloud for each individual detector is fast, therefore an idea is to let each Jetson do the cut-out, transform it to the global frame, and at last transfer the point cloud over a ROS topic. In this configuration, the master computer only has to fuse the point clouds together. This will also make the system scalable, making it easy to add more sensor nodes to the system, covering larger areas, in addition giving the point cloud more detail. Transferring point clouds over a network may be demanding, but by improving the object cut-out, less unnecessary details in the scene will be captured. Alternatively, a down-sampling filter could be applied in order to reduce the object resolution.

Further, an option is to expand the sensor network by attaching a camera to the iron roughneck, similarly to the system explained earlier in this thesis. By enhancing the point cloud with a sensor this close, the increased details will increase the system's accuracy. Although, this will only give a better resolution on one side of the pipe, the longitudinal alignment will be better, similarly to what can be seen in Fig. 24.

7 References

[1] NOV and Seadrill. West Aquila MMC Standbreaking with Dual Hydraracker, 2017. URL https://www.youtube.com/watch?v=Ru4MDT-4n9Q.

[2] Noel.martignoni. ROS-master-node-topic. URL https://commons.wikimedia.org/w/index.php?curid=46655352.

[3] Kjell Magne Fauske. Neural network — TikZ example. URL http://www.texample.net/tikz/examples/neural-network/.

[4] Aphex34. Convolutional neural network, . URL https://commons.wikimedia.org/w/index.php?curid=45679374.

[5] Aphex34. Max pooling, . URL https://commons.wikimedia.org/w/index.php?curid=45673581.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 6 2015. URL http://arxiv.org/abs/1506.02640.

[7] Yin Zhou and Oncel Tuzel. VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection. Technical report.

[8] Charles R. Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J. Guibas. Frustum PointNets for 3D Object Detection from RGB-D Data. 11 2017. URL http://arxiv.org/abs/1711.08488.

[9] Matos11. Seal. URL https://pixabay.com/photos/seal-animal-water-hairy-3585727/.

[10] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. 4 2018. URL http://arxiv.org/abs/1804.02767.

[11] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. 12 2016. URL http://arxiv.org/abs/1612.03144.

[12] DrBob Pbroks13. Pinhole camera. URL https://commons.wikimedia.org/w/index.php?curid=4099853.

[13] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, Mihai Dolha, and Michael Beetz. Towards 3D Point cloud based object maps for household environments. *Robotics and Autonomous Systems*, 56(11):927–941, 11 2008. ISSN 09218890. doi: 10.1016/j.robot.2008.08.005. URL https://linkinghub.elsevier.com/retrieve/pii/S0921889008001140.

[14] Msm. Fitted line. URL https://commons.wikimedia.org/w/index.php?curid=2071406.

[15] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast Point Feature Histograms (FPFH) for 3D registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217. IEEE, 5 2009. ISBN 978-1-4244-2788-8. doi: 10.1109/ROBOT.2009.5152473. URL http://ieeexplore.ieee.org/document/5152473/.

[16] Robert Yarnall Richie. Roughneck connecting drilling pipe. URL https://commons.wikimedia.org/w/index.php?curid=53462523.

[17] Joshua Doubek. Iron roughneck. URL https://commons.wikimedia.org/w/index.php?curid=27167596.

[18] Nodes - ROS Wiki. URL http://wiki.ros.org/Nodes.

[19] Master - ROS Wiki. URL http://wiki.ros.org/Master.

[20] Messages - ROS Wiki, . URL http://wiki.ros.org/Messages.

[21] Topics - ROS Wiki. URL http://wiki.ros.org/Topics.

[22] WritingPublisherSubscriber(c++) - ROS Wiki. URL http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29.

[23] message_filters - ROS Wiki, . URL http://wiki.ros.org/message_filters.

[24] Callbacks and Spinning - ROS Wiki. URL http://wiki.ros.org/roscpp/Overview/CallbacksandSpinning.

[25] Jason Brownlee. How Machine Learning Algorithms Work (they learn a mapping of input to output). URL https://machinelearningmastery.com/how-machine-learning-algorithms-work/.

