A Machine Learning and Point Cloud Processing based Approach for Object Detection and Pose Estimation: Design, Implementation, and Validation

SIMON NYLUND AND FREDRIK BRINGAGER

SUPERVISORS

Prof. Frank Y. Li, University of Agder

Asst. Prof. Geir Jevne, University of Agder

Assoc. Prof. Ajit Jha, University of Agder

University of Agder, 2022
Faculty of Engineering and Science
Department of Information and Communications Technology

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven ğğ4-7 og 4-8 og Forskrift om eksamen ğğ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	Nei

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. ğ2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Abstract

This thesis presents an automatic forklift approach for lifting and handling pallets. The project more specifically develops a solution for autonomous object detection and pose estimation by Machine Learning (ML), point cloud processing, and arithmetic calculations. The project is based on a real-life scenario identified together with the industrial partner Red Rock, which includes a forklift operation, where the machine is supposed to identify, lift, and handle pallets autonomously. A key to achieving this automation is to localize and classify the pallet as well as to estimate the Six Dimensional (6D) pose of the pallet, which include its (x, y, z) position and (pitch, roll, yaw) orientation. Positioned directly in front of the pallet, the pose estimation must be performed around the range of 2-meter distance and 0° to ±45° angle.

A systematic solution consisting of two major phases, object detection, and pose estimation, is developed to achieve the project goal. For object detection, the You Only Look Once X (YOLOX)-S ML algorithm is selected and implemented. The algorithm is pre-trained on the COCO dataset. It is, after that transfer, learned on the Logistics Objects in Context (LOCO) dataset to be able to detect pallets in an industrial environment. To improve the detection inference, the algorithm is optimized with the Intel OpenVINO toolkit, resulting in improved inference latency by over 2.5 times on Central Processing Unit (CPU). The output of the YOLOX-S algorithm is a bounding box around the pallet, and a custom struct links object detection and poses estimation together. The pose estimation algorithm converts the Two Dimensional (2D) bounding box data into Three Dimensional (3D) vectors, in which only the relevant points in the point cloud are kept. In contrast, all irrelevant points are filtered out from the environment. A series of arithmetic calculations from the filtered point cloud are applied, including Random Sample Consensus (RANSAC) and vector operations, in which the prior calculates the largest vertical plane of the identified pallet. Based on the object detection output and the pose estimation calculations, a 3D vector and a 3D point resulting in the pallet's pose is found.

Several tests and experiments have been performed to evaluate and validate the developed solution. The tests are based on a developed ground truth setup consisting of an AprilTag marker which provides a robust and precise ground truth measurement. Results from the standstill experiment show that the algorithm can estimate the position within 0.3 and 7.5 millimeters for the x and y axes. Moreover, the z-axis managed to be kept within 1.6 and 28.6 millimeters. The pitch orientation was kept within 3.65° and 5.21°, while the yaw orientation managed to be within 0.86° and 2.64°. Overall standstill test results have evaluated the best and worst case, respectively, within 0° and 45° degrees.

Preface

This thesis is submitted in partial fulfillment of the requirements for the degree of Master of Science in Information and Communications Technology (ICT) at the University of Agder. The project is performed from early January to mid June and is the final part of the Master's program in form of IKT 590 Master's thesis with 30 ECTS credits.

The project is developed in collaboration with Red Rock, which is a maritime robotics company located in Kristiansand, Norway. We would like to thank Red Rock for their contribution to the project and for advises on the road. As Red Rock will use the project results in further development and investigation on the topic of autonomous forklift operations, comprehensive explanation and documentation of the project's development steps are emphasized. Under the execution of the project a larger part of code has been developed, which also is available. The complete source codes for the real time pose estimation and YOLOX-S implementation are avaiable at: *simoen17:realtime_pose_estimation.git* and *simoen17:yolox.git*, respectively.

Finally, we would like to thank our supervisors Frank Y. Li, Geir Jevne, and Ajit Jha for their kind and helpful assistance. It has been inspiring to work with knowledgeable people like you through the Master thesis work. Many thanks.

Simon Nylund and Fredrik Bringager
University of Agder, campus Grimstad
17th of June

Contents

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
6D	Six Dimensional
AI	Artificial Intelligence
AP	Average Precision
CMOS	Complementary Metal-Oxide Semiconductor
CNN	Convolutional Neural Network
COCO	Common Objects in Context
CPU	Central Processing Unit
CVAT	Computer Vision Annotation Tool
DNN	Deep Neural Networks
DOF	Degrees Of Freedom
FoV	Field of View
FPS	Frames Per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HMI	Human Machines Interface
ICT	Information and Communications Technology
IMU	Inertial Measurement Unit
IoU	Intersection over Union
IR	Infrared
LiDAR	Light Detection and Ranging
LiPo	Lithium-Polymer
LOCO	Logistics Objects in Context
ML	Machine Learning
MoCap	Motion Capture
NPU	Neural Processing Unit
ONNX	Open Neural Network Exchange
PCL	Point Cloud Library
PR	Precision-Recall

R-CNN	Regional-Convolutional Neural Network
R&D	Research and Development
Radar	Radio Detection and Ranging
RANSAC	Random Sample Consensus
RGB	Red Green Blue
RGB-D	Red Green Blue Depth
ROI	Region of Interest
ROS	Robot Operating System
RPN	Region Proposal Network
SBC	Single-Board Computers
SDK	Software Development Kit
SLAM	Simultaneous Localization and Mapping
SoM	System-on-Module
SSD	Single Shot MultiBox Detector
ToF	Time-of-Flight
TOS	Terminal Operating System
USB	Universal Serial Bus
VPU	Video Processing Unit
WiFi	Wireless Fidelity
YOLO	You Only Look Once
YOLOX	You Only Look Once X

List of Figures

List of Tables

1. Introduction

The chapter introduces the background and motivation behind the thesis project. It places the problem statement into the context of future autonomous load handling, which requires autonomous object detection and pose estimation. The chapter also presents the research method used to develop and evaluate the project and gives an overview of the report's structure.

1.1 Background and Motivation

The project is developed in collaboration with Red Rock, which is a technology company enabling the future of autonomous load handling. The company is specialized in autonomous applications for the industrial and harsh environments such as harbor and industry operations, logistics, and offshore renewables, as seen in Figure 1.1. To achieve the future of load handling, Red Rock has developed a machine agnostic autonomy platform that is applied across multiple products like cranes, reach stackers, terminal tractors, and forklifts. The platform consists of technology both for autonomous driving and for autonomous lifting and handling of cargo to improve the efficiency and safety of industrial operations. The thesis project is based on handling and lifting operations, utilizing new technology to enhance operations and extend current capabilities.

Statics on operations of forklifts in the United States show that annually over 30 000 serious injuries [1] are happening due to human mistakes and fatigue or other safety-related errors. It is assumed that a substantial number of these accidents could have been avoided by removing people from harm's way with autonomous solutions. Today many of the cargo operations, such as lifting and moving pallets with the help of forklifts, are performed manually by operators. Another challenge of today's cargo operations is the operational window, where many vehicles are only operated during regular working hours and are standing still in the evenings and in the nighttime. This leads to inefficiencies, as all tasks have to be executed during the daytime. As a result of the identified challenge, a number of key motivations are highlighted. The most important one is to develop a solution that supports the transition toward autonomous operations. The results of this thesis project will contribute to achieving autonomous lifting and handling of cargo, which will increase human safety and improve operational efficiency.

Figure 1.1: Overview of Red Rock load handling technologies.

1.2 Problem Statement

The project is defined based on the idea of developing an automatic lifting and handling procedure, where the forklift needs to detect the desired pallet and estimate the correct pose of the pallet to position its lifting tool correctly. Although object detection is a well-studied topic in the literature, how to combine object detection and pose estimation in industrial environments for automatic pallet lifting, where operational robustness and the trade between time and accuracy are keys, is a challenging task. These are important considerations for this project.

To achieve the primary objective, which is to calculate the pose of the object, a 6D representation is desired. The 6D model represents the object's x, y, and z positions and the pitch, yaw, and roll orientations. Together the six parameters generate an accurate understanding of the object's pose. The 6D parameters for both static and dynamic pose defined for the project are seen in Figure 1.2, a short explanation of the defined parameters is described in the list below.

1.2.1 Dynamic Representations

- X translation, representing all dynamic horizontal movements, with right as the positive direction related to the camera frame.
- Y translation, representing all dynamic vertical movements, with down as the positive direction related to the camera frame.
- Z translation, representing all dynamic movements pointing with the camera, positive direction from the camera sensor to the camera lens.
- Yaw rotation for rotation around the Y-axis, following the right-hand rule.
- Roll rotation for rotation around the Z-axis, following the right-hand rule.

- Pitch rotation for rotation around the X-axis, following the right-hand rule.

1.2.2 Static Representations

- X position, representing static horizontal positioning, with right as the positive direction related to the camera frame.
- Y position, representing static vertical positioning, with down as the positive direction related to the camera frame.
- Z position, representing static positioning pointing with the camera, positive direction from the camera sensor to the camera lens.
- Yaw orientation for static orientation around the Y-axis, following the right-hand rule.
- Roll orientation for static orientation around the Z-axis, following the right-hand rule.
- Pitch orientation for static orientation around the X-axis, following the right-hand rule.

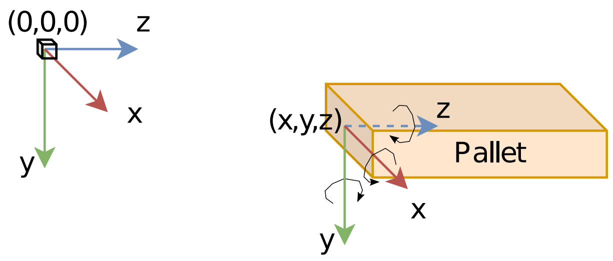


Figure 1.2: 6D of freedom related to the pallet's pose. The rotation is pitch, yaw, and roll for rotation around X, Y, and Z, respectively. Following the right hand rule.

To achieve the pose estimation, several secondary objectives are also underscored. The objectives include development steps and components needed to generate the pose and enable precise and automated estimation. The objectives include the development of an object detection algorithm, camera-LiDAR fusion, and point cloud processing to localize the pallet in 3D. These parts are focus areas for the project as well.

1.2.3 Research Questions

The following research questions are raised to further study object detection and pose estimation in industrial environments.

- Given the object in 3D space, how can we determine its position and orientation (pose) by a unique co-ordinate such that the forklift can maneuver to its location and lift it correctly?

- While machine learning (ML) model can detect the object in the scene; however, it gives the information in only 2D. How can we use additional sensors together with the camera to get additional information in 3D along with its orientation?

- On one hand, ML models trained on point cloud can detect the object in 3D space, but they are computationally expensive. Exploiting individual capabilities of the camera and LiDAR, can they be used together to detect the object in a scene and at the same time determine the orientation as well?

1.2.4 Delimitations

It is essential to underscore that the project is limited to the first part of the lifting and handling operation refereed to in Section 1.2, which is to detect the pallet and estimate the pose of the object. The project does not include communicating the information to the forklift's Human Machines Interface (HMI) nor controlling the forklift for positioning to lift the pallet. It is also assumed that the forklift knows which pallet to select, as this usually is communicated through the Terminal Operating System (TOS), which means that it would only be needed to calculate the pose of the single specific pallet for the given operation.

Another important assumption defined for this project is the roll orientation of the pallet. As the pallet is laying on a defined flat surface, there will be no movement around the Z-axis as seen in Figure 1.2, and consequently no roll movement. This implies that the roll value is not considered when testing and evaluating the developed design, as the value is expected to be constant for all tests.

1.3 Research Method

To achieve the project goal, a variety of different methods have been applied. An extensive, state-of-the-art survey has been performed to understand the fundamentals of object detection and pose estimation. Some of the findings are presented in Section 2.3. The state-of-the-art research identified relevant object detection algorithms and pose estimation models. The solutions were then evaluated based on their relevance to this project. Accordingly, the algorithm and models found the most suitable were selected and integrated with modifications to achieve the project goals.

With learning from the state-of-the-art study, and the evaluation of object detection algorithms and pose estimation models, a systematic solution has been designed. The developed solution has then been tested to evaluate the solution's performance and validate the results. The solution has also been applied across several scenarios to evaluate the performance in different contexts and conditions. From the experiments and tests, both numerical and visual results have been generated, which are evaluated and commented on for potential further improvements. The results conclude the developed model's performance in detecting the 6D pose of pallets.

1.4 Report Outline

The report is based around the project's two key components, i.e., object detection and pose estimation, and is consequently built up of four main parts: system design, implementations, testing and experimenting, and discussion of results. The parts are building on each other, where the system design, including components, is introduced first based on a real-world use case, followed by a walk-through of the implementations performed to achieve the design. After that, the implemented design is applied in three different test scenarios where graphs and key parameters are generated to evaluate the design's performance. Finally, the results are discussed to identify both strengths and limitations. The following list is an overview of the report's chapters in chronological order.

- Chapter 2 presents the enabling technologies and background theory that establish the foundation to achieve the project's design.

- Chapter 3 introduces the real-world use case that the project is based on, followed by an overview of the system design and components selected to achieve the design.

- Chapter 4 goes through the performed implementations to achieve the object detection part of the project. The implementations are based on components selected for the system design and result in a complete object detection algorithm.

- Chapter 5 uses the output from the object detection as a base and introduces the implementations to achieve the pose estimation part of the project. With input from the object detection, the pose estimation implementations realize the project's end-to-end system design.

- Chapter 6 highlights two test scenarios to evaluate the developed design in different contexts. The chapter presents both the test scenarios and their purposes, followed by the results generated from each scenario.

- Chapter 7 uses the results from Chapter 6 and experiences from the implementations in Chapters 4 and 5 to discuss some of the results and findings from the project work. The chapter also presents two additional developments that have been tested but not integrated into the system design.

- Chapter 8 presents the conclusions drawn from the project's implementations and tests, highlights the authors' contributions, and identifies a few directions for future work.

2. Background and Enabling Technologies

The purpose of the chapter is to introduce technological advancements that are key in developing this project. The chapter presents some of the background theories related to the topic of object detection and pose estimation. It also introduces some of the key technologies that enable the project's objective, followed by state-of-the-art research within the academic field of object detection and pose estimation in industrial environments.

2.1 Enabling Technologies

To achieve autonomous object detection and pose estimation, a few key technologies are highlighted. The rapid advancements in the field of these technologies have enabled the opportunity to map and locate objects in 3D and generate a digital representation of the sensed surroundings. The image data and digital representation algorithms and ML methods are capable of analyzing and categorizing the input, which makes the system capable of solving complex tasks. The following subsections are introductions to three different enabling technologies introduced.

2.1.1 3D Sensing Hardware and Point Clouds

One of the key enablers behind autonomous location and mapping is 3D sensing hardware. Robots can autonomously navigate based on the sensor data with accurate technologies that can generate a digital map of the physical environments. In 3D sensing hardware, multiple techniques are used to generate depth perception for the images' pixel values. Some of the solutions include sensors such as stereo cameras, Light Detection and Ranging (LiDAR), Radio Detection and Ranging (Radar), infrared sensing, and Red Green Blue Depth (RGB-D) cameras, which all are capable to generate a depth perception of the surroundings [2]. As a result of the generated depth, point clouds of the sensed environments can be created and analyzed. The point cloud represents the structure of the data in 3D. The point cloud includes depth coordinates for each pixel in an image, which means that it is possible to gather a 3D understanding of objects from 2D images [3].

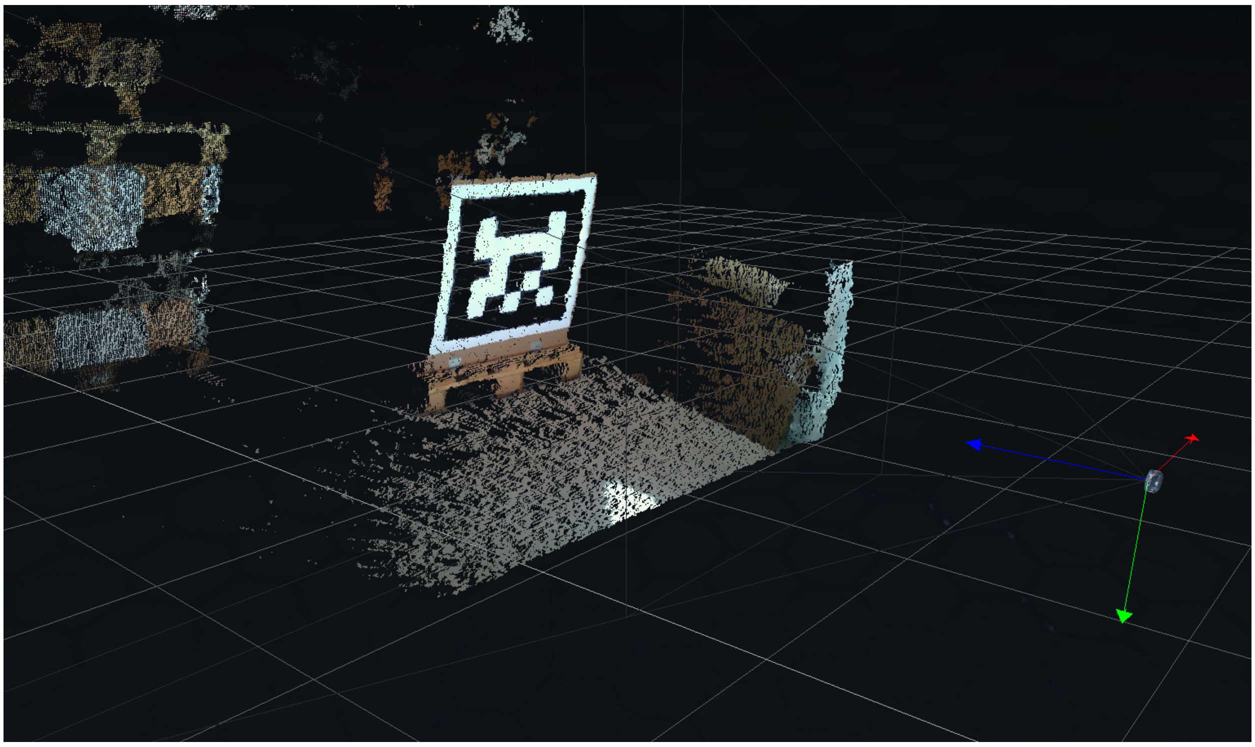


Figure 2.1: Display dump of a pointcloud from the Intel Realsense Viewer v2.50.0 [4].

Advanced point cloud processing development is key to analyzing and creating value for the generated 3D perception. The development of processing tools has improved rapidly in the past years, primarily due to open source software like Point Cloud Library (PCL), further described later in this chapter. From the step of transforming a 2D image to a 3D projection, complex geometric and mathematical operations can be performed. Through these algorithms, a perception of the environment's objects and their pose can be calculated, generating a comprehensive understanding of the characteristics of different surroundings. Figure 2.1 shows a display dump of a point cloud from the Intel Realsense Viewer software.

2.1.2 Machine Learning

Over the past decade, one of the key influencers within data science is ML. By developing models and algorithms capable of identifying and learning complex feature-based patterns and relations in large data sets, ML has proven its potential to solve complex problems through classification and regression from the detected relations. ML is a subcategory of Artificial Intelligence (AI), seen in Figure 2.2. While AI includes all sorts of methods that indicate intelligence, ML goes into programs and algorithms capable of learning features based on data without a defined model. Deep Learning, as also seen in Figure 2.2 is a particular part of ML that uses deep neural networks to solve complex challenges based on large data sets.

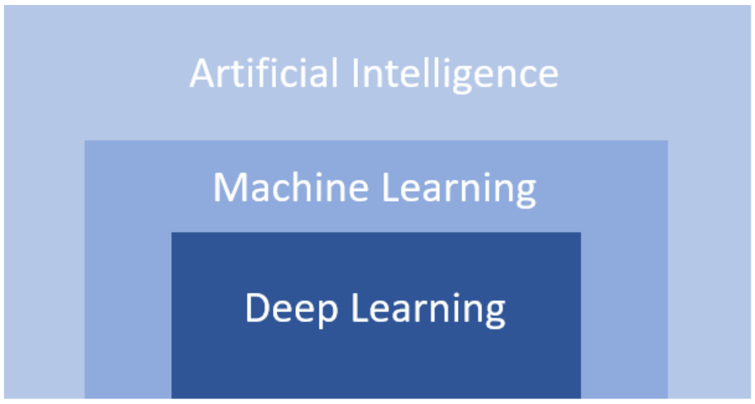


Figure 2.2: Overview of the relation between the different Artificial Intelligence categories.

Typically ML are separated into three different methods based on how they learn from input data. The base of all of them is that they are supplied with data as a feature vector (x ϵ Rm), which is further used to make predictions, f(x) = y ϵ Rv. The final goal behind the three techniques is to create a model that can make predictions as close to the ground truth as possible. In the following list is a short introduction to the three methods and their method for learning [5, 6].

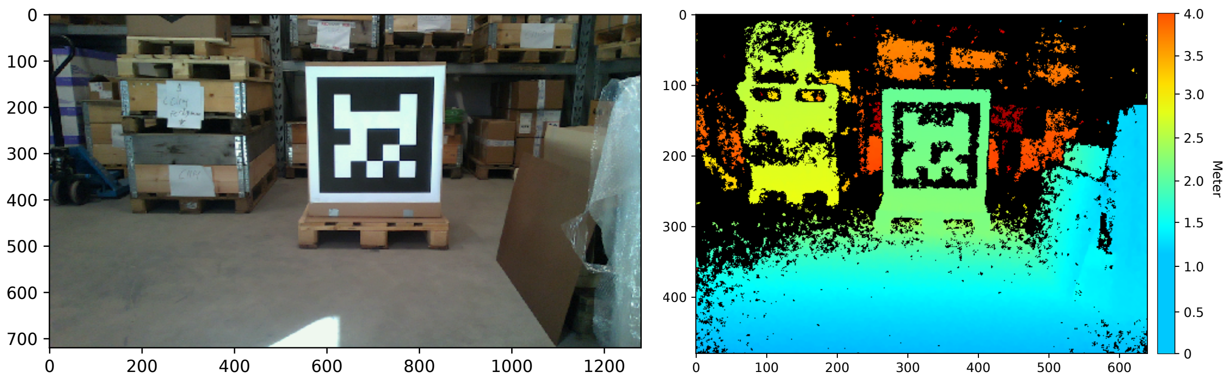
- Supervised learning inputs the true value for the data set's final output and used that to detect patterns and learn the correct combinations in the data. The performance of supervised learning is calculated through the loss function that evaluates the algorithm's final prediction against ground truth.
- Unsupervised learning usually uses clustering or statics to make the final prediction. The loss function is usually derived from inconsistencies between the relationship of different points in an x-dimensional space depending on the number of inputs.
- Reinforcement learning is one of the state of the art techniques, where human input and statics are substituted with machine input, which means that the algorithms are trained by set rules executed by agents, which are lines of code.

2.2 Building Blocks

To achieve the project's goal, several technologies and tools are utilized. Many of them are based on some of the state-of-the-art solutions within sensing hardware, ML, and embedded software. In the following subsections are the building blocks used to achieve the project's solutions introduced.

2.2.1 RGB-D Camera

For mapping and monitoring surroundings in 3D, RGB-D camera is a popular choice. The camera consists of a monoscopic camera and an Infrared (IR) sensor that create a color-coded depth image together. The camera is typically capable of operating in the range between 0.3 meters to 10 meters on 30 Frames Per Second (FPS) rate with resolution of 1280x800 pixels [7]. A point cloud is generated from the IR sensor that uses ToF principles, which is a key part of the project's system design. An illustration of the color-coded depth image is seen in Figure 2.3. One of the popular applications of RGB-D cameras is in Simultaneous Localization and Mapping (SLAM) systems, where both a visual understanding of nearby elements as well as further distances are needed [8]. A common application of RGB-D camera is seen in [9], where it is used for mapping unknown terrains by making a perception of the environment.



a) RGB image.

b) Perceived depth image.

Figure 2.3: An RGBD image acquired with Intel Realsense l515.

2.2.2 Open Source Software

The progression in the development of computer vision and ML has grown rapidly over the past years. One of the reasons for the fast growth is open source communities and ML frameworks, where people can share and contribute new code and improvements to existing code. The frameworks have established a high-quality foundation that is easily built on by external contributors and is available for however that want to take advantage of the optimized solutions. In the following paragraphs is a short introduction to some well-known communities and frameworks used for the report's project presented.

PCL

PCL is an open-source community with algorithms for point cloud processing and 2D/3D image geometry processing [10]. The library has state-of-the-art solutions for filtering, feature estimation, surface reconstruction, registration, model fitting, and segmentation [2].

OpenCV

OpenCV is an open-source library for computer vision. The community is one of the drivers behind the rapid development in computer vision applications, where contributors worldwide collaborate to achieve state-of-the-art solutions to complex problems. The scripts available at OpenCV give access to a large variety of codes to implement real-time computer vision and are used as the base to realize the report's design [11].

Some of the topics where OpenCV has proven to be a great tool are image filtering, histogram generation, machine learning based on pixel patterns, and object detection [12]. These are all relevant topics for the project's design.

ML Frameworks

In the world of AI, multiple open-source libraries and frameworks with code for efficient implementation of ML applications exist. Frameworks such as TensorFlow, Pytorch, and Keras offer the opportunity for iterative research and exploring of algorithms in new approaches and verticals.

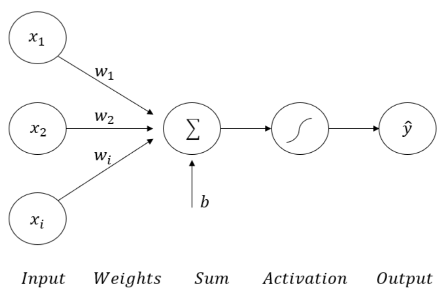
2.2.3 Deep Neural Networks

Deep neural network is one of the most common architectures for feature mapping in ML applications. It is used in many different applications for classification, prediction, filtering, and pattern recognition from large data sets. Neural networks consist of multiple layers, seen in Figure 2.4b), to achieve complex tasks using simpler mathematical functions. The first layer of a neural network is the input layer and is where the feature vector is inserted into the network. The data is accordingly weighted based on its importance and forwarded to the network's artificial neurons that input data and make simple calculations to generate an output. An illustration of the artificial neuron is seen in Figure 2.4a). The artificial neuron follows three simple steps: multiplication, summation, and activation [13]. Based on the summarized calculations of the inputted data and a considered bias, the result is run through an activation function that decides on the output. The formulated function of the operations is seen in Equation (2.1).

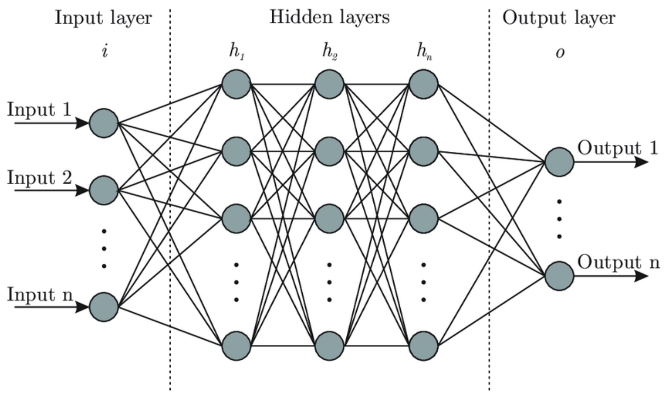
$$\hat{y} = F\left(\sum (x_i * w_i) + b\right) \tag{2.1}$$

Table 2.1: Description of Equation (2.1).

Symbol	Description
F	Activation function
x	Input data
w	Weights
b	Bias
i	Number of input nodes
$\hat{y}$	Output



(a) Forward propagating artificial neuron.



(b) Deep Neural Networks [14].

Figure 2.4: Deep Neural Networks Overview.

Between the network's input and output layers are usually multiple hidden layers. The hidden layers consist of the same artificial neurons and perform similar operations. However, different from the input and output layers are the hidden layers not interfered with, where the input layers feed the data while the output layers deliver the results. At the same time, is the network's intelligence usually found in the hidden layers where many trained weights influence the transported data through the network. The network's hidden layers **h** are seen in Figure 2.4b), the number of hidden layers defines the network's depth. The final layer is the output layer that makes the final prediction based on the information transported through the network [6].

For fully connected deep neural networks, some important expressions must be considered for the report's project. They are explained in the following list.

- Activation functions decide the neurons' output through a defined function like the step function and are the part that includes non-linearity in neural networks.
- Loss function is the parameter that enables training of neural networks through gradient descent; reaching the global minimum of multiple parameters is the end goal. Steps for generating the loss function are as follows: random initial weights are selected, input values are looped through the network until they converge, the gradient is computed, and weights are updated and returned to the network.
- Learning rate controls how much change is applied to the model as the weights are updated, where the final goal is to reach the global optimum. The parameter is also essential in avoiding local minimums that can result in underfitting and overfitting [15].

2.2.4 Convolutional Neural Network

A key neural network in computer vision is Convolutional Neural Network (CNN). The method uses convolutions that are linear mathematical operations between matrixes to process large amounts of image data and extract features from the data sets. The network has shown remarkable results in object classification, detection, and semantic segmentation [16]. CNN got multiple layers, which include the convolutional layer, non-linearity activation layer, pooling layer and fully-connected neural network [17]. CNN for object detection are typically separated into two parts, feature learning and classification. A short introduction to the two parts is given in the following paragraphs.

For the first part, feature learning a filter of weights, defined by the desired feature detection task and usually in the format of (3x3) or (5x5) patch size, called a kernel. The kernel is slid over the input data creating a new image that is reduced in size, but where neighbor relations and local features are embedded into the output. The final result of the process is decided by the summarized output of the slid window added with a bias and then ran through a non-linear activation function called ReLU. The procedure is called a convolution to create a feature map from relations in the data. Before the data is inserted into the neural network, a couple more operations are performed. The first one is pooling, where the purpose is to reduce dimensionality and remove spatial invariance. The operation is performed by sliding a similar filter over the input data, typically (2x2), and summarizing the number in each patch. This reduces the total size of the input while maintaining information and spatial structure. The next step for the object detection CNN is to flatten the output of the pooling such that the data is represented in a 1-dimensional matrix that can be inserted into the neural network. A complete overview of the end-to-end CNN is seen in Figure 2.5.

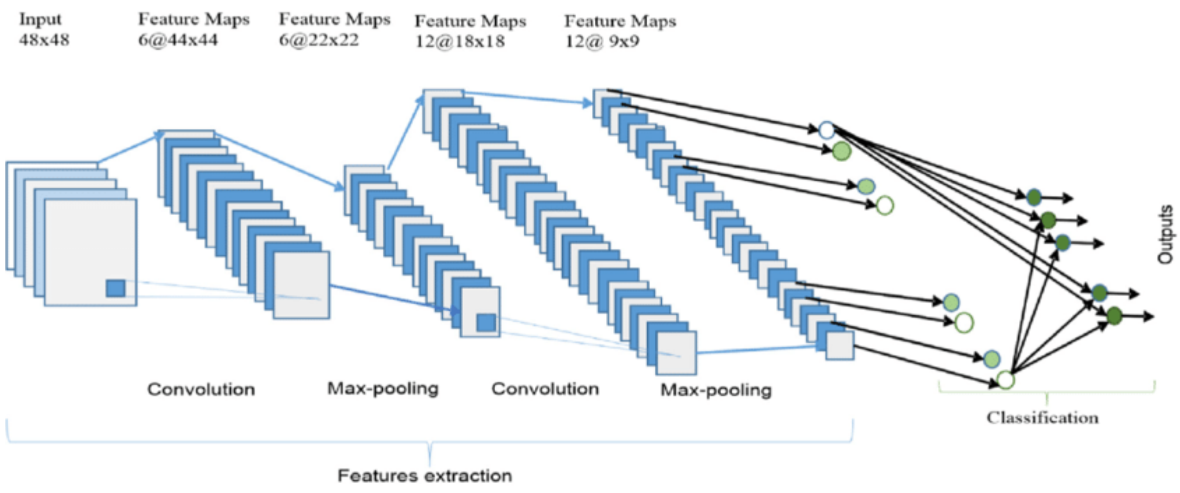
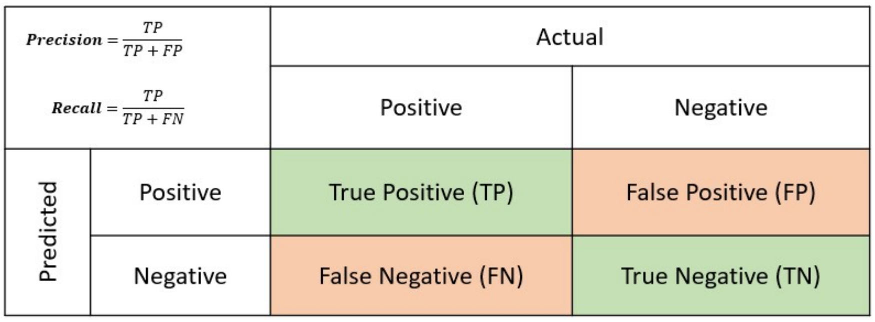


Figure 2.5: CNN [18].

2.2.5 Object Detection Metrics

Two highlighted key parameters evaluate the ML object detection models. They are Throughput measured in (FPS) and Average Precision (AP) measured in (%). The first parameter is Throughput, representing the number of processed frames or images per second. The higher the number, the faster the processing algorithm can process the input data, which is closer to real-time. The parameter is also highly related to the hardware the ML algorithm is executed on, which needs to be taken into consideration when evaluating the models based on their FPS performance [19].

AP is a more complex calculated parameter from the algorithm's precision and recall, shown in Figure 2.6. Four parameters represent AP: true positive and true negative, when the algorithm detects the correct either positive or negative truth, and false positive and false negative, when the algorithm predicts the opposite of the actual truth.



$Precision = \frac{TP}{TP+FP}$ $Recall = \frac{TP}{TP+FN}$		Actual	
		Positive	Negative
Predicted	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

Figure 2.6: Predicted vs. actual result.

As shown from the formulas in Figure 2.6, precision measures all correct predictions as a percentage score of all predictions. On the other hand, recall measures the correct positive predictions as a percentage score of all actual positive cases [21]. It is important to note that it is a trade-off between the two values [22] depending on the quality of labeling and ground truth. Together, the two measurements form the Precision-Recall (PR) curve, which shows the relation between the two measurements, seen in Figure 2.7. From the found PR curve the AP is the area under the curve. If a certain threshold value for the AP is desired, it is implemented by setting a cutoff score, where all values below this score are set to false positive.

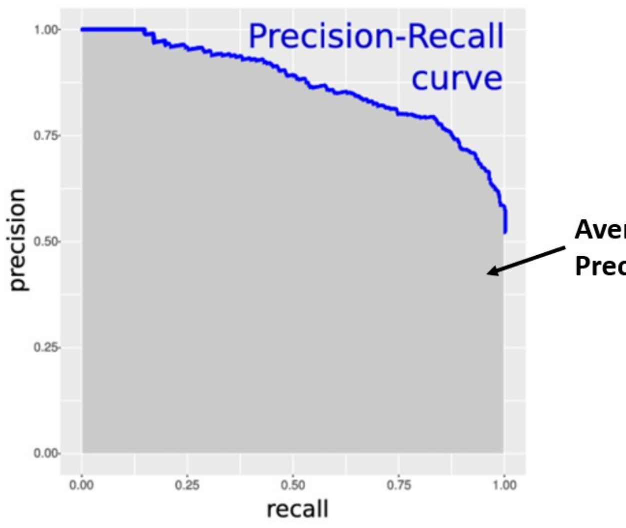


Figure 2.7: Precision-Recall curve [20].

Another important evaluation metric related to AP is Intersection over Union (IoU), which measures the overlap of an object detection bounding boxes relative to ground truth, seen in Figure 2.8. Further descriptions of bounding box methods are presented in Section 4.2. Based on the IoU value calculated from Equation (2.2), it is possible to evaluate how well the algorithm is preforming, if the bounding boxes perfectly fits ground truth IoU = 1, if it does not match at all IoU = 0 [23]. To find the AP the first step is to define a desired IoU value e.g. 0.5(50%) which means that all value above 0.5 are true positives. The set value can be changed and should be tested for multiple to avoid biases. A typical scenario is to test AP for AP25 (25%), AP50 (50%), and AP75 (75%).

$$IoU = \frac{\text{Area of intersection}}{\text{Area of union}} \tag{2.2}$$

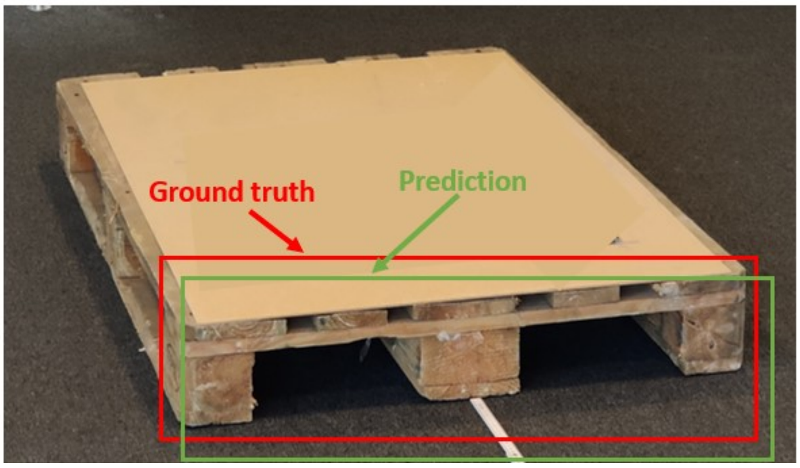


Figure 2.8: IoU illustration, which represents the area where the green and red box overlaps. Depending on the percentage of overlapping, different AP scores are defined such as AP50 (50% overlap) and AP75 (75% overlap).

2.2.6 Large Data Sets and Data Labeling Tools

A key to enabling high performing ML model is large data sets with images of the desired object from multiple distances and angles as well as locations and external influences such as light. Due to the significance of sufficient data, many acknowledged technology companies such as Facebook and Microsoft have published and openly shared very large sets of common objects in different contexts. One example of this is Microsoft's Common Objects in Context (COCO) which is one of the most used data sets for the initial training of neural networks. The dataset consists of complex everyday scenes with familiar and simple objects in natural contexts [24]. It consists of 200,000 images with more than 40,000 of them annotated. Out of the images, 80 object categories are labeled by bounding boxes and per-instance segmentation masks to achieve precision. An overview of some of the objects included in the data set is seen in Figure 2.9.

With large data sets at hand, precise annotation is needed. From the presented COCO data sets, a large number of the targeted objects must be labeled giving the algorithms ground truth values. Based on the ML algorithm's prediction the deviation in result from the ground truth values is used to generate a loss function, which is the basis of learning machines. The annotation can be performed manually by marking the desired object that is desired to detect or semi-automatic, where algorithms are used to make labels, where the predictions are manually overlooked afterward. The outcome of the labeling is a file with values showing the labeled data.

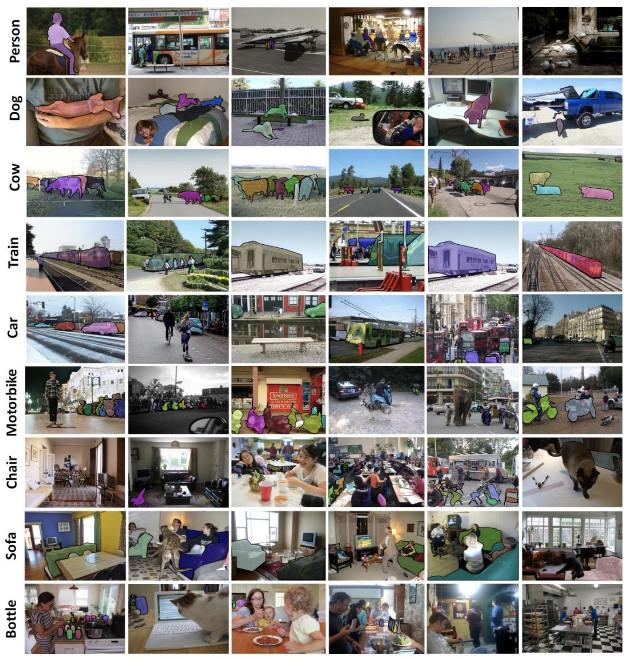


Figure 2.9: COCO Items [24].

2.2.7 Fiducial Markers

To ensure accurate & robust localization and 6D estimation of objects fiducial markers are common tools. The markers consist of a fixed square border with a coding/ decoding system with different IDs. The marker IDs use a different composition of bits in various positions that specify the different types of markers [25, 26]. To read and interpret the IDs computer vision algorithms are implemented specialized to detect the position of the marker and pose based on the structure and composition of bits with the fixed square birder as reference. Together, the solution enables the detection of the position and pose of each given marker, as long as the marker is sufficiently visible. There are multiple different types of fiducial markers, some of the more used once for accurate localization are ARToolkitPlus, AprilTag, and Aruco, they are seen in Figure 2.10. Due to the markers' capabilities, they have also become popular in providing ground truth with reference points and measurements, where the performance of object detection solutions is evaluated against the results of the fiducial markers.

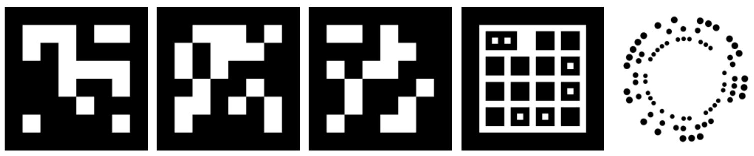


Figure 2.10: Fiducial markers from left to right, ARToolkitPlus, ArucoTag, AprilTag, TopoTag, and RuneTag [27].

2.3 Related State-of-the-Art Work

In the domain of object detection and pose estimation, a number of state-of-the-art articles are published. This includes both high-performing algorithms for object detection and different techniques for pose estimation. When it comes to industrial applications of the two technologies, state-of-the-art work is more narrow, but still, a few articles and papers have tested solutions for the lifting and handling of pallets. In the following subsections are some of the state-of-the-art work on the topic of object detection and pose estimation presented.

2.3.1 Object Detection

In [28], the journey towards real-time object localization and classification was introduced using CNN in combination with region proposal techniques to increase the speed of data processing, while still maintaining precision. Multiple of the best-performing object detection solutions used today are based on principles from [28].

From [29], a new major leap towards real-time object detection with high precision was introduced. By learning from [28], the paper introduced a one-stepped approach for both object localization and classification, which substantially improved algorithm efficiency. In whats referred to as real-time object detection, [29] is one of the pioneers.

The paper, [30] presents an application of CNN based object detection in logistics warehouses. To achieve the application a large data sets of pallets in relevant contexts was utilized, accordingly, a single shot multibox detector ML algorithm was trained on the data set and applied for testing.

2.3.2 Pose Estimation

One of the papers proposing a model for identifying pallet holes is presented in [31], where they use a RGB-D camera and semantic segmentation to identify the pallet and its holes. From the RGB-D camera a points cloud is generated, which is used to get depth and orientation understanding of the desired object.

In [32] they are using a deep CNN to recognise pallets and localise the pallet by a selected bounding box. Further, [32] uses a RGB-D camera to generate depth data to identify the pallets center coordinate and in what direction the pallet is standing.

Some of the solutions that offer high precision pose estimation is presented are [26], and takes advantage of physical markers, called Fiducial markers, with patterns and ids to estimated the pose. As the markers requires physical adaptions to be applied in real-life cases, where they need to be mounted to the desired object, they are typically used in purposes where the environment is know and closed of. For this reason, using the Fiducial markers presented in [26] as the ground truth value is a good opportunity.

3. System Design

The chapter introduces a real-world use case, which explains the process that this project is based on. From the real-world use case and other technical specifications defined together with the industrial partner a number of requirements and expectations are set. They are related to performance requirements, testing, and evaluation. Based on the real-world use case and requirements a system design is presented. The design shows the end-to-end pipeline for the solution developed for this project. The design is built on a number of components, which are introduced as well. Together, the chapter gives an overview of the key aspects of achieving the desired object detection and pose estimation solution.

3.1 Real-World Use Case Identification

The project is based on a real-world use case found in the industrial partner's R&D portfolio. An illustration of the project is seen in Figure 3.1 and is based on a forklift operation where the vehicle has received an order to pick a specific pallet and is supposed to execute the task autonomously. From the real-world use case, the forklift approaches the pallet within a range of 0° to ±45°, as this is required to keep the forks clear of the pallet. At this stage, where the forklift will start positioning the forks to lift the pallet the distance from the pallet is approximately 2 meters, as also seen in Figure 3.1. The task requires multiple steps as the forklift needs to navigate autonomously to the given location, then it must identify and classify the identified object to be sure it is the right object class before it needs to estimate the pose of the object to correctly position itself to be able to lift and handle the pallet. The final step of the autonomous operation is to communicate the pallet's pose to the forklift's HMI to send commands to the electrical drives. The total process enables the forklift to complete the process without human involvement, which increases operational safety and efficiency as the solution is able to work both day and night.

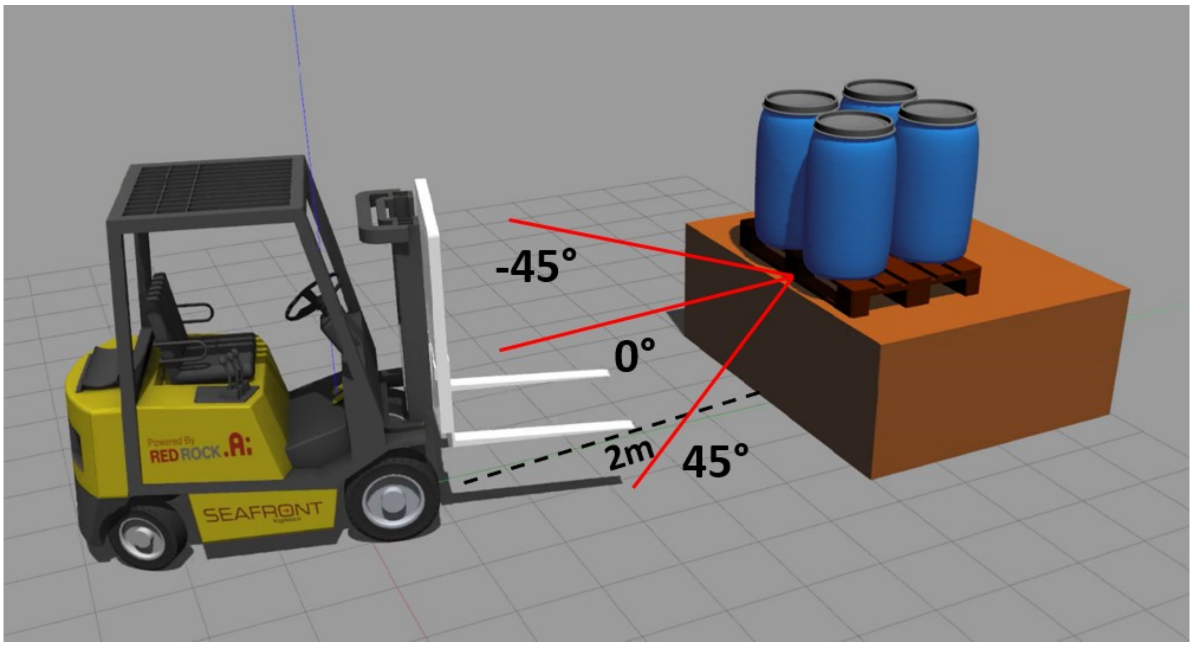


Figure 3.1: Autonomous forklift simulation.

The previous paragraph explained the complete process to enable autonomous operations. This project is more specifically limited to focusing on two of the parts required to enable the complete operation, as well as the other limitations presented in Subsection 1.2.4. The two highlighted phases are:

- **Object detection**, where the sensors on the forklift need to recognize and classify the object or more specifically pallet from the defined range, 0° to ±45° and around 2 meters.
- **Pose estimation**, where the detected object x, y, and z coordinates followed by pitch, roll, yaw orientation are calculated and displayed, with respect to the origin of the camera frame.

The following section will go into details of requirements and expectations related to the needs and limitations highlighted by the industrial partner. The following chapters will dig into the implementations of the two key parts, with technical specifications and development steps to achieve a design that fulfills the desired objective seen in the real-life use case.

3.2 Requirements and Expectations

From the collaboration with the industrial partner Red Rock, a set of requirements and expectations are defined. They are based on needed adaptions to enable the design for operation in industrial environments with defined performance expectations.

3.2.1 Requirements

The project is developed based on a number of requirements. The requirements are defined based on the specification needed to implement the solution in real-life operation and to achieve the desired system design performance. As the project is consisting of two main parts the requirements are separated into sections for each of them, followed by one section for the performance evaluation. An introduction to the key tasks for each of the parts is seen in the list below, and closer descriptions with detailed requirements for each part are presented in the following subsections.

- Object detection: develop an object detection algorithm with a proper trade-off between Throughput measured in FPS and AP, and that is capable of finding the x and y coordinates of pallets.
- Pose estimation: estimate the pallet's pose and generate results for both a position (x,y,z) and orientation (pitch, roll, jaw). As the pallet is expected to be laying on a flat surface the roll is assumed to be constant at all times and does not need to be included.
- Ground truth: develop a model to estimate ground truth in a dynamic environment.

Object detection requirements

For the object detection algorithm, the most important requirement is that the algorithm is capable of delivering results in close to real-time, which in this context means that it needs to process the data faster than the sampling frequency of the implemented sensor. The sensor selected for this project is capable of sampling at 30 FPS. A second important requirement is that the algorithm must deliver an AP, which is capable of delivering an accurate bounding box around the pallet. Accordingly, the following requirements are defined.

Table 3.1: Overview of objected detection requirements, where the throughput represents the processing capacity of the solution measured in FPS.