[26] Artificial neuron. URL https://en.wikipedia.org/wiki/Artificial_neuron.

[27] Anish Singh Walia. Activation functions and its types-Which is better? URL https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f.

[28] Andrej Karpathy. CS231n Neural Networks, . URL http://cs231n.github.io/neural-networks-1/.

[29] Andrej Karpathy. CS231n Convolutional Networks, . URL http://cs231n.github.io/convolutional-networks/.

[30] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2 2015. URL http://arxiv.org/abs/1502.03167.

[31] Convolutional Neural Network: Pooling Layer, . URL https://en.wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer.

[32] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for Simplicity: The All Convolutional Net. 12 2014. URL http://arxiv.org/abs/1412.6806.

[33] Convolutional Neural Network: Loss layer, . URL https://en.wikipedia.org/wiki/Convolutional_neural_network#Loss_layer.

[34] Eugenio Culurciello. The History of Neural Networks - Dataconomy. URL https://dataconomy.com/2017/04/history-neural-networks/.

[35] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. 12 2016. URL http://arxiv.org/abs/1612.08242.

[36] Sha Luo, Huimin Lu, Junhao Xiao, Qinghua Yu, and Zhiqiang Zheng. Robot detection and localization based on deep learning. In *Proceedings - 2017 Chinese Automation Congress, CAC 2017*, volume 2017-Janua, pages 7091–7095. IEEE, 10 2017. ISBN 9781538635247. doi: 10.1109/CAC.2017.8244056. URL `http://ieeexplore.ieee.org/document/8244056/`.

[37] Zachary Singer. Understanding Machine Learning on Point Clouds through PointNet++, 2019. URL `https://towardsdatascience.com/understanding-machine-learning-on-point-clouds-through-pointnet-f8f3f2d53cc3`.

[38] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. PointCNN: Convolution On X-Transformed Points. 1 2018. URL `http://arxiv.org/abs/1801.07791`.

[39] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. 6 2017. URL `http://arxiv.org/abs/1706.02413`.

[40] CyberAILab. A Closer Look at YOLOv3. URL `https://www.cyberailab.com/home/a-closer-look-at-yolov3`.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. 12 2015. URL `http://arxiv.org/abs/1512.03385`.

[42] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 6 1981. ISSN 00010782. doi: 10.1145/358669.358692. URL `http://portal.acm.org/citation.cfm?doid=358669.358692`.

[43] J Matas and O Chum. Randomized RANSAC with Td,d test. *Image and Vision Computing*, 22(10):837–842, 9 2004. ISSN 0262-8856. doi: 10.1016/J.IMAVIS.2004.02.009. URL `https://www.sciencedirect.com/science/article/pii/S0262885604000514`.

[44] O. Chum and J. Matas. Matching with PROSAC Progressive Sample Consensus. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 220–226. IEEE. ISBN 0-7695-2372-2. doi: 10.1109/CVPR.2005.221. URL `http://ieeexplore.ieee.org/document/1467271/`.

[45] Rahul Raguram, Jan-Michael Frahm, and Marc Pollefeys. A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus. pages 500–513. 2008. doi: 10.1007/978-3-540-88688-4{_}37. URL `http://link.springer.com/10.1007/978-3-540-88688-4_37`.

[46] Changhyun Choi and Henrik I. Christensen. RGB-D object pose estimation in unstructured environments. *Robotics and Autonomous Systems*, 75:595–613, 1 2016. ISSN 0921-8890. doi: 10.1016/J.ROBOT.2015.09.020. URL `https://www.sciencedirect.com/science/article/pii/S0921889015002158?via%257B%255C%25%257D3Dihub`.

[47] Elise Lachat, Hlne Macher, Tania Landes, and Pierre Grussenmeyer. Assessment and calibration of a RGB-D camera (Kinect v2 Sensor) towards a potential use for close-range 3D modeling. *Remote Sensing*, 2015. ISSN 20724292. doi: 10.3390/rs71013070.