Parameter	**Value**
Throughput	$\geq$30 FPS
AP	$\geq$40%

Pose estimation requirements

For pose estimation, the requirements are related to tests and evaluations of the developed solution's performance under standstill tests. As a base, it is required to test the developed solution relative to ground truth and perform a statistical analysis of the results, finding the precision σ, mean value μ, and accuracy by measuring the error between estimated and ground truth. As the range for the forklift when approaching the pallet is between 0° to ±45° and 2 meters, as described in Section 3.1, evaluations are only required to be performed in this range. When testing the algorithm from 0° to ±45°, the results from both the positive and negative angles are assumed to be similar, such that it is only required to test the system design from one of the (±) angles. For the defined range between 0° and 45°, it is expected that the accuracy μ is within certain values, shown in Table 3.2. An overview of the defined testing limitations and performance parameters are also shown in Table 3.2.

Table 3.2: Overview of required test and performance parameters.

Parameter	**Range**
Distance in meter	2 m
Angle in degrees	0° to ±45°
Accuracy μ in millimeters	< 30 mm
Accuracy μ in degrees	< 8°

Ground truth requirements

To evaluate the performance of the pose estimation algorithm a ground truth method is required. As the algorithm should be capable of being tested in arbitrary locations and environments, the ground truth must be flexible such that it can easily be moved around without influencing the performance.

3.2.2 Expectations

The industrial partner has a few expectations for the final result as well. One of the central targets is that the object detection and pose estimation model should be able to run on constrained devices such as System-on-Module (SoM) and Single-Board Computers (SBC) machines, which means that the developed solution should be converted to a file format that achieves maximum throughput for constrained devices. Another expectation for the final result is that the algorithm should be tested in an industrial environment to evaluate performance and to identify potential obstacles for the solution to be applied in real-life. It is also desired to test the pose estimation algorithm beyond the defined requirement of 45° and evaluate how far it can go before it reaches its limit. In the following list are the project's expectations listed.

- Test the pose estimation algorithm in standstill condition beyond 45° to evaluate the capability of the solution.
- Test the algorithm in moving condition from 0° to 90°, with focus on the results between 0° and 45°. Additionally, evaluate the capabilities and limitation of the algorithm in moving condition beyond 45°.
- Optimize the software to be run on constrained devices.
- Test solution up to 3 meters distance and evaluate the results.
- Test solution in relevant real-life environment.

3.3 System Design

The solution created for the project consists of two major tasks, object detection and pose estimation. The two parts are accordingly integrated into a single model for implementation in an embedded application. An overview of the design, as well as the flow of the solution, is presented in the following subsections.

3.3.1 Design Overview

Figure 3.2 shows an overview of the design, it consists of four key phases, they are training of object detection model, execution of object detection model, pose estimation, and performance evaluation of the pose estimation result. The following paragraphs give an introduction to the key phases needed to achieve the project's goal.

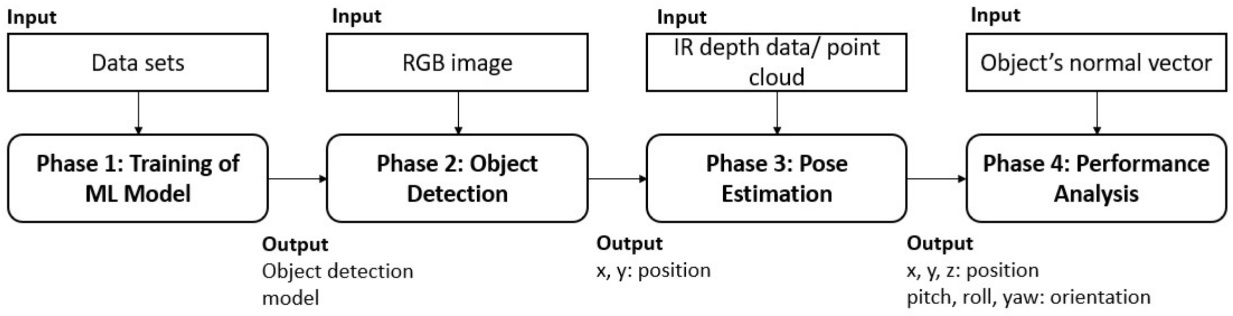


Figure 3.2: Overview of system design.

In the first phase, the object detection model is trained on an extensive data set of relevant objects in everyday environments. The algorithm is first pre-trained on the COCO data set typically used for pre-training before training on relevant objects for the project through the LOCO data set by using transfer learning principles. The process results in a specialized object detection model that is used for object detection in phase 2.

The second phase executes the specialized object detection model with raw input data from a warehouse sample. The algorithm further recognizes and classifies the identified object. The solution results in boxed objects including x and y positions of the objects followed by a confidence score representing the certainty of the classified object. The solution is capable of recognizing and classifying multiple objects simultaneously, although it is only needed to identify a single object to achieve the project's goal.

Based on the object detection model in phase 2, the pose estimation algorithm is executed. The pose estimation algorithms inputs images of the boxed objects and crop the full image based on the boxes to only include the contour and x, y position of the object for further processing. From the cropped image a 3D point cloud is filtered by the same sensor used to capture the image. The filtered point cloud represents a digital 3d point model of the object. Based on the point cloud's density in certain areas, the vertical plane of the object is estimated. Finally, from the corner of the box showing the pallet's position and the plane representing the pose of the pallet, a normal vector together with a 3D point is generated at the center front of the pallet. The normal vector and 3D point represents the 6D position of the object as desired, described in Section 1.2.

3.3.2 Step-by-Step Workflow

In more detail, Figure 3.3 shows the four phases of the design. The model is pre-trained on the COCO dataset for object detection. The labeled LOCO dataset is used as there are many images of labeled pallets. Transfer learning is done with the finished labeled LOCO dataset; thus, the model can detect pallets and pallet voids.

Phase two is applied inferencing with the trained model on custom input data of the warehouse; the inference is optimized with OpenVINO from Intel, improving the inference performance by reducing the inference latency, making the system capable of running on constrained devices.

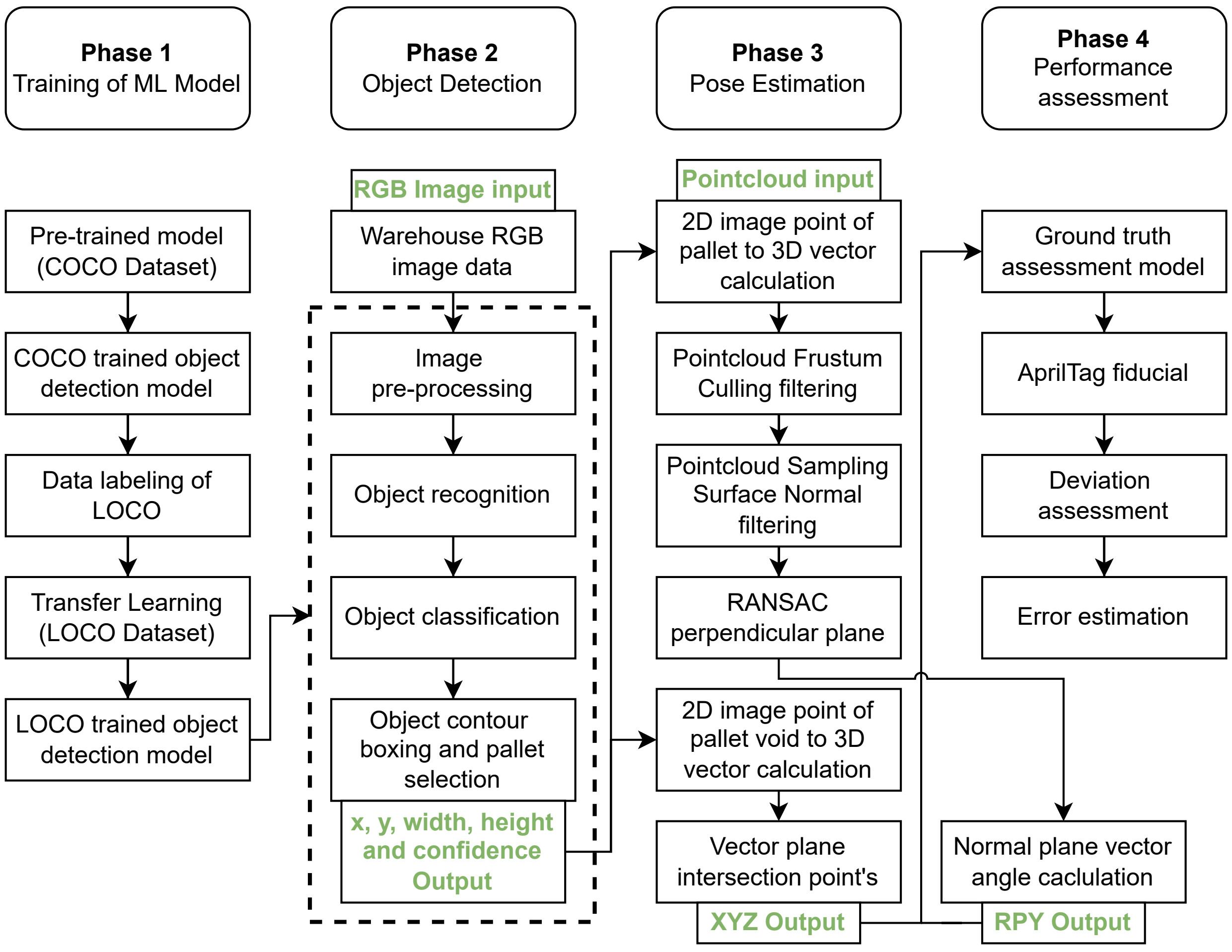

Phase 1
Training of ML Model
Phase 2
Object Detection
Phase 3
Pose Estimation
Phase 4
Performance assessment
Pre-trained model (COCO Dataset)
COCO trained object detection model
Data labeling of LOCO
Transfer Learning (LOCO Dataset)
LOCO trained object detection model
RGB Image input
Warehouse RGB image data
Image pre-processing
Object recognition
Object classification
Object contour boxing and pallet selection
x, y, width, height and confidence Output
Pointcloud input
2D image point of pallet to 3D vector calculation
Pointcloud Frustum Culling filtering
Pointcloud Sampling Surface Normal filtering
RANSAC perpendicular plane
2D image point of pallet void to 3D vector calculation
Vector plane intersection point's
XYZ Output
Ground truth assessment model
AprilTag fiducial
Deviation assessment
Error estimation
Normal plane vector angle caclulation
RPY Output

Figure 3.3: Step-by-step pipeline of system design.

The third phase is pose estimation, with the camera's point cloud as input and the object contour box output. By using the intrinsic camera parameters and the corner points of the bounding box from the object detection, are four 3D vectors calculated. Multiple filters are applied to filter out only the relevant points of the pallet. Frustum culling is a filter that filters the points within the bounding box, and sampling of surface normals is used to filter out the horizontal floor points. Perpendicular RANSAC is then used to create a plane of the pallet front, and the normal vector of the plane is used to calculate the orientation of the pallet and get pitch and yaw. A center vector from the four bounding box corners is derived to find the final 3D point of the pallet position. The 3D vector and plane intersection create the 3D point as the XYZ pallet position.

To evaluate the solution, phase four is performance analysis, where the algorithm is evaluated relative to the AprilTag fiducial marker that represents ground truth. The position of the marker in relation to the pallet is known and is set as a 3D vector constant prior. Creating an offset between the fiducial marker and pallet voids. The deviation between the fiducial marker and the system is calculated to evaluate the system.

3.4 Solution Components

To achieve the presented design several components are required, they are shown in Figure 3.4 related to the relevant phase of the design. The component list includes both hardware units, software solutions, and data sets. A key behind the design is the utilization of code available through large and open software communities, modified and integrated into the system architecture. A short introduction to each of the components used to realize the system design is explained in the following subsections.

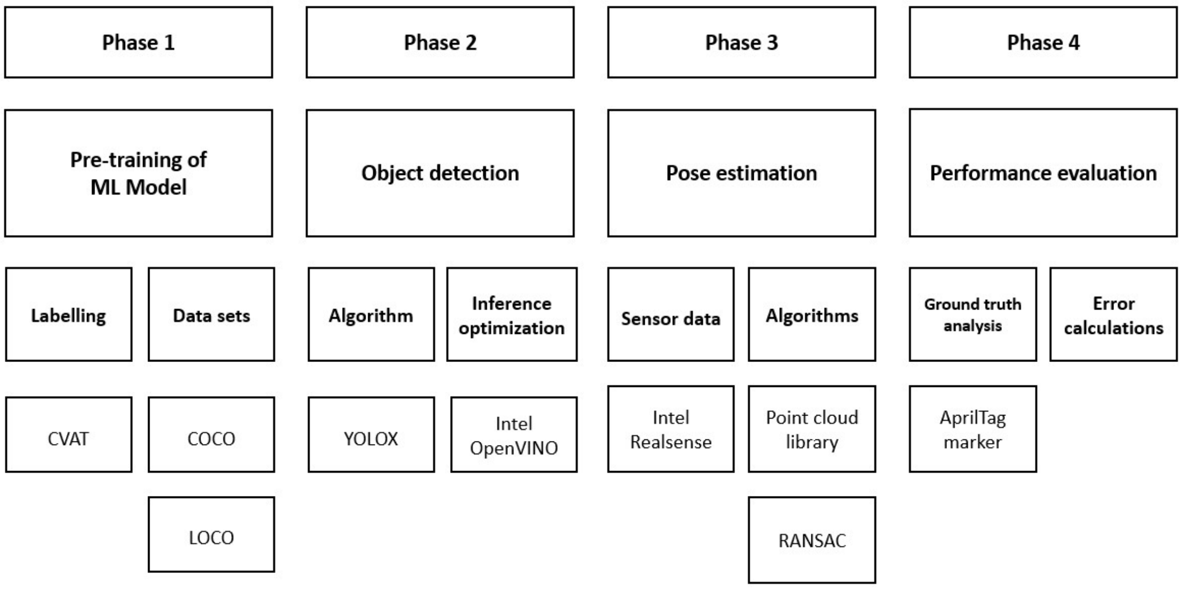


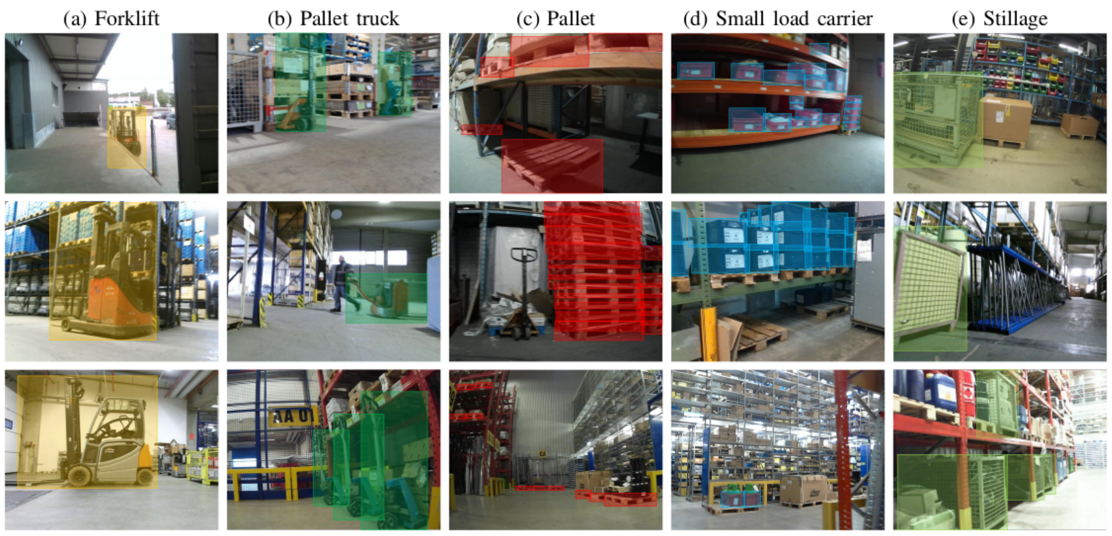
Figure 3.4: Overview of system design with components.

3.4.1 Intel Realsense RGB-D Camera

To generate the image for object detection as well as the point cloud to estimate the pose, Intel's Realsense RGB-D camera is used. The unit contains both an IR sensor that uses time of flight principles to generate a depth map of the surroundings, and a Red Green Blue (RGB) camera to create a colored representation of the environment. Together the Intel Realsense offers a robust solution to satisfy the requirements for the project's design [33].

3.4.2 LOCO Data Set

Training artificial deep neural networks require extensive amounts of data. To achieve this a comprehensive labeled data set is needed. LOCO is a scene understanding dataset for logistics, highlighting the detection of typical logistics-related objects. The annotated objects include forklifts, pallet trucks, pallets, small load carriers, and stillages. The dataset consists of 40,000 images captured in 5 logistics environments by five different low-cost cameras with different resolutions and Field of View (FoV). Out of the images over 5000 are manually annotated resulting in well over 150 000 annotations. A snippet of the data set and its different classes are seen in Figure 3.5 [34].



LOCO sample images of different object categories. For illustration purposes, only labels of specified classes are shown. Note the object difference within classes (e.g. pallet truck), image quality due to different cameras, and the contextual difference within different scenes.

Figure 3.5: LOCO Items [34].

3.4.3 Computer Vision Annotation Tool

To label and annotate large data sets, efficient labeling tools are needed. Intel's Computer Vision Annotation Tool (CVAT) is a web-based solution built to speed up the labeling process. CVAT enables efficient annotation for object detection, image classification and segmentation, performed by user-specified labels and attributes [35].

3.4.4 Point Cloud Library

From the PCL library presented in Subsection 2.2.2, two specific tools are used for the system design. The first one is the `pcl::FrustumCulling()` filter function which is used to filter out points related to a cameras orientation and FoV. The second one is `pcl::SamplingSurfaceNormal()` in which a `pcl::PointNormal` is calculated using a set N points of each grid. Reducing the number of points [10].

3.4.5 PyTorch

PyTorch is an open-source machine learning framework [36], that is created to accelerate the development process of ML projects and the time from code prototyping to production. From the PyTorch framework comprehensive ML algorithms can be imported, which accelerates the initial phase of implementing ML. The community around PyTorch is open, which means that algorithms are available for external contributions and improvements.

3.4.6 YOLOX

YOLOX is a new high-performance detector used for real-time object detection and classification. The detector is an updated version of YOLO, which was one of the first detectors to realize both object detection and classification in one stage. YOLOX offers a very fast solution compared to its alternatives due to its ability to deliver the end result in one stage. This is an important factor considering the time-related requirements for the project's design [29]. A closer introduction to the YOLOX architecture is presented in Chapter 4.

3.4.7 Intel OpenVINO

Optimizing and tuning neural network performance is considered challenging as many hyper parameters and metrics need to be adjusted to reach optimal performance [37]. OpenVINO is a deep learning toolkit by Intel that addresses optimizing issues. The toolkit is developed to optimize a neural network and inference performance for embedded devices [38]. It does so by enabling popular deep learning frameworks such as TensorFlow and PyTorch to be optimized to improve the inference latency. OpenVINO aims to accelerate application workloads by using a library of predetermined functions and preoptimized kernels [39]. For the project's design, OpenVINO is used post-training to accelerate inference performance.

3.4.8 AprilTag Marker

To establish a ground truth of the object's real-life pose, fiducial markers are proper tools as their estimations has proven to be very precise in closed of environments [26]. For this project, AprilTag markers are highlighted as a good choice. The solution is typically used for tasks such as camera calibration, robotics, and augmented reality, which is a good match for this project. The AprilTag consists of a square marker with a black border and an inner binary matrix that determines the attributes of the marker [40], an typical AprilTag is shown in Subsection 2.2.7. The detection procedures of AprilTags consist of multiple steps, including looking for linear segments, detecting squares, and based on the findings calculate the marker's position and orientation as well as decoding the barcode [40]. An essential element when utilizing AprilTag markers is to ensure that they are sufficiently mounted to the object, where deviation in position and uneven surfaces could result in undetectable errors.

3.4.9 RANSAC Algorithm

To estimate the planes of objects RANSAC is a preferred option. The method uses a random sampling technique on point clouds, which tries to fit a model robustly, y = f (x;a), to the point cloud data. The function is defined by certain parameters, such as finding the largest plane in the vertical direction, and by evaluating different functions and their inliers and outliers [41] related to the parameters the function that corresponds best to the data set's inliers is selected. From the regression of these points, a plane showing the function of the points is generated. A fitted function generated by RANSAC is seen in Figure 3.6.

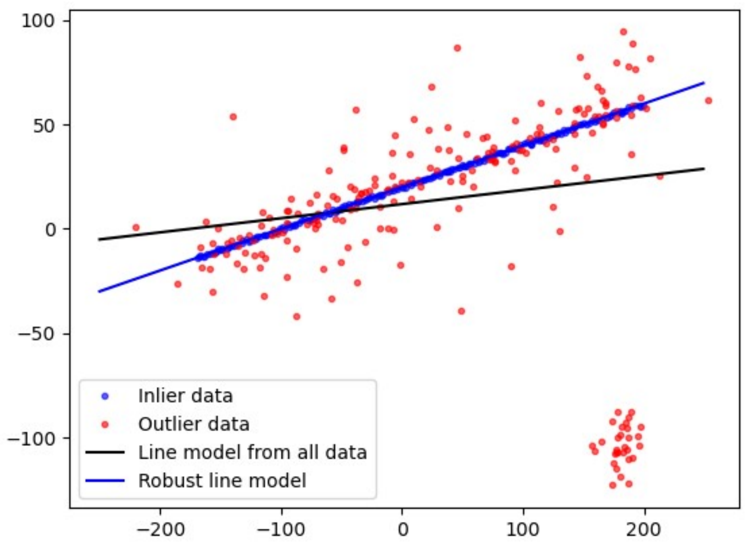


Figure 3.6: RANSAC display [42].

3.5 Chapter Summary

Chapter 3 has introduced the key development steps and enabling components to achieve the project, designed to be applied in the real-life use case. Based on the presented steps from system design, Chapters 4 and 5 will show the implementation of the introduced components to develop the solution.

First, in Chapter 4, the data processing and object detection algorithm are applied to detect the desired pallet. Second, in Chapter 5, the detected object is inputted followed by generation of depth sensor data from the Intel Realsense, which is further processed and analyzed using the presented point cloud processing tools with the goal of estimating the pose of the object.

4. Object Detection

The chapter presents three steps for developing and implementing the project's object detection algorithm. The chapter is based on the system design and components presented in Chapter 3. The first step is data processing, where a pre-trained network is used as a base followed by transfer learning measures to adapt the algorithm to detect pallets. Thereafter, an introduction to different object detection algorithms is presented, resulting in the selection of the algorithm most suitable for this project. The final step, is the preparation of the selected algorithm for industrial robust applications through inference optimizing tools. Together, the chapter explains the implementation of the selected object detection algorithm and the necessary adaptions to make it suited for detecting pallets in an industrial environment.

4.1 Data Processing

As the training of networks from scratch is costly in terms of time and computing power, utilizing pre-trained networks is beneficial. To develop and train the ML model a pre-trained generalized network is used as a base. The pre-trained model is trained on general objects, which means that specializing the network for detection of pallets is needed. To achieve this, a large data set from a logistics environment is labeled, annotated, and applied to the model using transfer learning principles. An overview of the total pipeline is seen in Figure 4.1. In the following subsections is an introduction to the data sets used to train the network, tools for large-scale data labeling, and steps for transfer learning presented.

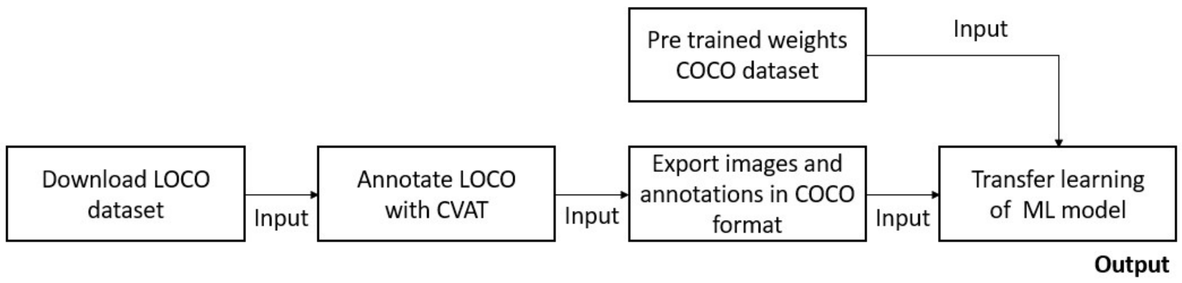

Pre trained weights COCO dataset
Input
Download LOCO dataset
Input
Annotate LOCO with CVAT
Input
Export images and annotations in COCO format
Input
Transfer learning of ML model
Output

Figure 4.1: Overview of data processing.

4.1.1 Pre-training of ML Model

For pre-training of the ML model Microsoft's COCO dataset is used, seen in Figure 2.9. The data set is one of the most used for the initial training of neural networks. The dataset consists of complex everyday scenes with familiar and simple objects in natural contexts [24], such as cars, trains, gods, etc. The COCO annotations are stored in a JSON file, including information about the bounding boxes, categories, images, and annotations. An overview of the COCO annotation is seen in Figure 4.2, where the id along parameters describing the category, segmentation, and bounding box are seen.

```
annotation{
    "id": int,
    "image_id": int,
    "category_id": int,
    "segmentation": RLE or [polygon],
    "area": float,
    "bbox": [x,y,width,height],
    "iscrowd": 0 or 1,
}
```

Figure 4.2: COCO annotation data format [43].

4.1.2 Data Labeling

The next step is to label data of pallets in different contexts to specialize the pre-trained model. To achieve good results, a substantial amount of high-quality labeled data, with pallets in different environments, from various angles and distances, varying backgrounds, and external conditions such as light and dust are needed. For this the LOCO dataset is implemented, previously introduced in Subsection 3.4.2. The LOCO dataset is imported by first uploading the images into each subset and then importing the annotations for each subset.

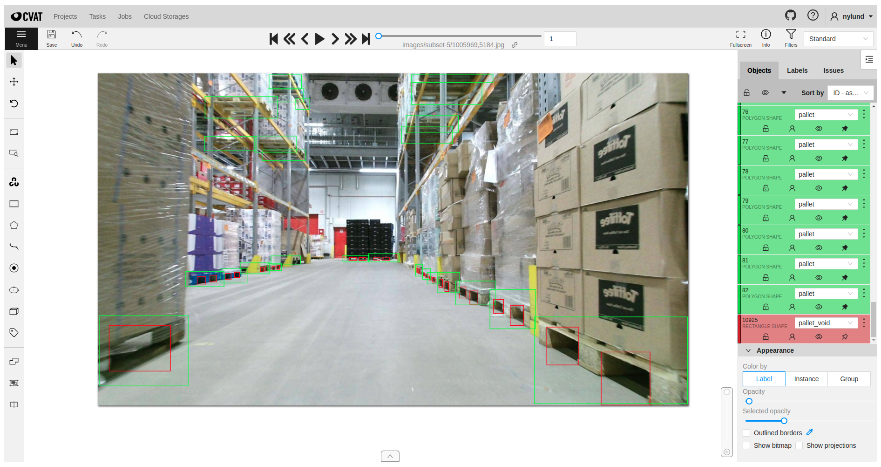
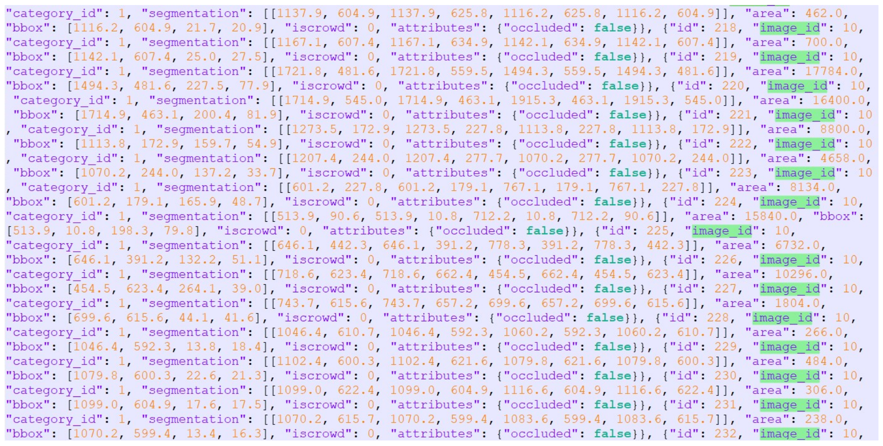


Figure 4.3: CVAT data labeling [44].

There are many ways of annotating data. Historically data annotation has been a manual process where humans have marked the desired region of interest, but as the amount of data has increased exponentially new tools to more efficiently label large data sets have been developed. Accordingly, the labeled data has been imported into Computer Vision Annotation Tool (CVAT), introduced in Subsection 3.4.3, that runs on a docker container locally on the computer. [45] The labeled data from the LOCO dataset has a total of five classes [34], where every class except pallet was removed from the dataset before training. An illustration of the data annotating in CVAT is seen in Figure 4.3. From the annotation of the LOCO data set a JSON file in the same format as the one for COCO is generated. A clip of the file is seen in Figure 4.4, where the category id, segmentation, area, and bounding box information are included.

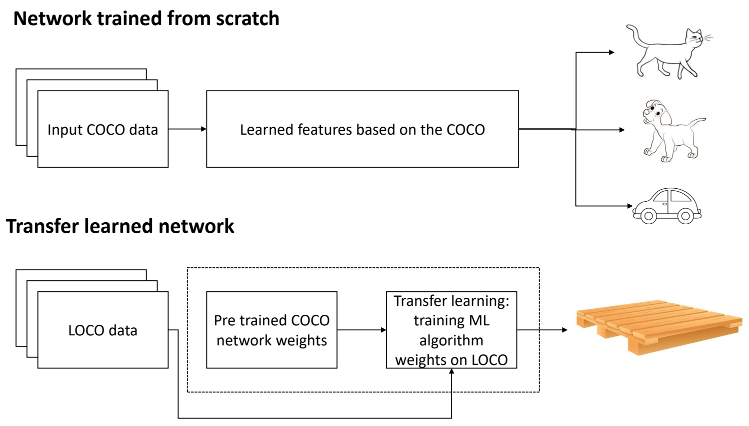


```
"category_id": 1, "segmentation": [[1137.9, 604.9, 1137.9, 625.8, 1116.2, 625.8, 1116.2, 604.9]], "area": 462.0,
"bbox": [1116.2, 604.9, 21.7, 20.9], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 218, "image_id": 10,
"category_id": 1, "segmentation": [[1167.1, 607.4, 1167.1, 634.9, 1142.1, 634.9, 1142.1, 607.4]], "area": 700.0,
"bbox": [1142.1, 607.4, 25.0, 27.5], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 219, "image_id": 10,
"category_id": 1, "segmentation": [[1721.8, 481.6, 1721.8, 559.5, 1494.3, 559.5, 1494.3, 481.6]], "area": 17784.0,
"bbox": [1494.3, 481.6, 227.5, 77.9], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 220, "image_id": 10,
 "category_id": 1, "segmentation": [[1714.9, 545.0, 1714.9, 463.1, 1915.3, 463.1, 1915.3, 545.0]], "area": 16400.0,
 "bbox": [1714.9, 463.1, 200.4, 81.9], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 221, "image_id": 10
, "category_id": 1, "segmentation": [[1273.5, 172.9, 1273.5, 227.8, 1113.8, 227.8, 1113.8, 172.9]], "area": 8800.0,
 "bbox": [1113.8, 172.9, 159.7, 54.9], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 222, "image_id": 10
, "category_id": 1, "segmentation": [[1207.4, 244.0, 1207.4, 277.7, 1070.2, 277.7, 1070.2, 244.0]], "area": 4658.0,
 "bbox": [1070.2, 244.0, 137.2, 33.7], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 223, "image_id": 10
, "category_id": 1, "segmentation": [[601.2, 227.8, 601.2, 179.1, 767.1, 179.1, 767.1, 227.8]], "area": 8134.0,
"bbox": [601.2, 179.1, 165.9, 48.7], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 224, "image_id": 10,
"category_id": 1, "segmentation": [[513.9, 90.6, 513.9, 10.8, 712.2, 10.8, 712.2, 90.6]], "area": 15840.0, "bbox":
[513.9, 10.8, 198.3, 79.8], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 225, "image_id": 10,
"category_id": 1, "segmentation": [[646.1, 442.3, 646.1, 391.2, 778.3, 391.2, 778.3, 442.3]], "area": 6732.0,
"bbox": [646.1, 391.2, 132.2, 51.1], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 226, "image_id": 10,
"category_id": 1, "segmentation": [[718.6, 623.4, 718.6, 662.4, 454.5, 662.4, 454.5, 623.4]], "area": 10296.0,
"bbox": [454.5, 623.4, 264.1, 39.0], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 227, "image_id": 10,
"category_id": 1, "segmentation": [[743.7, 615.6, 743.7, 657.2, 699.6, 657.2, 699.6, 615.6]], "area": 1804.0,
"bbox": [699.6, 615.6, 44.1, 41.6], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 228, "image_id": 10,
"category_id": 1, "segmentation": [[1046.4, 610.7, 1046.4, 592.3, 1060.2, 592.3, 1060.2, 610.7]], "area": 266.0,
"bbox": [1046.4, 592.3, 13.8, 18.4], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 229, "image_id": 10,
"category_id": 1, "segmentation": [[1102.4, 600.3, 1102.4, 621.6, 1079.8, 621.6, 1079.8, 600.3]], "area": 484.0,
"bbox": [1079.8, 600.3, 22.6, 21.3], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 230, "image_id": 10,
"category_id": 1, "segmentation": [[1099.0, 622.4, 1099.0, 604.9, 1116.6, 604.9, 1116.6, 622.4]], "area": 306.0,
"bbox": [1099.0, 604.9, 17.6, 17.5], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 231, "image_id": 10,
"category_id": 1, "segmentation": [[1070.2, 615.7, 1070.2, 599.4, 1083.6, 599.4, 1083.6, 615.7]], "area": 238.0,
"bbox": [1070.2, 599.4, 13.4, 16.3], "iscrowd": 0, "attributes": {"occluded": false}}, {"id": 232, "image_id": 10,
```

Figure 4.4: LOCO annotations in similar JSON format as for COCO.

4.1.3 Transfer Learning

As introduced, training a network from scratch is time-consuming, and requires substantial computing power and large annotated data sets. To streamline this process transfer learning was introduced to utilize already learned features in pre-trained networks and apply them in new tasks, with tuning from the training of the new data set [46]. This way, domain knowledge from general object detection is transferred, which reduces the required training time for new applications, as shown in Figure 4.5. A key requirement behind transfer learning is that the features of the new object detection task are based on similar characteristics as the one for the pre-trained weights. This applies to the objects involved in this project.



Network trained from scratch
Input COCO data
Learned features based on the COCO
Transfer learned network
LOCO data
Pre trained COCO network weights
Transfer learning: training ML algorithm weights on LOCO

Figure 4.5: Network pre trained on COCO and transfer learned on LOCO to detect pallets.

From the pre-trained COCO ML algorithm a file with weights is generated. With the pre-trained weights as a base, the weight file and the annotation file of the LOCO data set are exported for transfer learning to train a new specialized ML algorithm. The new algorithm uses the COCO weights as the foundation and specializes in detecting pallets from annotating the LOCO data set. The format exported for transfer learning is seen in Figure 4.6. It includes LOCO annotations and images from the data set broken down into four subsets for training and validation. Note that the third subset is missing since there is a mismatch in the annotation file and image filename in the LOCO subset.

```
.
├── annotations
│   ├── instances_Train.json
│   └── instances_Validation.json
├── train2017
│   └──   images
│   ├──       subset-1
│   ├──       subset-2
│   └──       subset-4
└── val2017
└──     images
└──         subset-5

9 directories, 2 files
```

Figure 4.6: File structure exported from CVAT for transfer learning.

A total of 4 485 images are used for training, containing 86 318 annotations of only pallets, while the validation dataset has a total of 1 006 images and 10 684 annotations. This results in that 81.68% of the images being used for training, and 18.32% is used for validation. As for annotations, 88.99% is used for training, while 11.01% is used for validation. The typical split between the training and validation sets is 80/20, respectively [47, p.121] thus, the LOCO subset split is sufficient.

4.2 Machine Learning Algorithm Selection

The section goes through the considerations made for choosing a suitable object detection model. Among thousands of proposed object detection ML models, a few have been selected for further investigation due to their capabilities in regard to the project's objective, they are introduced in the following subsection. Thereafter, a comparison of the different models is shown followed by the selection of the most suitable algorithm.

4.2.1 Considered ML Algorithms

To select an algorithm that is capable of satisfying the introduced requirements presented in Subsection 3.2.1 a study on different object detection networks is performed. One of the early and initial approaches for analyzing images used a sliding window approach, where a filter was slid over each part of the input image searching for Region of Interest (ROI)s, and from the defined areas using artificial networks to identify objects and classify them. For real-time applications in industrial environments, the solution's performance has been inadequate as searching for ROIs is too computationally demanding and time-consuming both in regard to training and inference. Another challenge of the initial method is its accuracy and scope of detection as objects usually appear in different aspect ratios, positions and pose, and with a variety of backgrounds. Due to the highlighted challenges, multiple improved solutions have been developed to address the issues. A key behind the improved versions is the implementation of CNN.

In the following paragraphs are some of the algorithms that use new approaches to address the challenges introduced. The algorithms also show the development within object detection as their fundamentals are based on similar techniques, but updated with better-performing methods. To evaluate the different algorithms, two important parameters are highlighted; FPS and AP, explained in Subsection 2.2.5. One of the keys to achieving a robust design is the trade-off between time and precision considering the combination of these two parameters is important.

Regional-Convolutional Neural Network

One of the algorithms that first addressed object detection with the use of CNN was R-CNN. The solution uses a selective search method, where a set number of region proposals are decided, which means that the sliding window search is limited which reduces computational time. R-CNN proposes 2000 ROIs based on areas that are likely to contain objects according to the selective search algorithm. The ROIs are further cropped and warped into similar sizes before they are fed into a CNN for feature extraction before the ROI for object recognition and classification [28], an overview of the steps are seen in Figure 4.7.

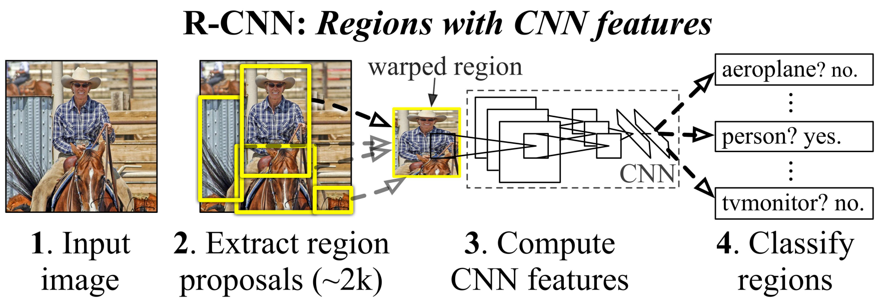


Figure 4.7: R-CNN overview [28].

One of the challenges with the R-CNN is that each ROI of the image needs to be passed through the algorithm, which means that 2000 ROIs must be processed, making the solutions computational intensive and time-consuming. Due to the challenge [48] developed an improved solution named Fast R-CNN. Instead of inputting all regions of interest, Fast R-CNN uses a CNN to first extract features from the image based on attributes such as texture, color, and edges, and from the feature extraction map ROIs are selected. The ROIs are then cropped, warped, and passed through a CNN for object recognition and classification. The technique makes sure that regions of actual interest are evaluated, which reduces the total computational need for the algorithm and substantially reduces performance time.

As an advancement of Fast R-CNN, [49] introduced Faster R-CNN as a step towards real-time object detection. Due to Fast R-CNN's ROI and the selective search technique the solution requires too much computing complexity to achieve real-time. Faster R-CNN addresses this issue by substituting ROI and selective search by a neural network that identify and learn the region proposals, called the Region Proposal Network (RPN). Similar to Fast R-CNN, a CNN is used to extract features from the image, but the identified feature map is then passed through the RPN for region proposal prediction. The method enables the model to learn from the feature map, which follows the opportunity for continuous improvements based on loss evaluation. The predicted regions from the RPN are then passed through a ROI pooling layer, before the object classification is executed for each ROI similar to the Fast R-CNN.

The Faster R-CNN is evaluated as a suitable option for the project introduced in this report, but at the same time, it still has a few issues with properly solving the object recognition and classification tasks. Among others is the ROI technique a complex solution when considering multiple objects in a total image, especially when the total R-CNN pipeline needs to be executed for each area of interest. The technique also struggles when it comes to generalizing object detection, identifying various different objects located in a greater context. As a result of the R-CNN models challenges, YOLO was introduced as an option highlighting computational speed by design. The YOLO series is introduced in the following part.

YOLO

One of the great advancements within object detection through CNN was published in the work [29], where the algorithm YOLO was introduced. The solution went from a two-stepped approach, where the object is first identified and then classified as for R-CNN networks, to a one-stepped approach where both procedures are performed in the same iteration. The key behind YOLO's design is its capability of real-time speeds while maintaining good AP. The algorithm is referred to as one of the best option for balancing the trade-off between time and accuracy. The network is also generalizable, which makes it robust in applications of new domains or with new unexpected inputs [50].

One of the major differences between YOLO and R-CNN is that YOLO approaches object detection as a regression problem, where the total image is separated into a fixed grid consisting of cells (S), resulting in an (S x S) matrix. Within each cell, a number of bounding boxes (B) are predicted, where each cell is responsible for the detection of bounding boxes within its frames. From each of the predicted bounding boxes the likelihood for objects to appear is found, resulting in a number of potentially detected objects within each cell, the likelihood of objects appearing within each cell is defined as (Pc). If the cell contains objects that belong to one of the pre-defined classes given that there are objects within the given bounding box, the object is classified (C). As a result of the network's components, the CNN predicts a vector (y) for each cell [29]. The vector includes:

- Bx and By, that represents the center coordinates of the bounding box.
- Bh and Bw, that contains the width and height of the bounding box.
- Pc, probability the bounding boxes contain objects.
- C, is the number of classes.

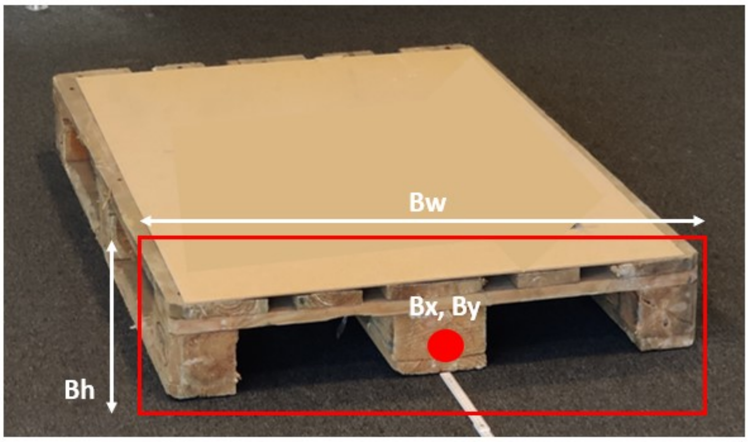


Figure 4.8: YOLO bounding box of pallet front.

To select the most suited bounding box and evaluate the performance of the predicted bounding box, YOLO uses the IoU method where it measures the correspondence between the predicted bounding box and the labeled ground truth bounding box, explained in Subsection 2.2.5. As previously described, for each grid cell the algorithm predicts multiple bounding boxes. Out of the predictions, the bounding boxes with the highest IoU related to the ground truth are selected, the technique is also known as non-maxima suppression. Based on deviation from ground truth a loss function is generated, where the algorithm will try to converge towards the value of ground truth [29].

YOLOX

Although YOLO highlighted many of the required steps to achieve real-time, accurate, and generalizable object detection, it still struggles when it comes to locating and identifying objects that are smaller, in groups, or that are closely located especially when the complete image is in a greater context. Based on the core from YOLO, presented in Figure 4.2.1, multiple updated and improved versions have been developed to address these issues. Some of the key advancements within the YOLO series is the development classification and localization techniques, as well as data augmentation, approaches [51].

The latest widely used YOLO version, YOLOX, takes a leap towards achieving an optimal trade-off between speed and accuracy compared to the other YOLO solutions across all model sizes [52]. YOLOX was designed for practical use in both R&D and in industry and is implemented in the PyTorch framework, where also the deploy version is available through ONNX and OpenVINO frameworks [53] for industrialized applications. Due to its proposed updates such as decoupled head, anchor free detection, and advanced label assignment strategy, YOLOX achieves overall better AP and higher throughput compared to its predecessors. A further explanation of YOLOX architecture and why it is suitable for this project is presented in the following subsection.

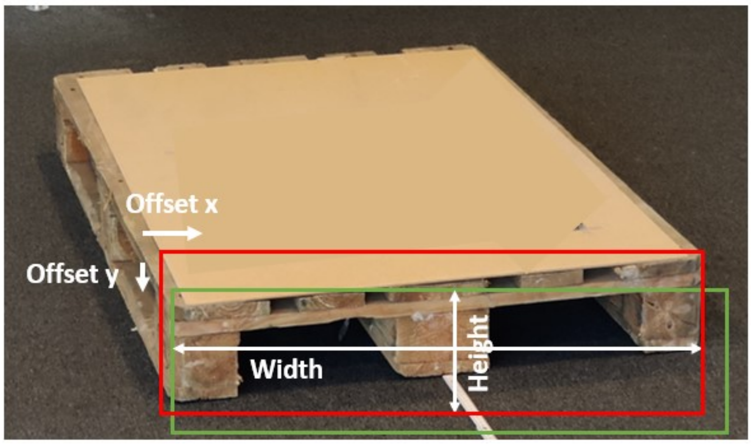


Figure 4.9: YOLOX tuning of pallet detection by offset values.

A key difference in YOLOX compared to previous versions is its anchor/ bounding box free detection technique, which reduces the number of predictions per grid cell and the complexity related to optimizing bounding boxes, where the both affect the overall latency of the network. Switching the algorithm to anchor box free manner is fairly simple as the number of predictions per grid cell is changed to only detect one, followed by each grid cell directly predicting the offset of the bounding box compared to ground truth as well as the width and height of the bounding box, seen in Figure 4.9. The generated values are then used to train and improve the algorithm. YOLOX does also use a decoupled head approach, where it goes from a single pipeline with both object classification, regression, and localization, as was used in the previous YOLO version, to a distributed pipeline for classification, regression, and localization. The result of the technique improves convergence speed, which is essential when training an end-to-end object detection model. YOLOX also takes advantage of advanced data augmentation to increase the diversity and variety of input training data, where realistic modification to the existing input data is applied [52].

The YOLOX architecture utilizes multiple network backbones such as Darknet and CSPnet. The backbones consist of different structures of CNN, with varying network depths and compositions. There are a total of 7 different YOLOX versions [54]. These versions differ foremost in the number of network parameters, which greatly affect the required computing power to keep the throughput high. Most of the YOLOX versions are based on Modified CSP v5, the exception is YOLOX-DarkNet53 which uses the Darknet-53. For this project YOLOX-S is highlighted, which is based on the Modified CSP v5 backbone. The version is preferred due to its sufficient parameter size for the hardware available for this project, as well as its good trade-off between speed and accuracy compared to its counterparts with similar network size [55].

4.2.2 Selection of ML Algorithms

To find the most suited solution for the project, a qualitative comparison followed by the pros and cons of the introduced ML models is performed. The comparison is further used to evaluate the different object detection models against each other, to make a selection on which to use for object detection in this project.

Qualitative comparison

To directly make a qualitative comparison of object detection models is difficult as the different models are tested with multiple different backbones and on different data sets. Other varying parameters found when evaluating the models are varying training time, which directly affects AP, and also what hardware the training is executed on influencing the FPS. To compensate for these uncertainties, to tables are created to compare the different models. The first table, seen in Table 4.2, uses information from the papers where the object detection models' were published to get an initial understanding of their performance, followed by a pros and cons comparison. This table does not consider varying data sets and training time but provides an overall assessment of the models. The second table, seen in Table 4.1, tests the AP performance of the different models on similar dataset, where the COCO test-dev is used. Table 4.1 also includes performance parameters for a few more of the recent versions of R-CNN and YOLO. The comparison gives a more accurate understanding of how the models perform compared to each other but is only tested for AP. Together the two tables provide a sufficient understanding of the advantages and limitations of the different models, which makes it possible to make a final conclusion on which model to use for object detection in this project.

Comparison based on the COCO test-dev dataset:

Table 4.1: Comparison of the object detection algorithms based on the same COCO test-dev dataset [56]. Note that the comparison is only performed for AP.

Model	**Backbone**	**Data set**	**AP**
Fast R-CNN	VGG-16	COCO test-dev	19.7%
Faster R-CNN	VGG-16	COCO test-dev	37.4%
Mask R-CNN	ResNetxT	COCO test-dev	39.8%
YOLO v3	Darknet 53	COCO test-dev	33%
YOLO v4	Darknet 53	COCO test-dev	43.5%
YOLOX-S	Modified CSP v5	COCO test-dev	51.2%

General comparison of object detection algorithms:

Table 4.2: Overview of ML algorithms' performances. The numbers are collected from the papers where the algorithms were published. Note that the algorithms are tested on different data sets and hardware, which means that the comparison does not directly compare the algorithms against each other. Accordingly, the overview is foremost used as an outline to evaluate the solutions separately and how they fit the project. See Table 4.1 for a more direct comparison.

Model	**Data set**	**AP**	**FPS**	**Pros**	**Cons**
R-CNN [28]	Pascal VOC	66%	6	Easy to interpret results, accurate results, multiple design options for adjustments.	Slow, costly computations, multiple complex steps.
Faster R-CNN [49]	Pascal VOC	73.2%	7	Accurate predictions, tolerable balance between AP and FPS, efficient training compared to other R-CNN algorithms.	Many components that reduces FPS performance and creates potential bottlenecks.
YOLO [29]	Pascal VOC	63.4%	45	Fast as it only process the inputted image once, AP is good compared to processing time, quickly identifies multiple objects.	Struggle on smaller objects and when multiple object are located in the same grid cell, requires pre work to define and tune anchor boxes.
YOLOX-S [52]	COCO	39.6%	102	Fast, good trade of between FPS and AP, simple design, little tuning needed.	Struggle with detecting multiple objects within single grid cells, accuracy issues when detecting smaller objects.