[48] Hamed Sarbolandi, Markus Plack, and Andreas Kolb. Pulse Based Time-of-Flight Range Sensing. *Sensors (Basel, Switzerland)*, 18(6), 5 2018. ISSN 1424-8220. doi: 10.3390/s18061679. URL `http://www.ncbi.nlm.nih.gov/pubmed/29882901http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=PMC6022202`.

[49] Kyle Simek. Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix , . URL `http://ksimek.github.io/2012/08/22/extrinsic/`.

[50] Kyle Simek. Dissecting the Camera Matrix, Part 3: The Intrinsic Matrix , . URL `http://ksimek.github.io/2013/08/13/intrinsic/`.

[51] Thomas Opsahl. Lecture 5.3 Camera Calibration, 2016. URL `https://www.uio.no/studier/emner/matnat/its/UNIK4690/v16/forelesninger/lecture_5_3_camera_calibration.pdf`.

[52] Thiemo Wiedemeyer. IAI Kinect2, 2015. URL `https://github.com/code-iai/iai_kinect2`.

[53] Damien Lefloch, Rahul Nair, Frank Lenzen, Henrik Schäfer, Lee Streeter, Michael J. Cree, Reinhard Koch, and Andreas Kolb. Technical Foundation and Calibration Methods for Time-of-Flight Cameras. pages 3–24. Springer, Berlin, Heidelberg, 2013. doi: 10.1007/978-3-642-44964-2{_}1. URL `http://link.springer.com/10.1007/978-3-642-44964-2_1`.

[54] P.J. Besl and Neil D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 2 1992. ISSN 0162-8828. doi: 10.1109/34.121791. URL `http://ieeexplore.ieee.org/document/121791/`.

[55] Olivier D. Faugeras and Martial Hebert. A 3-D Recognition and Positioning Algorithm Using Geometrical Matching Between Primitive Surfaces. *undefined*, 1983. URL `https://www.semanticscholar.org/paper/A-3-D-Recognition-and-Positioning-Algorithm-Using-Faugeras-Hebert/db393e1758be80deeb0e099389adfe1de80bfe24`.

[56] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-Squares Fitting of Two 3-D Point Sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9 (5):698–700, 9 1987. ISSN 0162-8828. doi: 10.1109/TPAMI.1987.4767965. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=4767965`.

[57] Documentation - Point Cloud Library (PCL). URL `http://pointclouds.org/documentation/tutorials/pfh_estimation.php#pfh-estimation`.

[58] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 643–650. IEEE, 12 2008. ISBN 978-1-4244-2286-9. doi: 10.1109/ICARCV.2008.4795593. URL `http://ieeexplore.ieee.org/document/4795593/`.

[59] Anders Glent Buch, Dirk Kraft, Joni-Kristian Kamarainen, Henrik Gordon Petersen, and Norbert Kruger. Pose estimation using local structure-specific shape and appearance context. In *2013 IEEE International Conference on Robotics and Automation*, pages 2080–2087. IEEE, 5 2013. ISBN 978-1-4673-5643-5. doi: 10.1109/ICRA.2013.6630856. URL `http://ieeexplore.ieee.org/document/6630856/`.

[60] Gregory G Slabaugh. Computing Euler angles from a rotation matrix. 6(2000):39–63, 1999. URL `http://www.gregslabaugh.net/publications/euler.pdf`.

[61] Erlend Buan. Kinect Network Pose Estimator, 2019. URL `https://github.com/jenbu/kinect_network_pose_estimator`.

[62] Erlend Buan and Marko Bjelonic. jenbu/Yolo_pipe_detector. URL `https://github.com/jenbu/Yolo_pipe_detector`.

[63] Erlend Buan. Annotation Tools, 2019. URL `https://github.com/jenbu/Annotation_tools`.

[64] Lingzhu Xiang, Florian Echtler, Christian Kerl, Thiemo Wiedemeyer, Lars, hanyazou, Ryan Gordon, Francisco Facioni, laborer2008, Rich Wareham, Matthias Goldhoorn, alberth, gaborpapp, Steffen Fuchs, jmtatsch, Joshua Blake, Federico, Henning Jungkurth, Yuan Mingze, vinouz, Dave Coleman, Brendan Burns, Rahul Rawat, Serguei Mokhov, Paul Reynolds, P.E. Viau, Matthieu Fraissinet-Tachet, Ludique, James Billingham, and Alistair. libfreenect2: Release 0.2. 4 2016. doi: 10.5281/ZENODO.50641. URL `https://zenodo.org/record/50641#.XIl3iYUo9hE`.