Selected object detection model

YOLOX-S is the preferred solution to fulfill the requirements, seen in Table 4.3, and achieve the objective for this project. As seen in Table 4.2, YOLOX-S is performing at a much higher speed compared to the other models. On the other hand, the AP is not as good as for the other models but is due to it being tested on a much more complex data set. As shown in Table 4.1, where the models are compared on the same data set YOLOX-S, is performing well with satisfying results in regard to the project's defined requirements. Summarized, YOLOX-S is the algorithm that best satisfies the trade-off between AP and

FPS, which is an important target for this project. YOLOX-S is also a preferred solution due to its efficient design, which makes it suitable for constrained and mobile devices where processing efficiency is key, while accuracy and speed need to be ensured [57]. YOLOX-S's simple parameter tuning also makes a good option for industrial application where operative robustness and transparency is key [58]. In the following section is the implementation of YOLOX-S explained.

Table 4.3: YOLOX-S performance compared to requirements presented in Section 3.2.

Parameter	**Project Requirement**	**YOLOX-S (from paper [52])**	**YOLOX-S (from COCO test dev [56])**
Throughput	30 FPS	102 FPS	Test only performed for AP
AP	40%	39.6%	51.2%

4.3 Implementation of YOLOX-S

The section introduces the key actions for training and implementation of the YOLOX-S network. The implementation is followed by optimizing measures with Intel OpenVINO. An explanation of the optimizing tool as well as the conversion from Python to OpenVINO is presented as well.

4.3.1 Training YOLOX-S

By following [59], a custom exp file has been created in the COCO format by copying the yolox_s.py as a template from [60]. The final custom exp file is listed in Appendix D as Listing D.1 and is using the default training settings recommended in [59]. The pretrained COCO weights `yolo_x_s.pth` from [54] is copied and renamed, then training is initialized by running `train.py` with the arguments shown in Appendix D as Listing D.2. All of the training parameters is located in Appendix A as Table A.1.

The results of the training for every epoch is shown in Appendix A as Figure A.1, Figure A.2 and Figure A.3. The best epoch from the COCO evaluations is 294 and the YOLOX-S trained on the LOCO dataset results is shown in Table 4.4. From the same Table 4.4 it is seen that the algorithm is providing an AP of 24.0%, and an AP50 of 53.2%, which means that the predicted bounding boxes is at least 50% match with ground truth, as introduced in Subsection 2.2.5. It is also shown that the inference time is good, 6.74 ms, which equals 148 FPS.

Table 4.4: COCO evaluation results for YOLOX-S training epoch 294. Inference time is from evaluation during training which has been run on a NVIDIA GeForce RTX 3060 Laptop GPU.

Model	Parameters	Dataset	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	Inference time
YOLOX-S	9.0 M	LOCO	24.0	53.2	17.2	7.9	24.3	40.6	6.74 ms

4.3.2 Optimization of YOLOX-S Inference

For ML frameworks such as PyTorch, utilizing optimization tools to accelerate inference performance for embedded applications is vital. Reducing the floating-point precision from FP32 to FP16 will reduce the computation time with the trade-off of precision, which is very minimal as testing with Retina-Net show an accuracy drop of only 0.01 percent [61]. Optimizing the neural network increase the efficiency, thus making it capable of high throughput inference on low-ended devices, such as the NVIDIA Jetson compute series [62]. Other computing devices on the edge include Video Processing Unit (VPU) and Neural Processing Unit (NPU) devices which is specific hardware that is capable of achieving high throughput-to-power ratio compared to Graphics Processing Unit (GPU)s and CPUs [63]. This also allows integrating the object detection using languages such as C++.

Intel OpenVINO

Intel OpenVINO is a toolkit that facilitates the transition between training and deployment environments and optimizes code inference performance [64]. The solution has the capability to be applied across different hardware platforms e.g. CPU, CPU, and FPGA, which makes it great for prototyping purposes. Figure 4.10 shows a diagram of three different processes for deploying a trained model using OpenVINO. The different types include paddle and ONNX file formats, where the model is sent directly to the OpenVINO runtime, while for the third type it is passed through the model optimizer which creates an Intermediate Representation (IR) of the model that can be inferred by the OpenVINO runtime [64]. For this project, the model is converted to the ONNX file format before it is optimized in the OpenVINO Runtime engine in order to use the OpenVINO Model Optimizer tool, further described in the following paragraphs.

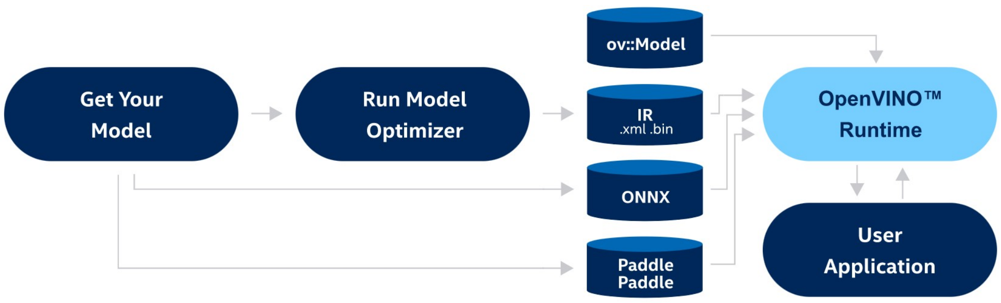


Figure 4.10: Deploying Intel OpenVINO [64].

Implementation of Intel OpenVINO

To use OpenVINO, is it required to first convert the PyTorch model to Open Neural Network Exchange (ONNX) format, which is an open standard for machine learning models [65]. By following [66] is the model converted to ONNX format. The command is shown in Listing D.3.

After the PyTorch model has been converted to an ONNX model which is a single file with the .onnx file extension is it possible to convert the model into multiple other versions such as Caffe, mlir, TensorRT or OpenVINO. The tutorial at [67] explains how to convert from ONNX to OpenVINO IR using python. The full command is in Appendix D as Listing D.4.

Table 4.5: Comparison of inference time with YOLOX-S between the Python demo and OpenVINO implementation on a 11th Gen Intel Core i5-11300H with 8 cores running at 4.40GHz.

Framework	**Device**	**Inference latency in milliseconds**	**Standard Deviation**	**Frames**	**Description**
PyTorch Python	CPU	191.53	30.07	762	Running the tools/demo.py with warehouse video.
OpenVINO C++	CPU	70.49	3.69	762	Running run_object_detection() with warehouse video.

A comparison of the OpenVINO and the YOLOX PyTorch implementation is performed to evaluate the inference latency. A test video from the local warehouse is used as input, containing 762 frames of multiple pallets. The results from Table 4.5 show that the inference latency is reduced from an average of 191.53 to 70.49 milliseconds, which is over 2.5 times lower latency.

4.4 Object Filtering Method

The pose estimation method is only interested in a single object; however, the object detection algorithm detects all perceived pallets in the image. Therefore, a bounding box filtering method is required to select the single desired pallet. The desired outcome is to select the pallet closest to the camera's front.

The bounding box selection algorithm is based on a custom weight score function depending on the current location of the bounding box. The weight scoring system can be split into two steps; first, a score for the bounding box y position is calculated, and second, the score for the x position is calculated. The final score is the sum of the x and y position scores compared to all the bounding boxes. The bounding box with the highest weight score is selected and used for the pose estimation.

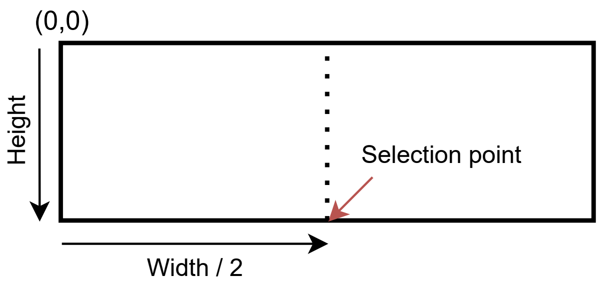
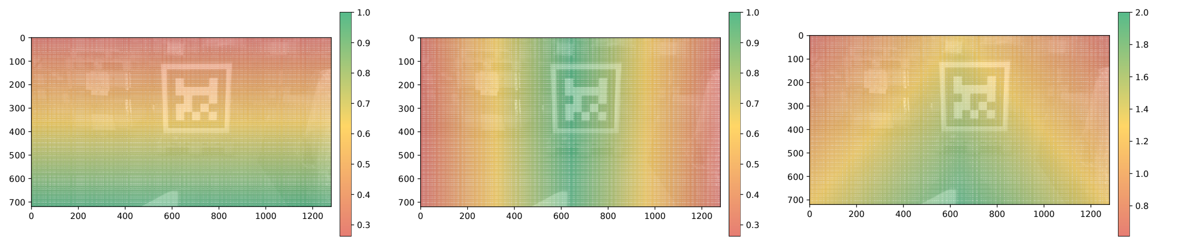


Figure 4.11: Bounding box selected center point.

The definition of the bounding box center has been adjusted to select the closest bounding box. As shown in Figure 4.11 has, the center has been translated down to be the lowest point on the bounding box. This point coordinate is used to evaluate the bounding box weight score.



(a) Heat map of vertical selection. (b) Heat map of horizontal selection. (c) Weight sum of 4.12a and 4.12b for final weight score.

Figure 4.12: Three different pallet selection heat maps where 4.12c is the sum of 4.12a and 4.12b.

A color heat map of the weight scoring system is shown in Figure 4.12. Using the 4.12c heat map, a bounding box at coordinate (800x600) score higher than a bounding box at (200x200). If only these two boxes were detected in the image, would the (800x600) bounding box be selected due to the higher weight score. Overall the filter selects the pallet closest to the center of the camera.

4.5 Chapter Summary

In Chapter 4 the implementations to realize object detection of pallets is shown. The two key steps are data annotating of the LOCO data set and then transfer learning of the COCO pre-trained YOLOX-S. Different ML algorithms have been evaluated for the purpose, but YOLOX-S has proven to be the best solution due to its good trade-off between AP and FPS as well as its simple design and good robustness. To make the algorithm suitable for real-life applications the YOLOX-S algorithm is optimized by Intel's OpenVINO, which improves the algorithm's performance on constrained devices. The output of the object detection solution is boxes around the detected pallets followed by x,y coordinates of the pallets' center. The output is further integrated into the system design for pose estimation, where the pose estimation model uses the boxed pallet to generate a 3D point cloud that is processed and analyzed. The following chapter 5 will go through the steps needed to apply the arithmetic estimation calculations on the detected pallet to identify its pose.

5. Pose Estimation

Based on the algorithm developed in Chapter 4 the detected object's pose is calculated. To achieve 6D pose estimation introduced in Section 1.2, collecting 3D data as a point cloud is needed. The chapter introduces a camera with satisfying specifications to achieve this task, followed by the implementations to integrate the sensor. To process the generated data an introduction to the point cloud processing measures are presented, followed by the techniques used to analyse the point cloud to find the 6D pose trough arithmetic calculations, plane estimation, and vector calculations. In the end of the chapter, a performance analysis tool is presented implemented to evaluate the performance of developed pose estimation model.

5.1 3D Data Acquisition

To generate a point cloud of the pallet a suitable sensor is selected. The sensor is presented in the section followed by its specifications and capabilities. A key part of creating and analyzing the generated depth image is the translation from 2D to 3D projections, where mathematical functions are used to identify points of the pallet within a 3D environment. The steps to execute the translation is described as well in the section.

5.1.1 Intel Realsense LiDAR Camera

To generate a depth image Intel's Realsense LiDAR camera is selected, seen in Figure 5.1. The sensor enables high and accurate depth data collection, which is packaged into a fairly small unit that makes the design flexible and easy to work with. The sensor consists of three main components: a single packaged accelerometer and gyroscope, RGB image sensor providing colored data, and 860 nm IR laser generating the depth point cloud [68]. An overview of the sensor's specifications is seen in Table 5.1.

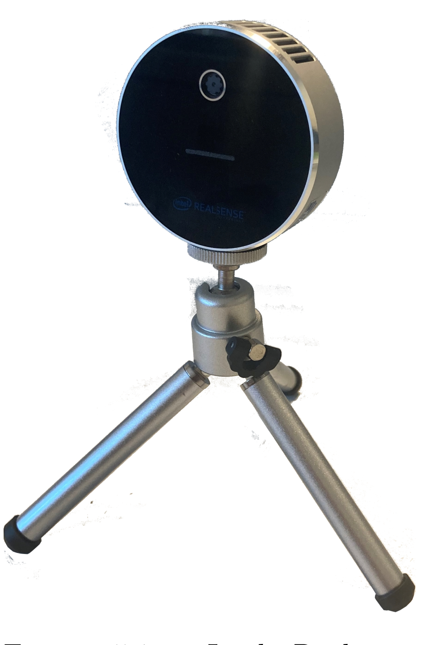
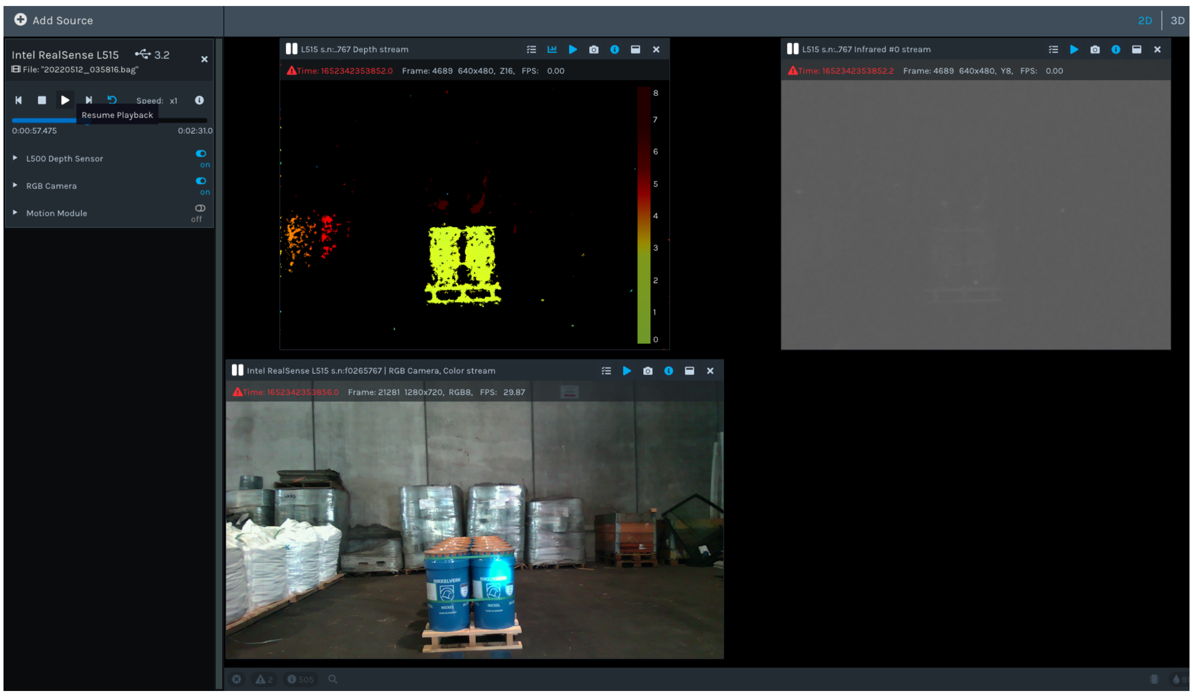


Figure 5.1: Intel Realsense RGB-D Sensor.

Table 5.1: Intel Realsense specifications [68].

Specification	**Metric**
Depth capture in meters	0.25 to 9
Capacity (depth) in FPS	30 @ 1024x768 resolution
Capacity (color) in FPS	30 @ 1920x1080 resolution
Field of view (depth) in degrees	(70 x 55)±2
Field of view (color) in degrees	(69 x 42)±1
Dimensions (Diameter x Height) in millimeters	61 x 26
Power consumption in watt	3.5

To process the data generated from the RealSense, Intel offers a Software Development Kit (SDK), where the generated depth and colored data is logged, visualized, and qualified. The SDK contains multiple features including RealSense Viewer used to visualize the RGB image and point cloud, calibration and depth quality tools utilized in optimizing and tuning the sampled data, and debug tools that generate log files. An image of the Intel RealSense Viewer when it is applied to the forklift operation is seen in Figure 5.2, where the sampled image and point cloud are displayed [68, 4].



Add Source
2D
3D
Intel RealSense L515
3.2
File: "20220512_035816.bag"
Speed: x1
Resume Playback
0:00:57.475
0:02:31.0
L500 Depth Sensor
on
RGB Camera
on
Motion Module
off
L515 s.n:..767 Depth stream
Time: 1652342353852.0 Frame: 4689 640x480, Z16, FPS: 0.00
L515 s.n:..767 Infrared #0 stream
Time: 1652342353852.2 Frame: 4689 640x480, Y8, FPS: 0.00
Intel RealSense L515 s.n:f0265767 | RGB Camera, Color stream
Time: 1652342353856.0 Frame: 21281 1280x720, RGB8, FPS: 29.87

Figure 5.2: Scene captured by Intel Realsense. In the upper part of the image is the generated point cloud and in the lower part is the RGB image.

The Intel RealSense l515 interface with the system using the Intel RealSense SDK 2.0; this enables the use of functions specified for the Intel RealSense camera system. The SDK does also contain multiple software wrappers, which makes it simple to convert it into OpenCV image matrix format and PCL point clouds accordingly [68]. Implementing two functions in the configuration enables the software to have two modes. The first mode is to load the data from a pre-recorded Robot Operating System (ROS)-bag sequence, which contains all images, point clouds, Inertial Measurement Unit (IMU) data, and camera intrinsic parameters [4]. While the second mode, is to use the Universal Serial Bus (USB) port for connecting the Intel RealSense directly for live operation, all of which is configured in the `PoseEstimation::setup_pose_estimation()` function. It is important to note that the real-time playback is enabled to read all of the frames with no frame drops if a pre-recorded ROS-bag is used, as implemented in Listing D.6.

5.1.2 2D to 3D Projections

The output from the object detection is 2D points from the camera frame. These points are required to be mapped into 3D space, and the camera model is essentially describing the mapping between Euclidean 3D space to the Euclidean 2D image [69, p.153].

The conversion from the 2D image detection points to the 3D vectors is done in the *PoseEstimation::calculate_3d_crop()* function. First the custom struct output from the object detection, which contains the data (x,y, width, high, and confidence) for a single filtered bounding box is converted into four 2D corner points, representing the four corners of the bounding box. The object detection output is shown in Figure 5.3. This struct is the main interface between the `ObjectDetection.h` class and the `PoseEstimation.h` class.

```
struct object_detection_output {
  uint16_t x;
  uint16_t y;
  uint16_t width;
  uint16_t height;
  double confidence;
};
```

Figure 5.3: Object detection struct; the link between the `ObjectDetection` and `PoseEstimation` classes, and the link between phase 2 and 3 in Figure 3.3

To understand how a 2D point is converted into a 3D vector, is it required to understand how to find the K matrix, also known as the calibration matrix.

According to the pinhole camera model, a point in space with coordinates $\mathbf{X} = (X, Y, Z)^T$ is mapped to a point in the image plane where a line connecting the point $\mathbf{X}$ to the camera origin meets the image plane. This is illustrated in Figure 5.4. As shown, the point $\mathbf{X} = (X, Y, Z)^T$ is mapped to the red point $(f \cdot X/Z, f \cdot Y/Z, f)^T$ on the image plane.

Where the Z-axis intersects, the image plane is called the principal point, which is the center of the image. As for CMOS sensors, is this point not in the center of the image, so an x and y offset is set.

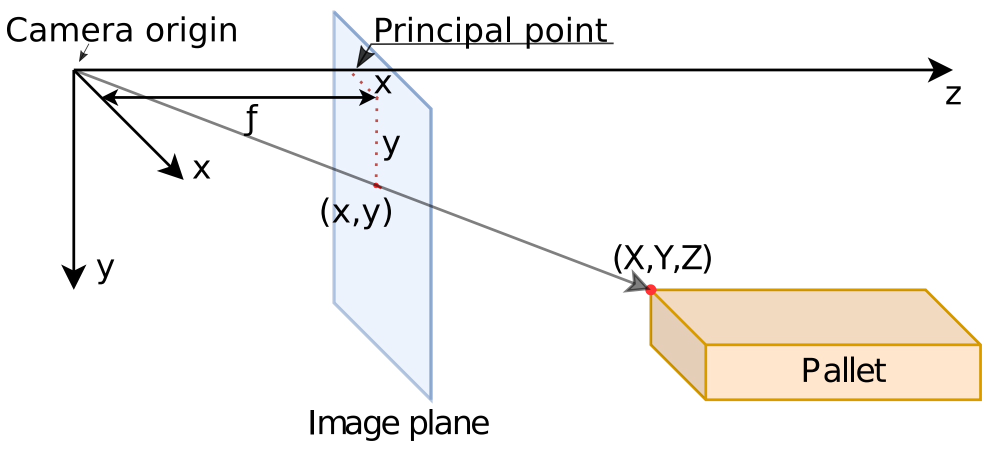


Figure 5.4: Perspective projection using the pinhole camera model.

If there is skewing on the pixel array, this must be calibrated as s. However, camera manufacturing today uses soft lithography to produce the CMOS sensors which are so precise that the skew can be assumed to be zero [70].

These values are the intrinsic camera parameters, which are represented as the 3x3 K matrix shown in Equation (5.1). There is no world coordinate frame as only the intrinsic camera parameters are used. Thus, every coordinate is related to the camera frame where $0, 0, 0$ is at the camera origin [69, p.157].

$$K = \begin{bmatrix} a_x & s & c_x \\ 0 & a_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

Table 5.2: Description of Equation (5.1) with values from Intel RealSense l515 calibration [69, p. 163].

Symbol	**Description**	**Value in pixels**
a_x	Scale factor in the x-coordinate direction.	907
a_y	scale factor in the y-coordinate direction.	907
c_x	X coordinate of the principal point.	663
c_y	Y coordinate of the principal point.	367
s	Skew is set if the x and y-axis are not perpendicular.	0.00

Using Equation (5.1) on all four 2D points from the object detection bounding box allows mapping the 2D points into 3D vectors in the camera frame, which will be used for the point cloud filtering in the following sections.

5.2 Point Cloud Processing

A point cloud is a set of 3D data points where each point is represented by three coordinates in a Cartesian coordinate system [71]. The point cloud used evaluate the pallet's pose is extracted from the solid-state IR LiDAR sensor of the Intel Realsense RGB-D camera. The arithmetic point cloud calculations aim to find a 3D vector and 3D point corresponding to the pallet's pose from the estimated plane and homogeneous camera vectors. The objective of the procedure is to identify where and how the forklift should position itself to enter and lift the pallet correctly. The following subsections go through some key elements to generate and process the point cloud.

The pointcloud from the Realsense device contain both point position in 3-dimensions and color values and is received as `rs2::pointcloud`, this data is converted to `pcl::PointCloud<pcl::PointXYZ>` with the `PoseEstimation::points_to_pcl()` function by using the PCL library. When all points is converted to a `pcl::PointXYZ`, the next step is filtering out the irrelevant points.

5.2.1 Point Cloud Filtering

Once the point cloud has been received from the Intel Realsense, the relevant points are first filtered using the `pcl::io::FrustumCulling<pcl::PointXYZ>` filter. The FrustumCulling filter is typically used to filter relevant points within a camera view from a point cloud. A pseudo camera is created from the detected object output, where the relevant points are only the once of the detected pallet and its parameters. The camera pose and FoV is dynamically created from the four (2D to 3D) projections vectors mentioned in Subsection 5.1.2. These four vectors are merged into a single center frustum vector which is used to derive the required angles for the frustum filter. The angles for both the camera pose and FoV for the frustum filter is calculated using basic vector dot products and trigonometry, such as the angle between two vectors shown in Equation (5.2).

$$\theta = cos^{-1}\left(\frac{\vec{A}\cdot\vec{B}}{|\vec{A}||\vec{B}|}\right) \tag{5.2}$$

Both $\vec{A}$ and $\vec{B}$ are 3D space vectors, however the vectors is projected onto two perpendicular planes resulting into four 2D vectors in which the angle is derived. The implementation is listed in Appendix D as Listing D.7 and the FoV implementation in Listing D.8. The angle output and FoV is then used for the frustum filter, thus resulting in a final output pointcloud `local_pallet` containing only the relevant pallet points. The frustum filter implementation is listed as Listing D.9 in Appendix D.

5.2.2 Plane Estimation with RANSAC

When only the relevant point cloud points of the pallet are extracted, the next step is to find the plane corresponding to the object's pose. To achieve this task the `pcl::SamplingSurfaceNormal()` and multiple RANSAC algorithms are executed in sequence. After the procedure is performed, the next step is to calculate the 3D point of the pose vector. The 3D point is the intersection between the frustum center vector and the pallet front plane. Both of the two steps are explained in the following paragraphs.

The pallet position is assumed to be located on the floor. First, the point cloud that consist of `pcl::PointXYZ` is run through `pcl::SamplingSurfaceNormal()` resulting in a new point cloud that consists of `pcl::PointNormal` points. The first RANSAC is performed parallel to the camera y-axis within 0.1 radians, where all the inliers represent the ground floor, resulting in a plane model that is later used. The ground floor inlier points are then removed from the rest of the cloud using `pcl::ExtractIndices<pcl::PointNormal>` filter. Accordingly, the remaining normal points are off the pallet front, and a second plane RANSAC is performed with `pcl::SACSegmentationFromNormals<pcl::PointNormal,pcl::PointNormal>`, however this time with the remaining points that are perpendicular to the camera y-axis within 0.1. The second calculated plane showing the pallet front is then also saved for later use. Implementation is in Appendix D as Listing D.11.

Multiple settings have been configured for the plane detection. Other than the `pcl::SACMODEL_NORMAL_PLANE` mentioned above, the `setAxis()` and `setEpsAngle()` are configured with the axis and angles mentioned above. The `setOptimizeCoefficients()` refines the plane results by optimising the plane using the least-squares method, this improves the plane estimation with the trade-off of computation time. The `setMaxIterations()` sets the maximum iteration for the optimization while the `setDistanceThreshold()` set the maximum distance for the points used as innliers in the model. The segmentation output is the plane coefficients as the plane (x,y,z) normals and the Hessian component d which defines the plane's distance to the origin as defined by Equation (5.6), where the plane's normal vector is $\vec{n} = (n_x, n_y, n_z)$. The inliers points used to calculate the plane is also included in the RANSAC output [2].

$$n_x = \frac{a}{\sqrt{a^2+b^2+c^2}} \quad (5.3) \qquad n_y = \frac{b}{\sqrt{a^2+b^2+c^2}} \quad (5.4) \qquad n_z = \frac{c}{\sqrt{a^2+b^2+c^2}} \quad (5.5) \qquad p = \frac{d}{\sqrt{a^2+b^2+c^2}} \quad (5.6)$$

After the first plane is calculated all of the plane outliers are extracted using the `pcl::ExtractIndices<pcl::PointNormal>` filter inverted by setting the parameter `setNegative()` *True*. Listing D.12 shows the implementation with PCL.

After the outliers is extracted a second RANSAC is performed similar to the first, except that the second RANSAC uses the normal points that is perpendicular to the camera's z-axis

Table 5.3: Description of Equation (5.3)-(5.6) [72, p. 540].

Symbol	Description
p	Distance of the plane from the origin.
d	Hessian normal form constant.
a	First parameter of general equation of a plane.
b	Second parameter of general equation of a plane.
c	Third parameter of general equation of a plane.

within 0.1 radians. The implementation is similar to Listing D.11 with the changes shown in Listing D.13.

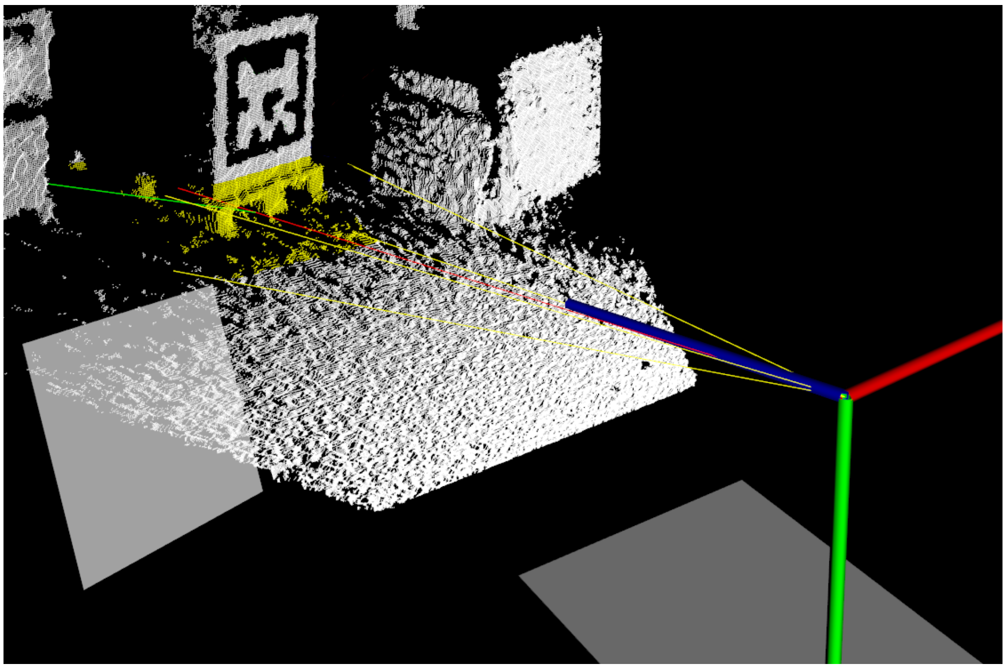


Figure 5.5: Point cloud output with the two generated planes from the object corner vectors and points marked in yellow.

Figure 5.5 shows the output of the two RANSAC planes, the first plane is of the floor perpendicular within ±0.1 radians to the camera Y-axis, while the second plane is from the front points of the pallet, perpendicular to the camera Z-axis within ±0.1 radians.

5.2.3 Pallet Pose Vector

After the floor and pallet front planes are calculated, the final operation is to calculate the pose vector. This is done in two steps; first, the 3D points of the vector's origin are found. This is calculated by finding the intersection point between the center-frustum vector and pallet front plane. The point is located on the center-frustum vector so a scalar is required to find the distance, which is calculated using Equation (5.7).

$$S_I = \frac{-\vec{n} \cdot \vec{w}}{\vec{n} \cdot \vec{u}} \tag{5.7}$$

Table 5.4: Description of Equation (5.7) [73, p. 61].

Symbol	**Description**
S_I	Distance scalar.
$\vec{w}$	Plane origin.
$\vec{n}$	Plane normal vector.
$\vec{u}$	Line direction vector (center frustum vector).

The implementation is listed in appendix as Listing D.14 and Listing D.15.

Table 5.5: List of the final output variables corresponding to pose component.

Pose component	**Code variable**
X value for position	`plane_frustum_vector_intersect_.x`
Y value for position	`plane_frustum_vector_intersect_.y`
Z value for position	`plane_frustum_vector_intersect_.z`
x value of orientation vector	`second_ransac_model_coefficients_.at(0)`
y value of orientation vector	`(-1*first_ransac_model_coefficients_.at(2))`
z value of orientation vector	`second_ransac_model_coefficients_.at(2)`

The final output is a 3D point and a 3D vector, which together represent the pose of the pallet. All of the values, representing the pose are seen in Table 5.5. Three views of the final output are presented in Figure 5.6.

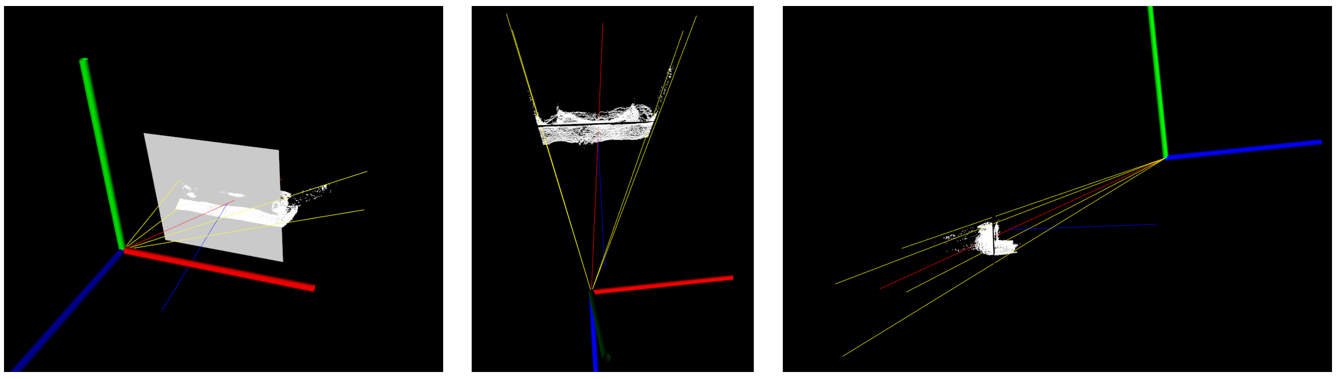

(a) Axonometric view. (b) Top view. (c) Side view.

Figure 5.6: Three different views of plane RANSAC, note that the camera frame Z and Y-axis are inverted. The corners of the object detection are represented as four yellow 3D vectors intersecting at the camera frame origin. The center frustum vector is the average of the four yellow vectors, represented as the red vector. The blue vector shows the orientation of the pallet, and the 3D intersection point between the blue, red, and plane is the translated pallet position relative to the camera frame.

5.3 Performance Analysis Tool

A ground truth comparison method is established using Fiducial markers to evaluate the developed pose estimation model's performance. The method is developed with a focus on flexibility such that it can be applied in multiple scenarios without the need for specific measurements or test site requirements. The method can also identify ground truth from a sufficient field of view and different distances. An explanation of the Fiducial method and the implementation of an AprilTag marker is presented in this section.

5.3.1 AprilTag Marker

To evaluate the performance of the pallet pose estimation, a single AprilTag has been used, located perpendicular to the camera, for the best visibility. The AprilTag has the lowest angle error at different degrees when using a single marker compared to other state-of-the-art fiducial markers, such as ARTag, ArUco, and STag [26]. A single 16.7-centimeter wide tag can have a detection range of up to 4.5 meters before deteriorating [26, 74]. The printed AprilTag has a dimension of 0.535 x 0.535 meters, and the test was done at 2 meters from the pallet. A large AprilTag marker is used to evaluate the pallet ground truth and is placed on top of the pallet. Due to its size the larger marker allows for better range. Figure 5.7 shows the pallet with the detected AprilTag marker on top.

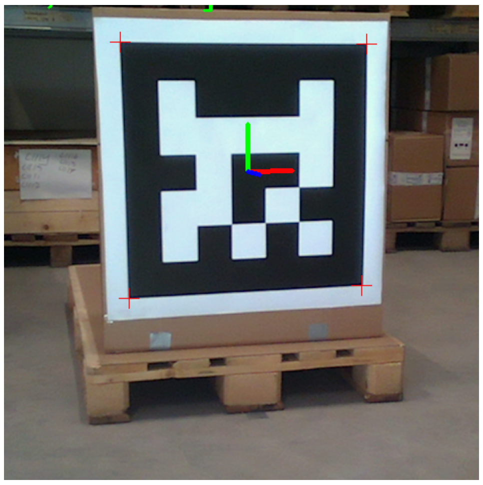


Figure 5.7: AprilTag with an angle of -19.38±0.27 degrees 2 meters from the camera.

To evaluate the accuracy of the AprilTag, [26] has done multiple angle tests with the most common tags in use, including AprilTag, and evaluated each tag towards an OptiTrack MoCap system.

The AprilTag marker can detect the pose within 1 degree of error between 10 and 80 degrees. Testing two different camera sensors shows that higher contrast improves the pose estimation [26] and that a larger marker improves the detection range. These are considerations made before implementing the AprilTag. A known issue with AprilTags is motion blur, where the detection is completely lost; thus, the project's tests are performed standing still or at low speeds [26]. Adjustments on the camera sensor settings can improve the motion blur by limiting the exposure time or applying posterior filtering [75, 76, 77].

5.3.2 Implementation of AprilTag marker

To reference ground truth, the AprilTag can detect and estimate pose with the use of only a single camera. OpenCV is used as the computer vision library, supported by the *opencv_contrib* library that is built alongside, as this extra library contains the latest modules for the OpenCV project. Modules such as the `aruco` contains a list of fiducial marker dictionaries for detection and pose estimation functions. Listing D.16 shows the required variables for detecting the marker and marker pose. A predefined dictionary `cv::aruco::DICT_APRILTAG_25h9` is used which is from the 2nd generation of AprilTag 25h9. The first number, 25, indicates the number of bits the tag contains, while the second number after "h" is the hamming distance that sets the error correction with the trade-off of the number of available id's [74][78, p. 939].

5.4 Chapter Summary

Chapter 5 shows the complete implementation of pose estimation of pallets by depth data generation, point cloud processing, arithmetic calculations, RANSAC, and plane estimation. It also explains the implementation of AprilTag markers that provides a sufficient representation of ground truth by a flexible and robust design. From the two steps, first object detection, and than based on the output estimation of the pallet's pose, the algorithm is capable of finding the 6D needed to satisfy the task. Based on the implemented step-by-step model and the ground truth method, the following Chapter 6 presents tests and experiments executed to evaluate the performance of the developed solution.

6. Experiments and Results

The chapter introduces the test scenarios that are used to evaluate the performance of the developed object detection and pose estimation algorithm. The first part of the chapter presents the test scenarios with their setup and purpose. Thereafter, the results are presented through graphs and plots, analyzing the data generated from each test scenario in regard to the requirements presented in Subsection 3.2.1.

6.1 Experiments Scenarios

To evaluate the developed model two scenarios are tested. They are selected with the intention of assessing the model in different contexts both related to accuracy where the model is evaluated against ground truth from standstill samples and in a moving condition, where processing speed has more influence on the final result. The purpose of the two scenarios is to evaluate the model toward implementation in real-life operations. Accordingly, the two scenarios are tested in Red Rock's warehouse where the goal is to evaluate the model's performance in a closed environment. The first warehouse test uses stand-still samples to assess the model's accuracy for all 6D parameters, while the second warehouse test is based on the same parameters, but this time the camera is in motion that is the way the final product will be operating. An overview of the two tested scenarios is presented in the list below and closer described in the following subsections.

- Scenario 1: Test of model by 3 seconds stand still samples at 2 meters from 0° to 45°, and evaluate the results based on the requirements from Subsection 3.2.1. Additionally, test of model beyond 45° and evaluate the solution's capabilities and limitations.
- Scenario 2: Test of model when camera is in motion at 2 meters distance from 0° to 45° and evaluate results in regard to what highlighted in Expectations from Subsection 3.2.2. Comments on results beyond 45° to identify capabilities and limitations.

6.1.1 Scenario 1: Test of Model by Stand Still Samples

Scenario 1 is a standstill test that evaluates the model from different angles. The AprilTag marker representing ground truth is mounted on top of the pallet, seen in Figure 6.2b), and the program generating the ground truth parameters is run in parallel to the pose estimation code. The camera is positioned in front of the pallet, and a 3-second sequence is captured. The tests are done at 2 meters distance from the pallet seen in Figure 6.1, which shows a top-down view of the test scenario. To satisfy the requirement of testing the proposed solution from 0° to 45°, and evaluate how far it can detect going towards 90°, 14 different tests are captured from different angles around the pallet. They are separated into the same amount of cases, where the μ and σ values are calculated for each test. An overview of the cases and their respective angles are seen in Table 6.1.

The purpose of the test is to evaluate the parameters of the vector showing the pallet's 6D pose relative to the vector generated from the ground truth AprilTag marker. Based on the deviation between the two values at different angles and distances a representation of the model's performance is created and is further analyzed.

Table 6.1: 14 different cases to test the proposed pose estimation model from multiple angles. Note that the exact angle values of the camera are captured from the ground truth AprilTag at the sample position and not by pre-measurements. The (±) values take into account the minor variations in sampled values. This was done to ensure the best possible precision.

Case:	**Angle [°]:**	**Case:**	**Angle [°]:**	**Case:**	**Angle [°]:**
Case 0	-0.21±0.03	Case 5	-52.64±0.16	Case 10	-81.39±0.56
Case 1	-18.61±0.02	Case 6	-61.42±0.07	Case 11	-84.74±0.37
Case 2	-25.11±0.04	Case 7	-66.08±0.17	Case 12	-86.96±0.76
Case 3	-35.42±0.06	Case 8	-71.22±0.18	Case 13	-88.28±0.58
Case 4	-43.31±0.06	Case 9	-77.50±0.14		

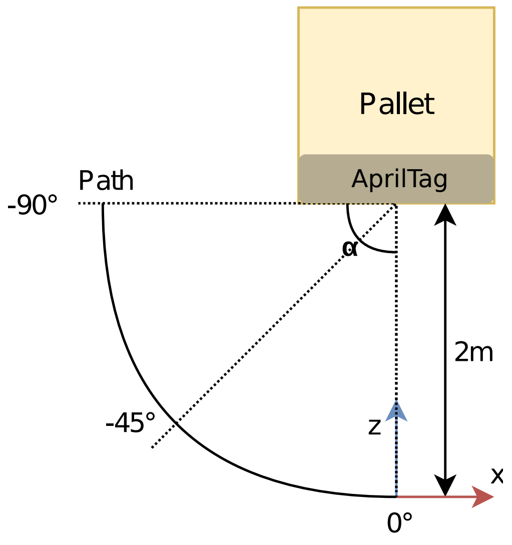
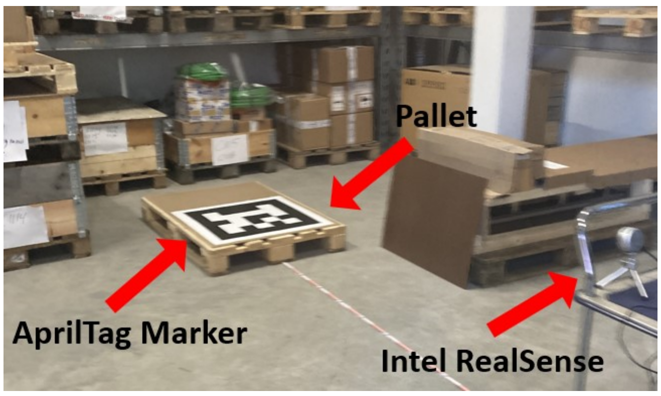
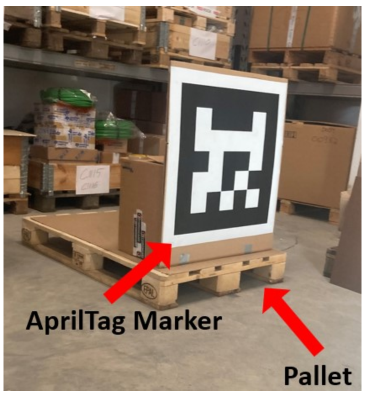


Figure 6.1: Top view of standstill test scenario with both test paths.

In Figure 6.2 the complete setup for stand still tests is seen, where the ground truth AprilTag marker is mounted to the pallet and the Intel RealSense generating the RGB image and point cloud is placed 2 meters from the pallet's location.



(a) Overview of setup including the AprilTag marker, pallet, and Intel RealSense. Note that the AprilTag was standing under the performed test as seen in 6.2b.



(b) AprilTag marker mounted to the pallet.

Figure 6.2: Stand still tests.

6.1.2 Scenario 2: Test of Model when Camera is in Motion

To test the model when the camera is in motion a sample where the camera is moved at a fixed angular velocity, seen in Figure 6.3, from 0° to 90° around the pallet is performed. The test is performed at 2 meters distance from the pallet, and the test construction is the same as for the one in Scenario 1, but this time the samples are continuously generated around the pallet. The purpose of the test is to generate a continuous sample around the pallet and evaluate the results over time as the camera is moved around the pallet. The key parameter in this context is the error between the ground truth AprilTag marker and the estimated pose, with the purpose of evaluating how the errors changes when the camera moves around the pallet.

$$\omega = K\frac{\Delta\theta}{\Delta t} \tag{6.1}$$

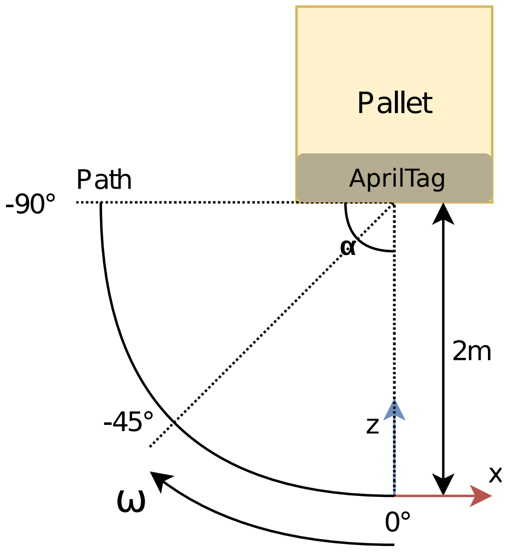


Figure 6.3: Top view of moving test.

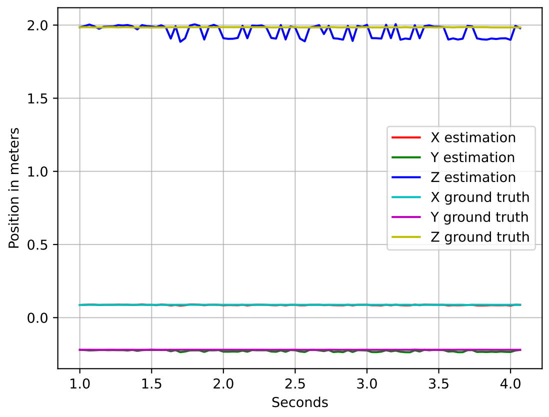
6.2 Experiment Results of Scenario 1

The standstill test is separated into four parts. First, an overview of the results related to the change in position and degree over a 3 seconds sample is shown. The plots show how the developed solution performs parallel to the ground truth. Then, to understand the accuracy of both the developed vector and the ground truth marker, a histogram plot is generated. The plot shows the density results of the two predictions related to one specific angle. For the case seen in Figure 6.4, the results at 0° are shown when the camera stands right in front of the pallet. After that, samples at 14 different angles around the pallet are separated into cases, introduced in Subsection 6.1.1. For each case, both σ representing the standard deviation and μ representing the mean value are the best fit estimated from the histogram. From σ, the spread in results is found, showing the precision of the prediction from the proposed method, and μ, the mean value is extracted, presenting the most confident prediction that will show the proposed method's accuracy. Finally, best and worst cases are highlighted to see at what angles the pose estimation algorithm performs best and worst, and also the precision and accuracy of the predictions at these stages. A summary of all the tests is seen in the following list.

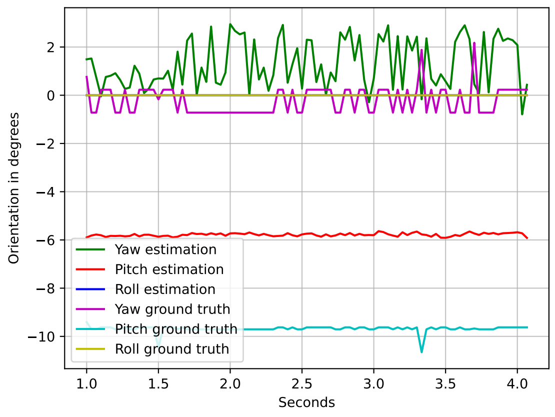
1. Results from the standstill sample at 0°, showing the pose for the 3 seconds sample. Histogram with a density plot showing the accuracy of the predictions at 0° angle for both the pose estimation vector and ground truth marker.
2. Analysis of the σ and μ values for 14 different cases representing different angles sampled around the pallet.
3. Plots highlighting the best and worst case performance based on the μ and σ values best fitted from the histogram.

6.2.1 Time plots for standstill test at 0°

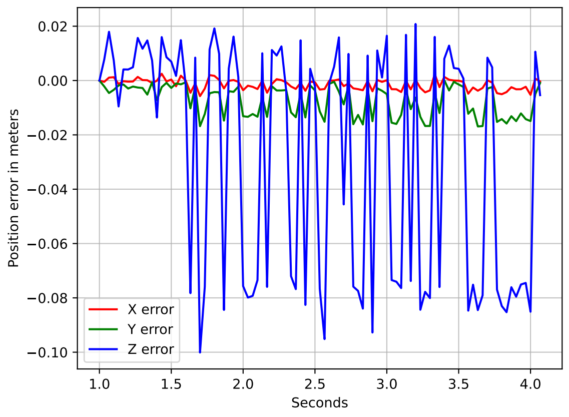
In Figure 6.4, the plot at 0° for both the proposed model and the ground truth model showing the change in position and in degrees over the 3 seconds sample. Note that the first 30 frames is removed before analysis, since there is camera movement when clicking on the record button, so every plot starts after 1 second.



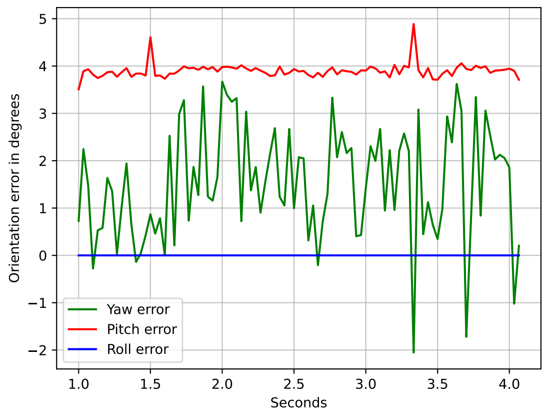
(a) All positions at 0°.



(b) All orientations at 0°.



(c) Position error at 0°.



(d) Orientation error at 0°.

Figure 6.4: Time plots for standstill test at 0°.