[65] Atle Aalerud, Joacim Dybedal, and Geir Hovland. Automatic Calibration of an Industrial RGB-D Camera Network Using Retroreflective Fiducial Markers. *Sensors*, 19(7):1561, 3 2019. ISSN 1424-8220. doi: 10.3390/s19071561. URL `https://www.mdpi.com/1424-8220/19/7/1561`.

[66] Marko Bjelonic. YOLO ROS: Real-Time Object Detection for ROS, 2018. URL `https://github.com/leggedrobotics/darknet_ros`.

[67] Pjreddie and AlexeyAB. darknet. URL `https://github.com/AlexeyAB/darknet`.

Appendix A Additional Figures

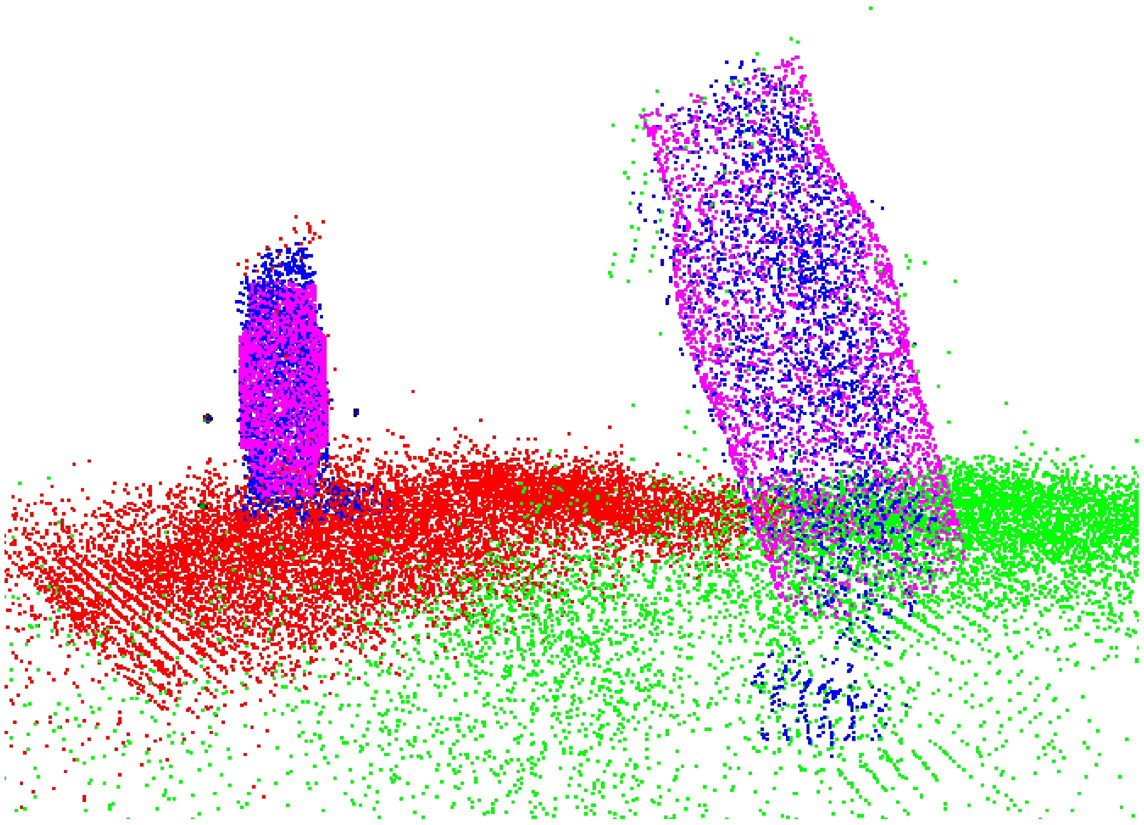


Figure 51: Misaligned box-end due to bad prealignment.