A histogram is plotted to determine the estimated method's accuracy and precision and compare it with the ground truth. The plot shows all the values from the 3 seconds sample with predictions in degrees on the x-axis and density on the y-axis. The histogram calculates a set of bins based on data density with a Gaussian distribution fit. The Gaussian distribution is represented by σ and μ, corresponding to the precision and mean value, respectively. The accuracy is the μ value error between μ ground truth and μ estimated value. The system accuracy represents the best and worst cases. Figure 6.5 shows a histogram with a Gaussian distribution fit from the yaw orientation and ground truth at 0°. Similar histograms are plotted for all pose values, attached in Appendix C.

All of the plots is generated by using the Seaborn [79] and SciPy [80] python libraries. The statistical evaluation metrics is calculated using SciPy stats. μ and σ are derived from the `stats.norm.fit(data)`, skew is derived from `stats.skew(data)` and the kurtosis is derived from `stats.kurtosis(data, fisher=True)`. The fisher definition is used as the method subtract 3.0 from the final results to return 0.0 for a Gaussian distribution [81]. The complete plotting script is located in Appendix D as Listing D.19.

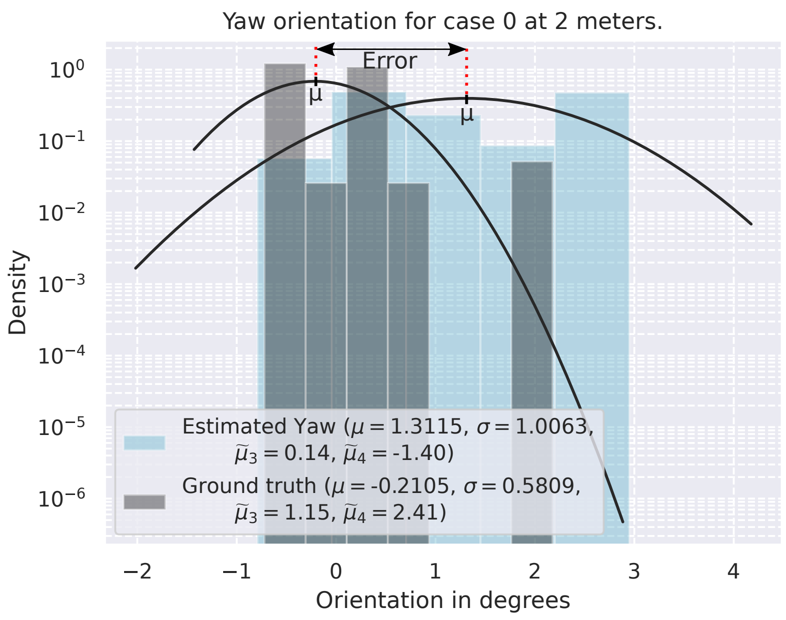


Figure 6.5: Yaw density plot at 0° with an error of 1.52°. Note that the y-axis is a logarithmic scale.

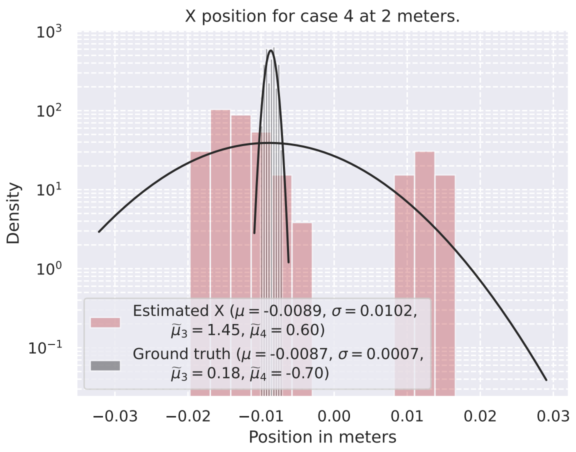
To further evaluate the Gaussian distribution fit, the calculation of the skew and kurtosis tells how symmetric the data is and how exaggerated the values are about the model's fit. As for an example Figure 6.5 the estimated yaw is very symmetric to the model as the skew is only 0.14; however, the kurtosis is -1.40; thus, the model over-represents the data. As for the ground truth, the model has a positive skew, thus skewed right due to the extreme outlier at 2°. The calculated skew and kurtosis indicate that the estimated yaw model represents the data better than the model for the ground truth. All the evaluation metrics is explained in Table 6.2.

Table 6.2: Overview of the statistical evaluation metrics.

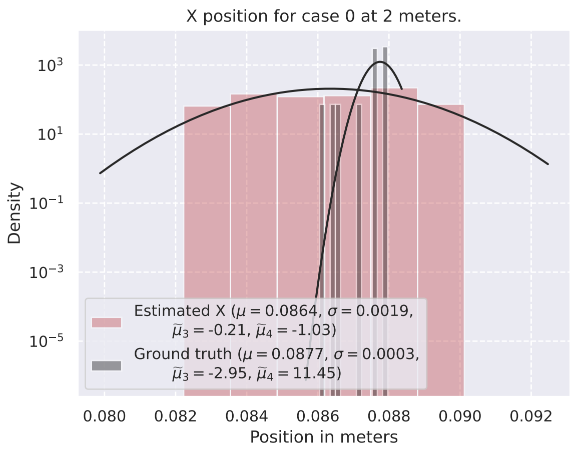
Symbol	**Name**	**Description**
μ	Mu	Mu is the mean, calculated by taking the sum of all data values and dividing it by the number of data values [82].
σ	Sigma	Sigma is the standard deviation of the data, estimated from the histogram plot [82].
$\tilde{\mu}_3$	Skew	Skew indicates how symmetric the data is in comparison to the model. Zero skewness indicates that the data is entirely symmetric to the Gaussian distribution model. A positive skew indicates that the data is skewed to the right, while negative skew indicates that the data is skewed to the left [83]. Skew is unit less.
$\tilde{\mu}_4$	Kurtosis	Kurtosis indicates the propensity of the data to have extreme values compared to the Gaussian distribution model. For symmetric distribution where the skew is zero, a positive kurtosis indicates an excess of the tail, center, or both, and a negative kurtosis indicates a deficiency of the tail, center, or both. A simplification is that kurtosis illustrates the peakedness or flatness of the data bins compared to the Gaussian model [81]. Kurtosis is unit less.

6.2.2 Best and worst case density plots between 0° to -45°:

Based on the tests of the different cases, showed in Table 6.1, the best case and worst case for each of the pose values are displayed. It shows at what angles the proposed solution works the best and the worst within the requirement of up to 45°. The highlighted cases are selected based on the μ error value, which refers to the accuracy of the prediction, or in other words, the difference between estimated value and ground truth. The results are calculated based on the μ from the histogram for each pose value and are shown in Figure 6.6, Figure 6.8, Figure 6.9, Figure 6.10, and Figure 6.11.



(a) X position best case is at $-43.31 \pm 0.06°$ with an error of 0.3 millimeter for case 4.



(b) X position worst case is at $-0.21 \pm 0.03°$ with an error of -1.4 millimeter for case 0.

Figure 6.6: Two density plots for best and worst x positions at the 2 meter standstill test.

Figure 6.6 shows two density plots for the x position. Figure 6.6a) is the best case, while Figure 6.6b) is the worst case. As for the best case in Figure 6.6a) which is case 4 at $-43.307 \pm 0.064°$ from the camera the error between the ground truth and estimated x position is below millimeter measurements. The ground truth is precise within ±0.7 millimeter, and the estimated x position is precise within ±10.2 millimeter. As seen in Figure 6.6a) there are there some extreme outliers for the estimated x position at 0.01 meters which skew the distribution to the right; however, the majority of the bins are at -0.005 meters. The Gaussian distribution fit bins for the ground truth are very symmetric; however, the density has an uneven spread, so the accuracy is the average for all bins.

The worst case is shown in Figure 6.6b) which is the most significant error between the estimated x position and ground truth of -1.4 millimeter to the left of the camera according to Figure 1.2. This is case 0, right before the pallet at $-0.21 \pm 0.03°$. The estimated Gaussian model is a good fit with low skew and kurtosis; the ground truth, however, has four extreme outliers that skew the model to the left. The estimated precision is within ±1.9 millimeter. As for the ground truth, the measured precision is within ±0.3 millimeter, which is similar to the best-case scenario from Figure 6.6a), despite the contrary of the best-case scenario, the left-skewed ground truth bins indicate that the model does not fit as well compared to the best-case scenario.

The object detection algorithm only impacts the x position output since the only frame used is the camera frame. The outliers are, therefore, an error in the object detection algorithm. As shown in Figure 6.7 does it appear a bounding box within the main bounding box which has a center that shifted to the right, which is positive x-direction. This is consistent with Figure 6.6a) as the outliers are 20 millimeters off from the primary bins. The BGR error does also create inaccurate data in the σ case plots.

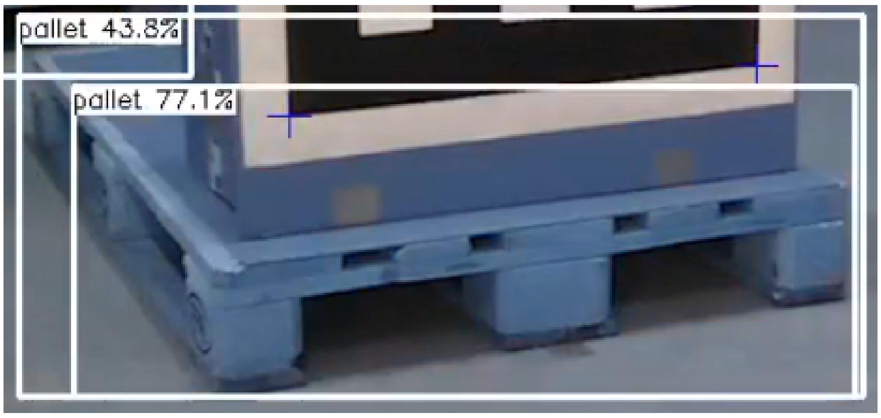
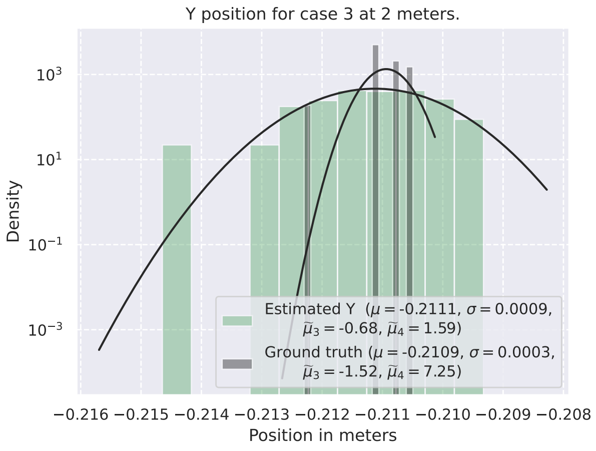
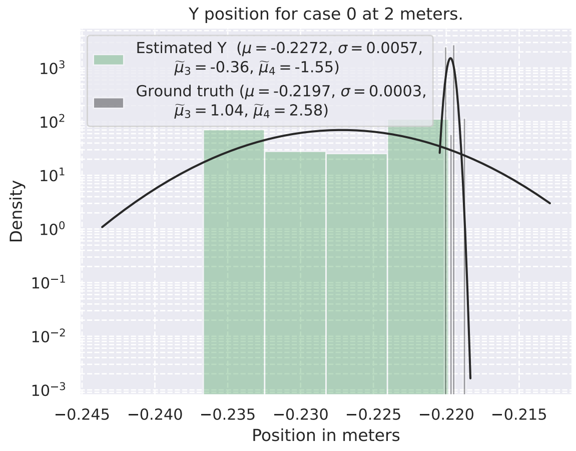


Figure 6.7: Faulty object detection during RGB to BGR bug.



(a) Y position best case is at $-35.42 \pm 0.06°$ with an error of 0.2 millimeter for case 3.



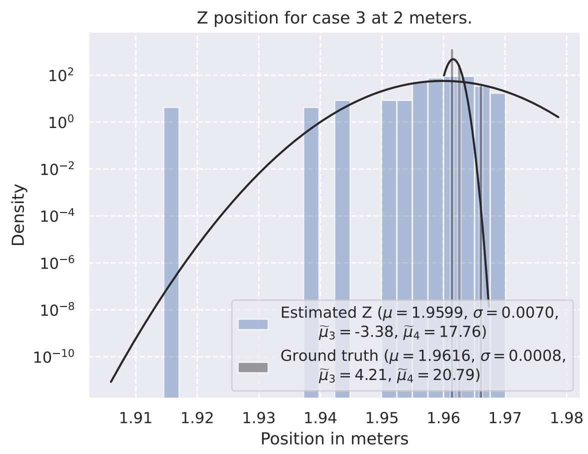
(b) Y position worst case is at $-0.21 \pm 0.03°$ with an error of -7.5 millimeter for case 0.

Figure 6.8: Two density plots for best and worst y positions at the 2 meter standstill test.

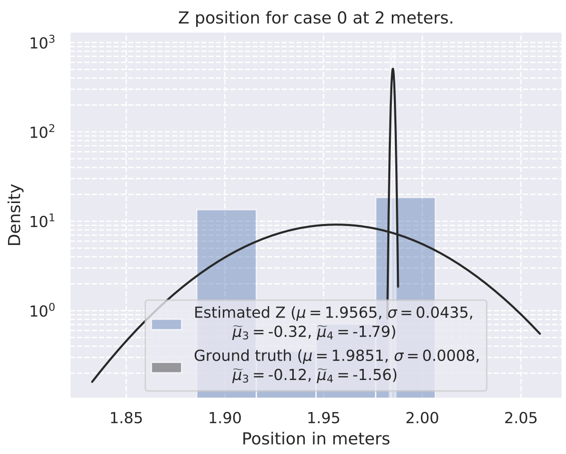
As for the y position, Figure 6.8 shows two density plots of the best and worst case for the y position. The best-case scenario shown in Figure 6.8a) is case 3 at $-35.42 \pm 0.06°$ where the error is 0.2 millimeter. The ground truth is precise within ± 0.3 millimeters, while the estimated y position is within ± 0.9 millimeters. As for how good the Gaussian model is compared to the bins, the estimated Gaussian model has a single extreme outlier below -0.212 meters which skews the model to the left; this is also consistent with the calculated skew of -1.52. From the comparison of the ground truth skewness and kurtosis, does the estimated y model fit better than the ground truth Gaussian model.

For the worst-case scenario, as shown in Figure 6.8b) which is case 0 right in front of the pallet at $0.21 \pm 0.03°$ has the worst error of -7.5 millimeter which makes it detect the pallet a little too high. The estimated model has no extreme outliers and is very symmetric with little to non skew than the estimated Gaussian model. As for the ground truth, the model is skewed to the right. However, with a precision of ± 0.3 millimeters makes it very precise similar to Figure 6.8a). On the contrary, the precision of the estimated model is ± 5.7 millimeters, which is outside of the 7.5 millimeter error.

The object detection algorithm only impacts the y and x position. The two case scenarios in Figure 6.8 are at two different angles, and the precision of the different cases is ± 0.9 millimeters at -35° and ± 5.7 millimeters for the -0.2° positions accordingly. An assumption is that the precision of the y position measurement should degrade. Similar to the x position, does the precision improve. This is due to the bounding box size from the object detection, where as the bounding box gets larger, the x and y position precision gets lower and improves.



(a) Z position best case is at $-35.42 \pm 0.06°$ with an error of 1.6 millimeters for case 3.



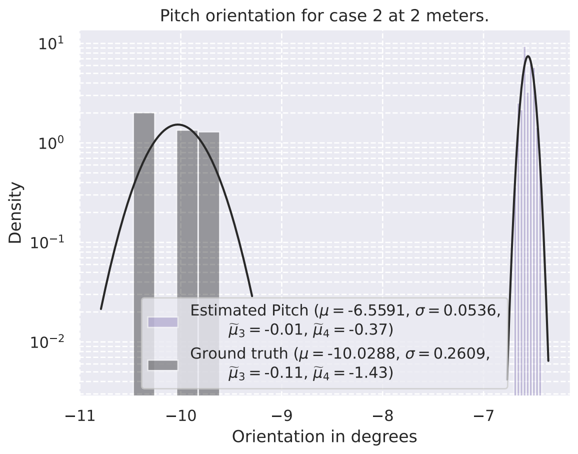
(b) Z position worst case is at $-0.21 \pm 0.03°$ with an error of 28.6 millimeters; the difference between estimated μ and ground truth μ for case 0.

Figure 6.9: Two density plots for best and worst z positions at the 2 meter standstill test.

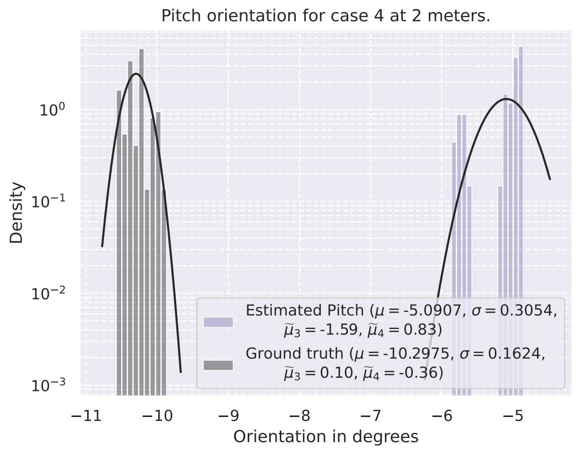
Similar to the x and y position, two density plots of the z position are shown in Figure 6.9. Case 3 has the lowest error of 1.6 millimeters at an angle of -35.42 ± 0.06. The ground truth precision is similar to the prior cases of ± 0.8 millimeters, while the estimated z position precision is ± 7.0 millimeters. As for the fitted Gaussian distribution model, the estimated z model has a single extreme outlier that skews the model to the right. With a skew of -3.38 and kurtosis of 17.76. As for the ground truth model, three outliers skew the model to the right. There is also a very high kurtosis of 20.79, and the bins are not symmetric. Both the skew and kurtosis for both models indicate that the model does not fit well with the data.

As for the worst case in Figure 6.9b), which is case 0 at 0.21 ± 0.03 the error is 28.6 millimeters. The ground truth precision is similar to the other parameters of ± 0.8 millimeters; however, the estimated precision is ± 43.5 millimeters. The estimated z position has two levels where the mean is the average between these two bin peaks as shown in 6.9b, even for that, the model is very symmetric with low skew.

Unlike the x and y positions, the z positions is only correlated by the generated RANSAC plane model from point cloud depth data. This data is much nosier than the camera image; thus, the z position has much lower precision than the x and y positions.



(a) Pitch orientation best case is at $-25.11 \pm 0.05°$ with an error of 3.47° for case 2.



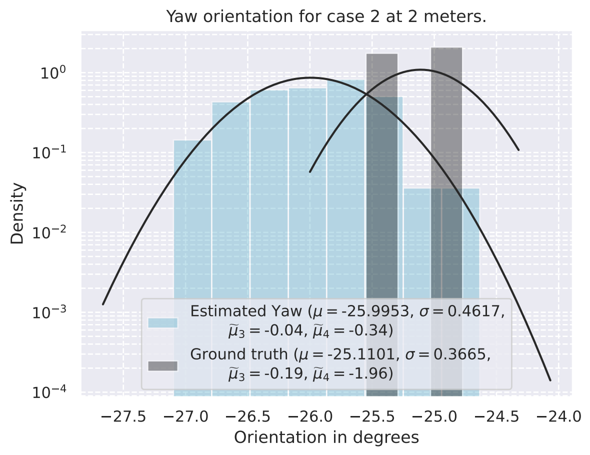
(b) Pitch orientation worst case is at $-43.31 \pm 0.06°$ with an error of 5.21° for case 4.

Figure 6.10: Two density plots for best and worst pitch orientations at the 2 meter standstill test.

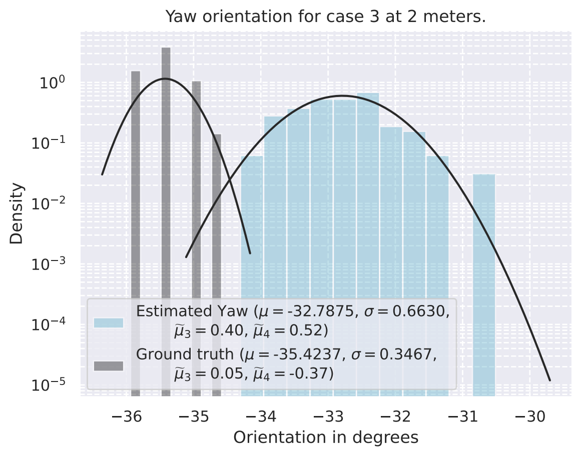
Two density plots for the pitch, shown in Figure 6.10 where case 2 has the lowest error of 3.47° compared to ground truth at a positional angle of $-25.11 \pm 0.05°$ shown in Figure 6.10a). As for the measured angle precision, the ground truth has a precision of $\pm 0.26°$, while the estimated precision is $\pm 0.05°$. As for the Gaussian model fit, both the ground truth and estimated models are very symmetric with low skew. The kurtosis is also very low, indicating that the Gaussian model is a good fit for the data bins.

The second density plot shown in Figure 6.10b) is case 4, which has the highest pitch error of 5.21° at the $43.31 \pm 0.06°$ position. The accuracy for the estimated pitch is within ± 0.31 while the ground truth pitch precision is within $\pm 0.16°$. As for the accuracy of the Gaussian model, the ground truth is very symmetric with low skew and kurtosis. However, the left-skewed estimated model is not accurate. As shown in Figure 6.10b) are there two levels, so the model is the mean in-between these two levels. As mentioned for the best x position case, case 4, the object detection has the RGB to BGR bug illustrated in Figure 6.7; thus, the two levels are from two different point clusters.

A note is that the precision for the estimated model is better than the ground truth. The ground truth uses the AprilTag at a very steep angle, while the pitch uses the floor plane, which does not influence the sensor/pallet angle of attack. The typical two to eight-tonne forklift has a fork length between 1000 - 1220 millimeter [84, 85, 86, 87]. To evaluate the error for a forklift with 1220 millimeter long forks, the best case pitch error of 3.47° will have an 74.0 mm error at the fork tips. In a similar calculation for the worst case at 5.21°, the tips of the forks will have an error of 111.2 mm.



(a) Yaw orientation best case is at $-25.11 \pm 0.05°$ with an error of 0.86° for case 2.



(b) Yaw orientation worst case is at $-35.42 \pm 0.06°$ with an error of 2.64° for case 3.

Figure 6.11: Two density plots for best and worst yaw orientations at the 2 meter standstill test.

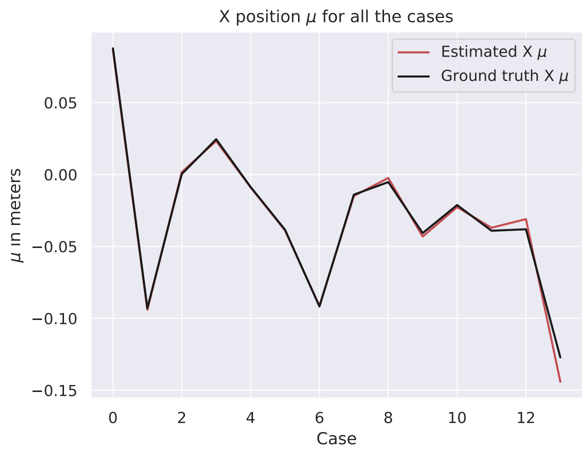
The most exciting orientation unit is the yaw angle. Figure 6.11 shows two density plots of the best and worst case, where the best case has an error of 0.86° as shown in Figure 6.11a) with an angle of attack of $25.11 \pm 0.05°$. The measured precision of ground truth is $\pm 0.37°$, while the estimated yaw has a precision of ± 0.46. The Gaussian model fit for both the ground truth and estimated yaw is a good fit as both are symmetric with low skew, and the kurtosis is close to zero, however, the ground truth has measure points at two levels; thus, the mean is the average of these two bins.

As for the worst-case shown in Figure 6.11b), the error is 2.64° at an angle of attack of -35.4 ± 0.06. The measured precision of the ground truth is within $\pm 0.35°$, while the estimated yaw has a precision of ± 0.66. The approximated Gaussian distribution fit is good for the ground truth and estimated yaw; skew and kurtosis are within ± 0.52.

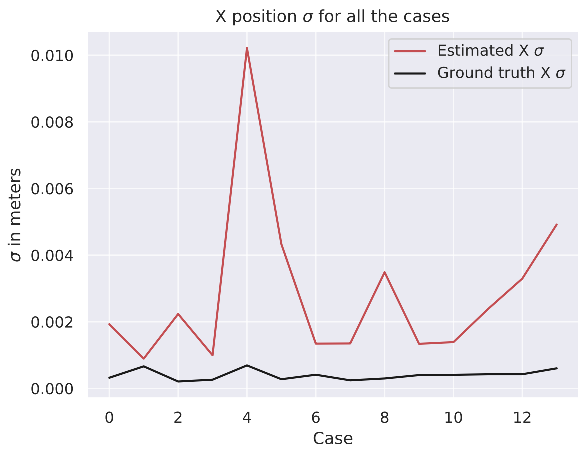
The yaw estimation uses the RANSAC plane estimation of the pallet front to estimate the yaw. There is a correlation between the number of point cloud points used for the RANSAC and the width of the bounding box. The wider the bounding box, the more point cloud points are used for the yaw prediction with points further apart. A similar fork tip error evaluation as the pitch has been done with the yaw. A forklift with 1220 millimeter long forks will have an error of 18.9 mm in the best-case scenario, while the error would be 56.2 mm for the worst-case scenario, all of the calculations are in Appendix B.