Appendix B Master Thesis Description

MASTER'S THESIS PROPOSAL - SPRING 2019

Pose Estimation of Drilling Pipe Using a 3D-Sensor Network

Atle Aalerud, Geir Hovland and Baltasar Beferull-Lozano
atle.aalerud@uia.no, geir.hovland@uia.no, baltasar.beferull@uia.no

1 INTRODUCTION

The offshore oil and gas industry is working on increasing the autonomy of offshore operations and many drilling operations are already automated. Moving forward, knowledge of vision technology and 3D mapping is crucial to increase autonomy further.

In this assignment the detection of drill pipe is to be investigated as a human operator is still needed to verify the position of drill pipe to ensure correct handling. Drill pipe is the major component of the drill string. It generally constitutes 90-95% of the entire length of the drill string which is the heavy seamless tubing used to rotate the bit and circulate the drilling fluid when drilling for oil. The drill pipe is a seamless pipe with threaded connections, known as tooljoints (fig. 1). At one end of the pipe there is the box, which has the female end of the connection. At the other end of each length of drillpipe is the male end of the connection known as the pin. The wall thickness and therefore the outer diameter of the tooljoint must be larger than the wall thickness of the main body of the drillpipe in order to accommodate the threads of the connection. Hence the tool joints are clearly visible in the drillstring.

2 PROBLEM DESCRIPTION

Today, it is difficult to know the actual position of drill pipe without visual confirmation from a human operator. If all pipes were the same length, it would be possible to estimate where the ends are. However, as drill pipe may vary in length, an operator is still present. The need of visual confirmation is an obstacle preventing fully autonomous pipe handling operations.

In fig. 2, an example of such operation is shown. The drill pipe lies on a Pipe Catwalk (shuttle that transports pipe horizontally) with the pipe extended to hand-over position.

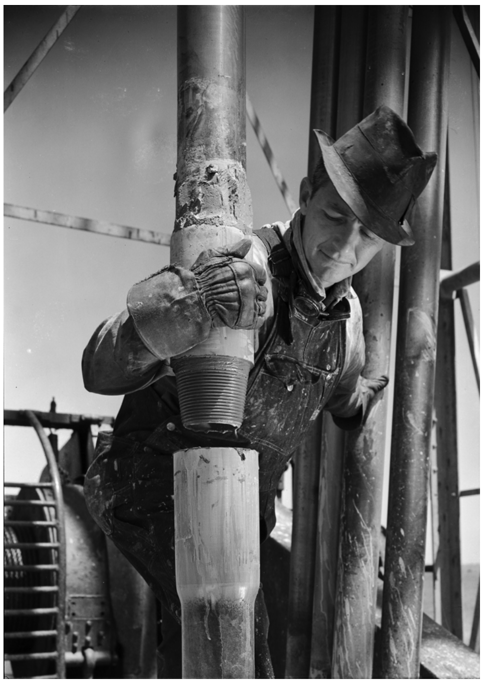


Figure 1. Old fashion roughneck connecting drill pipe. Here he is supporting the pin end tool joint to be placed in the box end of the drill pipe below. (Image, Wikimedia commons.)

This position is visually acknowledged by the operator before the HydraRacker (tall machine for transporting pipe vertically) latch on to the tool joint to hoist the pipe to a vertical position.

The purpose of this assignment is to create an automatic detection of the the tool joint. This would allow the automatic sequence to run continuously without the need for the operator's visual confirmation.

- *Joacim Dybedal and Geir Hovland is with SFI Offshore Mechatronics, Department of Engineering Sciences, University of Agder.*
- *Baltasar Beferull-Lozano is with the WISENET lab, Department of Information and Communication Technology, University of Agder.*

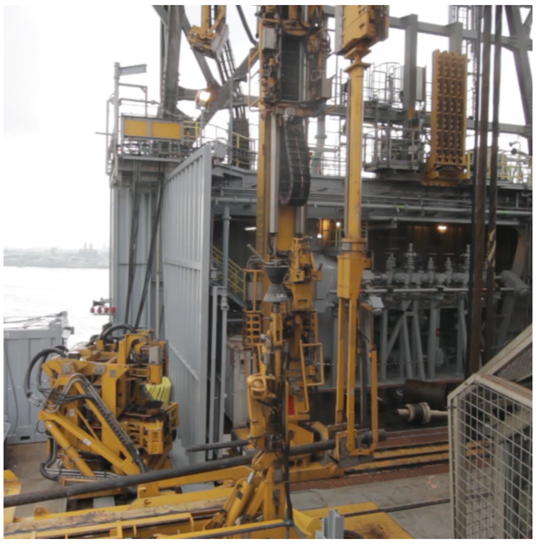


Figure 2. Modern pipe handling using Multi Machine Control (MMC) from NOV. The operator visually confirms the pipe hand-over position between robots before resuming automatic sequence. See full video at go.nov.com. (Image, NOV.)

The student(s) should have an interest in computer programming, computer vision and/or machine learning and would benefit from a basic knowledge of Linux (Ubuntu) and C++ programming. However, this is also an unique opportunity to learn new skills that are very heavily in demand in the industry today.

3 Available Resources

SFI Offshore Mechatronics has installed an Industrial Robotics Lab (IRL) in the Mechatronics Innovation Lab (MIL), including two rail-mounted ABB robots and one gantry-mounted ABB robot (fig. 4). IRL is also equipped with six 3D sensor nodes (fig. 3). The sensors map the environment in 3D and can deliver data to other ROS nodes for processing.[1]

The following resources will be available for the student(s):

- Access to the Industrial Robotics Lab in MIL when testing the software.
- Live 3D Sensor data from 6 sensor nodes (RGB, IR, Depth images and Point Clouds). The data can be accessed remotely from the student's workplace.
- A tool joint to be detected by the sensors.
- If the software is based on GSP, the student(s) will have access to expertise developed in the WISENET lab at UiA.

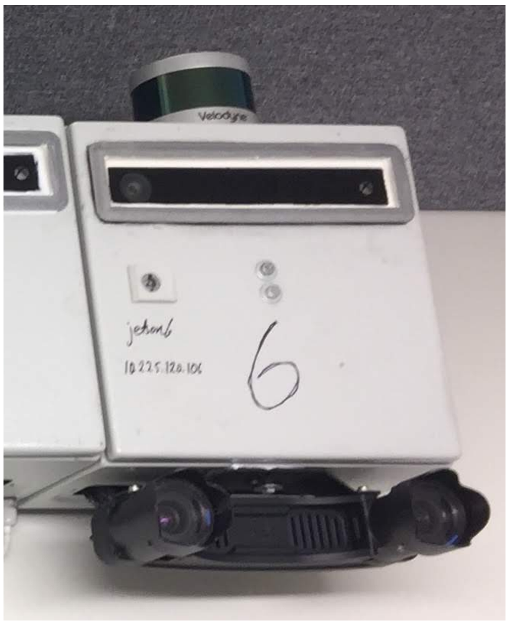


Figure 3. One of the available sensor nodes using a Microsoft Kinect (V2), Carnegie Robotics industrial stereo camera MultiSense S21, Velodyne 16 beam rotating LiDar VLP16 and the embedded computer NVIDIA Jetson TX2.

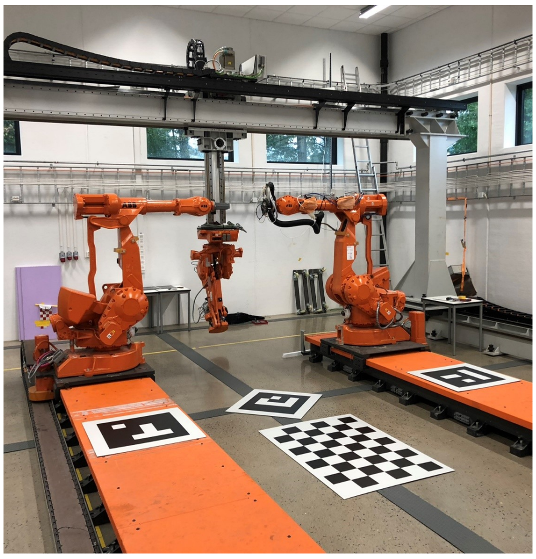


Figure 4. The Industrial Robotics Lab in MIL is used as the example case of an industrial environment. The lab consists of two rail-mounted ABB IRB4400 robots, one ABB IRB2400 robot mounted on a GÜDEL gantry and a processing facility.

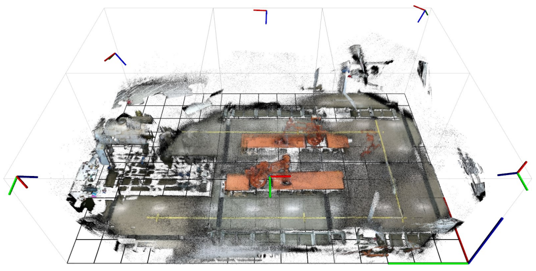


Figure 5. The combined (accumulated) point clouds of 6 Kinect point clouds.

4 Project Goals and Outcomes

The student(s) should investigate different detection techniques including, but not limited to, AI/Neural Network-based detectors (e.g. YOLO, SSD), graph signal processing (GSP) and more classical computer vision algorithms (using e.g. OpenCV, Point Cloud Library, etc.).

Based on the selected techniques, the student(s) should develop a software to detect and track drill pipe, based on 3D sensor data captured in the robotic cell. Detecting position of the box-end tool joint is especially important. The software can use both color images, depth measurements or a combination of both to accomplish this.

The software should be developed as a ROS (Robot Operating System) node that subscribes to real-time sensor data and publish pipe type, position and orientation (pose) for other ROS enabled system to use. The pose should preferably be found in less than one second.

The software should be demonstrated in IRL, MIL. This can be done in collaboration with on-going robotic research such that the robots may interact with the detected pipe.

Summarized, the following should be the outcome of the project:

- Evaluation of detection techniques.
- Create a ROS node capable of quickly detecting and publishing the tool joint pose.
- Industrial demonstration in MIL.

Further, if novelty of results and time allows it:

- Publish results in a conference paper.
- Classify different pipe types.

References

[1] A. Aalerud, J. Dybedal, E. Ujkani, and G. Hovland, "Industrial Environment Mapping Using Distributed Static 3D Sensor Nodes," in *2018 14th IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications (MESA)*. IEEE, 7 2018, pp. 1–6. [Online]. Available: https://ieeexplore.ieee.org/document/8449203/

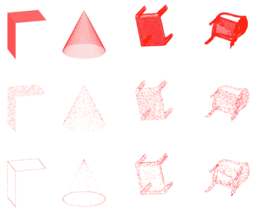


Figure 6. Example of Graph Signal Processing

Graph Signal Processing can take advantage of signal processing techniques (e.g. Graph Fourier Transform and frequency domain filters) by structuring point clouds as graphs. Toolboxes for GSP exists in e.g. Matlab (https://epfl-lts2.github.io/gspbox-html/) and Python (https://pygsp.readthedocs.io/en/stable/).

YOLO (You Only Look Once) is a state-of-the-art, real-time object detection system based on Darknet, an open source neural network framework written in C and CUDA. Read more: https://pjreddie.com/darknet/yolo/

The Robot Operating System (ROS, www.ros.org) is a set of software libraries and tools for building robot applications. It is a modular system where it is easy to create and use software packages which communicate through a standardized interface using ROS messages.

The Point Cloud Library (PCL, http://pointclouds.org/) is a standalone, large scale, open project for 2D/3D image and point cloud processing

OpenCV (Open Source Computer Vision Library, https://opencv.org/) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.

NOV MMC video
https://go.nov.com/#now_showing/video/4b1a0593edf9f92869073c5dd2ae566c

NOV MMC animation
https://go.nov.com/#now_showing/video/6ccb705bd68f10715546ced0ea2cd29b