6.2.3 Case plots from 0° to -90°:

The density plots presented in Subsection 6.1.1 evaluated the best and worst case within the requirements. The following plots summarize all of the cases including beyond the 45° requirement. The purpose is to evaluate the change between the test cases for all positions and orientations with ground truth. It is important to note that the calibration vector is reset in between all cases which translate the fiducial marker down to the pallet.



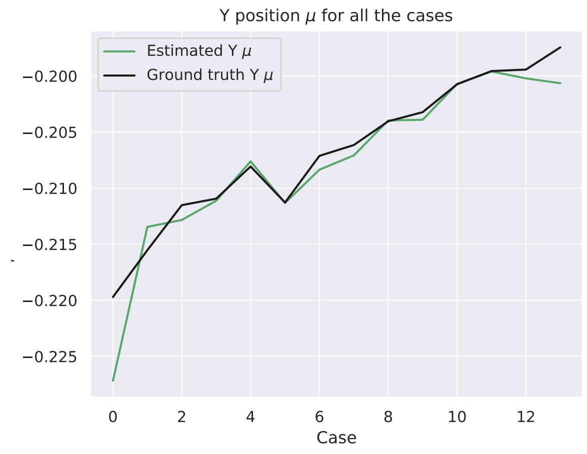
(a) X position μ for all of the cases.



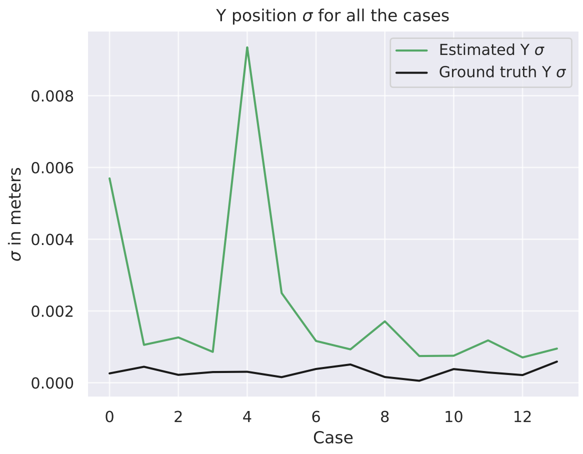
(b) X position σ for all of the cases.

Figure 6.12: X position cases for the 2 meter standstill test.

For the x position, as shown in Figure 6.12a) the pallet is kept right in front of the camera. There is also a good fit between the ground truth and estimated x position until after case 8, where some deviations exist. As for the σ in Figure 6.12b), the ground truth precision is kept within ± 1.0 millimeter, while the estimated x precision is within ± 4.0 millimeter excluding case 4, there is the color glitch in the input image, which distorts some of the pixels, this creates some outliers as shown in Figure 6.6a). In cases 8 and 9, does the pose estimation switch plane, which there is also some extra movement in the camera run, which creates some extra outliers.



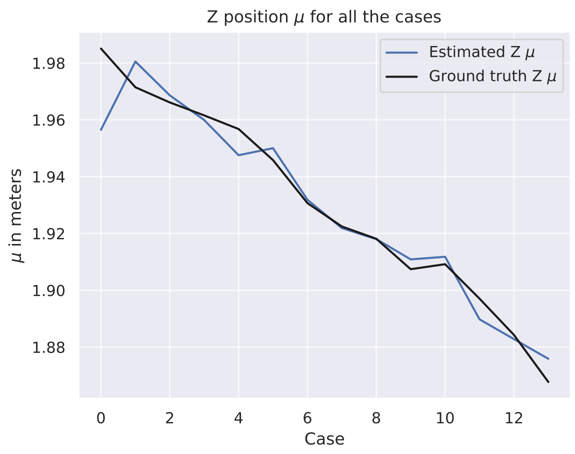
(a) Y position μ for all of the cases.



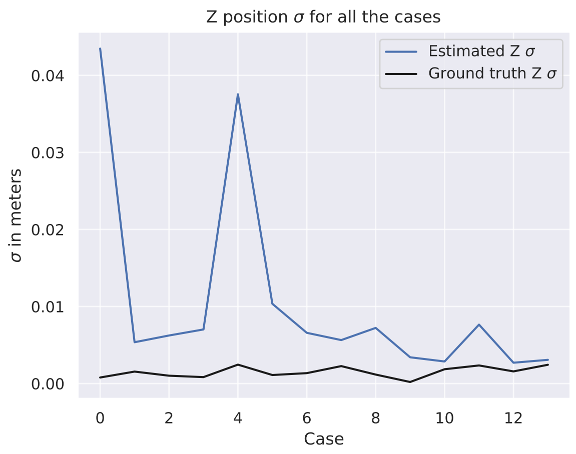
(b) Y position σ for all of the cases.

Figure 6.13: Y position cases for the 2 meter standstill test.

For the y position, as shown in Figure 6.13a), the pallet is moving up in the image frame as the cases progress. There is also a good fit between the ground truth and estimated y position throughout all cases. The precision presented in Figure 6.13b) shows that the ground truth is within ±1.0 millimeter which is similar to the x position, while the estimated y precision is within ±2.0 millimeters excluding case 0 and 4 which have σ peaks at ±6.0 and ±9.0 millimeter respectively. Similarly, as x, the y precision is degraded by the RGB to BGR bug. Overall the precision of x and y is pretty similar.



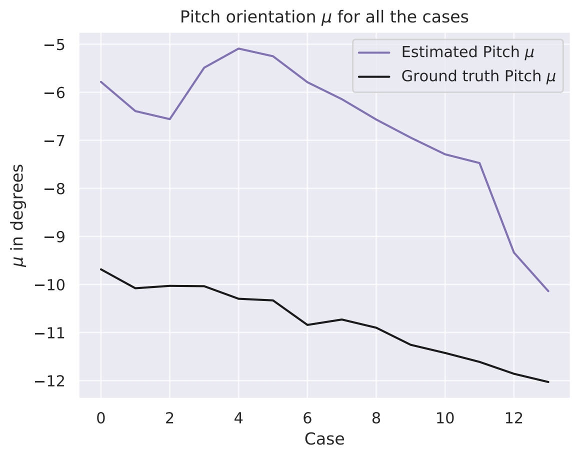
(a) Z position μ for all of the cases.



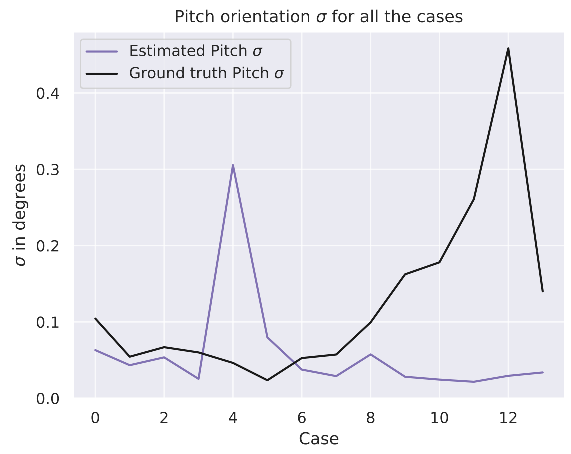
(b) Z position σ for all of the cases.

Figure 6.14: Z position cases for the 2 meter standstill test.

As for the z position, shown in Figure 6.14a), the pallet is moving towards the camera as the cases progress. Similar to x and y, there is a good fit between the ground truth and estimated z position for all cases. Figure 6.14b) present the precision of the z position of the pallet where the ground truth is within ±3.0 millimeters, while the estimated z precision is within ±8.0 millimeters excluding case 0 and 4. The z position suffers similarly to the x and y position during the RGB to BGR bug in case 4, where the σ peaks at ±40.0 millimeters.



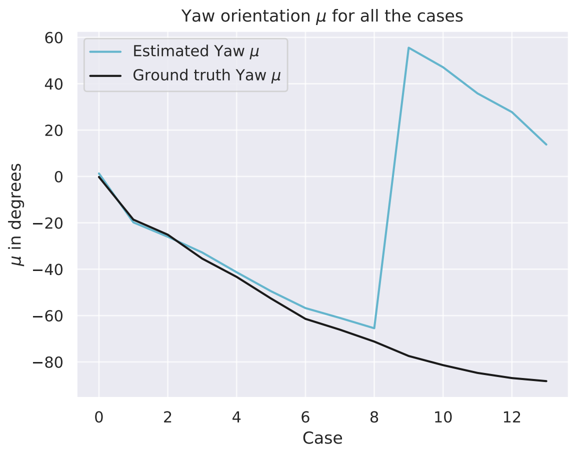
(a) Pitch orientation μ for all of the cases.



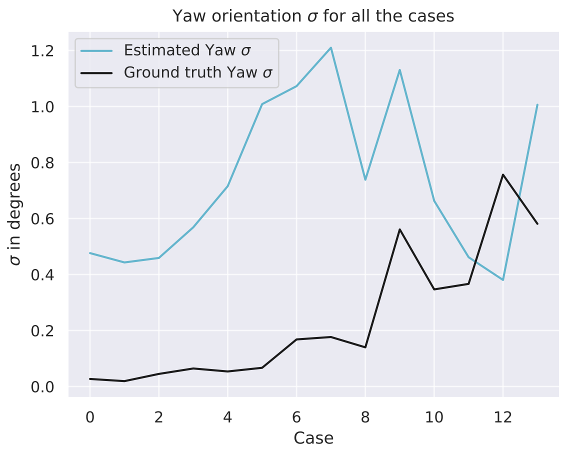
(b) Pitch orientation σ for all of the cases.

Figure 6.15: Pitch orientation cases for the 2 meter standstill test.

The pitch orientation in relation to the pallet decreases as the case number increases. As shown in Figure 6.15a) the pitch delta between case 0 and case 13 is 2.2°. The delta position angle relative to the pallet is -88°. The 2.2° difference might be explained by environmental factors such as floor flatness, cartwheels, and camera tripods. Figure 6.15b) presents the pitch precision. The estimated pitch precision is less than ±0.05°, except for case 4, where the pitch σ peaks at ±0.30°. On the other hand, the ground truth precision decreases exponentially as the cases progress, with a σ peak at ±0.45° in case 12. The angle of attack on the AprilTag increases, narrowing the AprilTag corners together; thus, the pitch sensitivity increases.



(a) Yaw orientation μ for all of the cases.



(b) Yaw orientation σ for all of the cases.

Figure 6.16: Yaw orientation cases for the 2 meter standstill test.

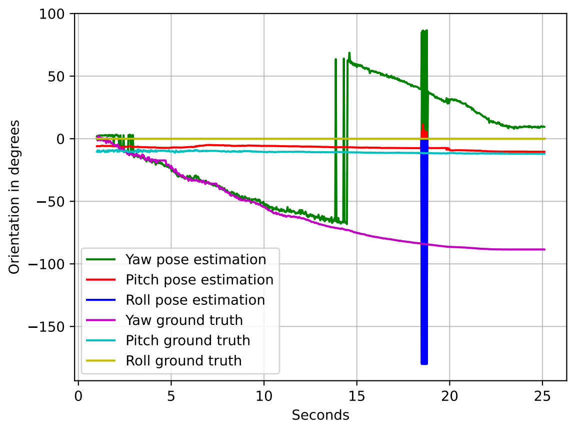
As shown in Figure 6.16a), the yaw angle decreases as the cases progress. The ground truth yaw angle decreases from 0° to -88° over the 13 cases. The estimated yaw angle follows the ground truth until case 9, where the error increases considerably; subsequently, the estimated yaw decreases. The extreme error in case 9 is since the estimated yaw select the pallet side, despite the error between the estimated yaw angle and ground truth should be constant at 90° for the remaining cases. The RANSAC is using points from both the front and side of the pallet, contributing to the error. As seen in Figure 6.16a), the estimated yaw is decreasing rapidly as fewer points are used in the front of the pallet. In the last case 13, the error between the estimated yaw and ground truth is only 102.05°.

The estimated yaw precision is shown in Figure 6.16b). The ground truth σ starts below ±0.1°. However, precision decreases as cases progress. The highest σ for the ground truth is ±0.8° in case 12. As for the estimated yaw precision, does the precision degrade similarly to ground truth with an overall lower precision with a peak σ of ±1.2°. However, after case 7, the precision increases. The precision increases since the selected pallet side has changed, and as the remaining cases progress, the angle of attack decreases.

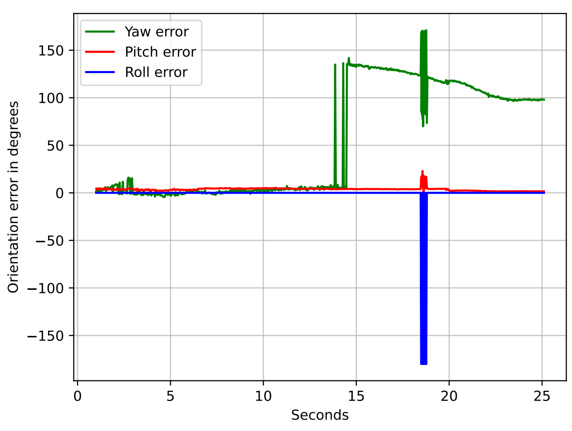
6.3 Experiment Results of Scenario 2

The next scenario is the moving test, where the camera is moved at a continuous velocity around the pallet. The moving test is performed at 2 meters distance going from 0° to -90°. To evaluate the performance of the moving test two different curves are plotted showing the orientation and position of the pallet and AprilTag marker as the camera is moving.

The first plot is shown in Figure 6.17a), where the changes in orientation over time is displayed. The essential value in this case is the yaw value for both the pose estimation and ground truth as it reflects the angular movement around the y-axis. As the camera moves, there is a drift on the yaw axis between the ground truth and our angle. After 13.77 seconds at a yaw angle of -68.85°, there is a change in the vector yaw angle, which last for 0.64 seconds and 4.13° when the pose estimation transition from the front plane to the side plane of the pallet. Further, as time progresses, the vector yaw starts at 30.56°. It turns with an angular velocity of -2.90° per second between 14.40 seconds and 18.10 seconds. The ground truth moves with an angular velocity of -5.67° per second in the same period. As also seen in Figure 6.17a), the ground truth marker is moving from approximately 0° to approximately -90°, which corresponds well to the sampled moving sequence. The same counts for the estimated model, but at approximately -60° in spikes to positive 50°. The reasons for this is that the object detection algorithm detects the side plane of the pallet and not the front plane. This is one of the limitations of the estimation model and is further discussed in Chapter 7.



(a) All orientations.

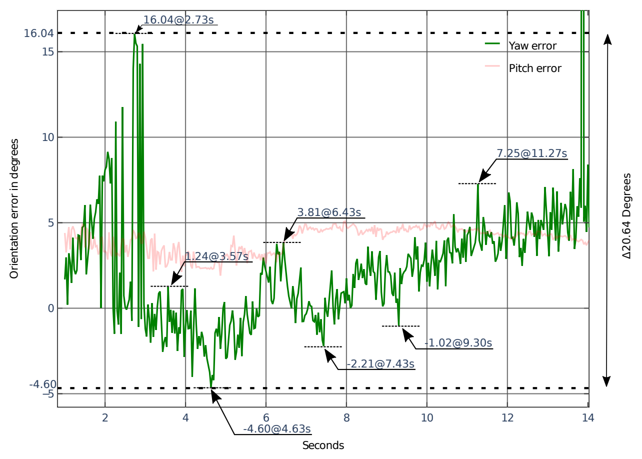


(b) Orientation error.

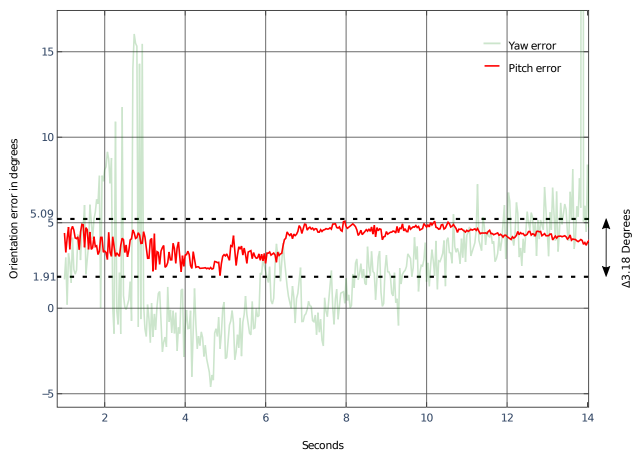
Figure 6.17: Orientation and orientation error for the 2 meter moving test.

The second plot is displayed in Figure 6.17b), where the error between the estimated orientation and the ground truth value is calculated. The plot shows that the estimated pitch value is following the ground accurately. The same counts for the yaw value until it spikes because of the same reason as presented in the previous paragraph, where the object detection algorithm detects the side plane of the pallet. The yaw angle is within 10° for the first 6 seconds and then increases linearly with an angular velocity of 4.06° per second between 7.00

and 13.73 seconds. Another thing to note is that after 18 seconds, where a sudden spike is seen. The spike is due to the RGB to BGR bug where the object detection algorithm selects another pallet. This also displays a limitation of the developed pose estimation solution. However, are the overall results from the sampling sequence satisfying the project's defined requirements.



(a) Extremes and delta for the yaw orientation error.



(b) Extremes and delta for the pitch orientation error.

Figure 6.18: Orientation error between 1 and 14 seconds from 6.17b.

To evaluate the orientation time plots further an extracted section of the orientation error is shown in Figure 6.18. This is the same orientation error data as shown in Figure 6.17b), however this is the period between 1 and 14 seconds. The yaw orientation error in Figure 6.18a) shows that in the initial 2.73 seconds, the estimated yaw is does not update, and the error is increasing linearly. After 2.73 seconds, the error peaks at 16.04°, where the yaw ground truth is -13.28° relative to the pallet. The error drops and is kept within -4.60 and 3.81° error until 9.30 seconds, where the error trend is overall increasing. All of the key error points is listed in Table 6.3.

Figure 6.18b) shows the pitch error, during the same period. The pitch error is kept within 1.91 and 5.09° offset during the whole period, resulting in a 3.18° delta between the maximum and minimum pitch error.

For the time points at 4.63 and 7.43 seconds in Table 6.3, has the error been caused by not detecting the AprilTag. As shown in Figure 6.19, the ground truth is not updated and is using the prior measurements. The screen recording verifies this during system testing.

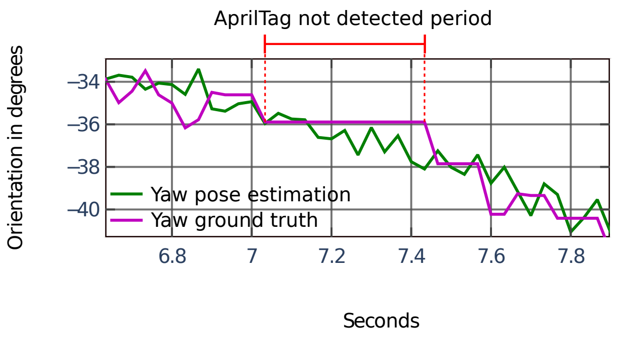
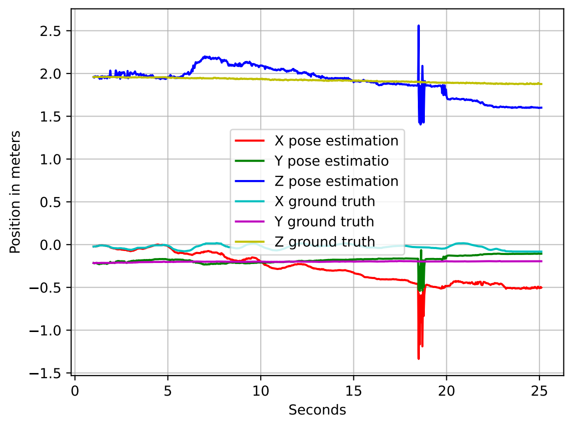


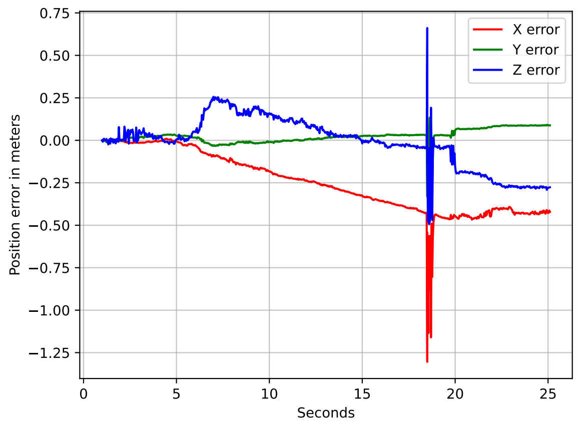
Figure 6.19: Period where the AprilTag is not detected, verified by screen recording.

Table 6.3: Key error points for Figure 6.18a). Table 6.4 lists all mentioned key points together with added yaw ground truth and AprilTag detection status for the respective time point. This evaluates the current yaw location at the time point and if the key error point is valid. If the AprilTag is not detected, the error spike is not valid.

Time point in seconds	Orientation error in degrees	Yaw ground truth in degrees	AprilTag detected
2.73	16.04	-13.28	Yes
3.57	1.24	-15.44	Yes
4.63	-4.60	-21.96	No
6.43	3.81	-34.01	Yes
7.43	-2.21	-35.89	No
9.30	-1.02	-50.39	Yes
11.27	7.25	-61.86	Yes



(a) All positions.



(b) Position error.

Figure 6.20: Positions and position error for the 2 meter moving test.

Similar to the moving test performed for the estimated orientation and ground truth, Figure 6.20 shows the translation of the pallet as the camera is moving around it. Figure 6.20a) shows all six translations for x, y and z for both the estimated position and ground truth, while Figure 6.20b) shows the error between the estimated position and ground truth. The camera initially pointing a little above, and about 2 meter from the pallet. This corresponds well with Figure 6.20a).

As for the error shown in Figure 6.20b), it is within 81 millimeters for the initial 6 seconds, after which the error for the x and z translations start to drift, while the y error continues ±30.0 millimeter. The drift starts after 6 seconds. The ground truth yaw is -31.64°, where the z error increases and x decreases; thus, the estimated pallet point moves away and to the right, respectively, related to the camera frame. After 7.1 seconds, both the x and z errors decrease linearly at the same rate, indicating that the ground truth and estimated correlate as the cart is interacted by hand with uneven force. The drift is due to the bounding box detecting more of the pallet side. Subsection 7.3.1 discuss a solution for the drift.

Finally, after 18 seconds, there is a momentary spike. This spike is due to a similar the RGB to BGR bug, in Figure 6.7 on page 58.

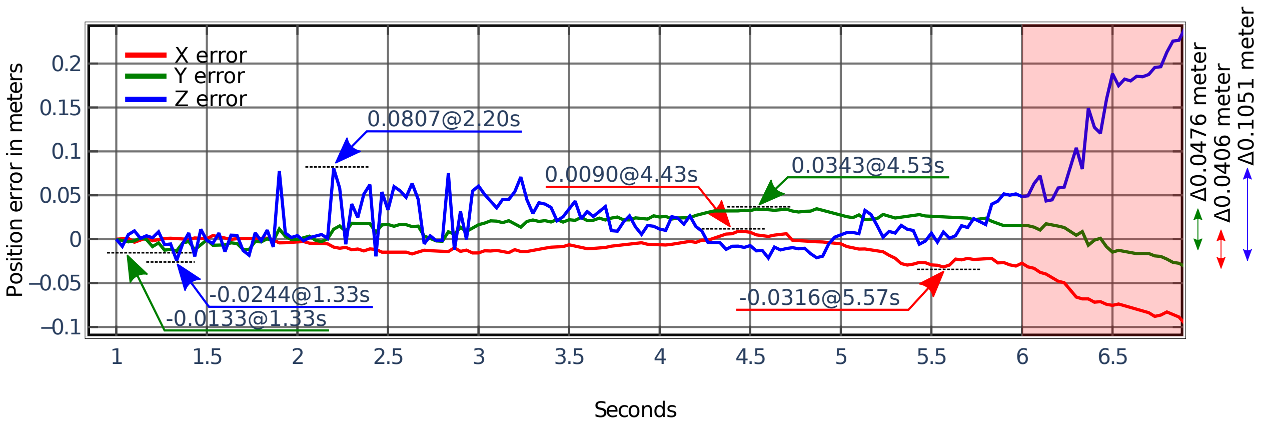


Figure 6.21: Position error in the initial period between 1 and 6 seconds from 6.20b in the 2 meter moving test with maximum and minimum key points.

Further evaluation of the position error of the 2 meter moving test is presented in Figure 6.21, which is an extracted version of the data in Figure 6.20b) between 1 and 6 seconds before the translation drift. The key points for all position parameters are the maximum and minimum captured values. In addition, is the delta between the maximum and minimum calculated.

The key maximum points for x and y error is not valid as the time points at 4.43 and 4.53 of the AprilTag is not detected, similar to Figure 6.19 between 7.0 and 7.4 second is Figure 6.22 in the 4.2 to 4.7 period where the ground truth is not updated and is using the prior measurements.

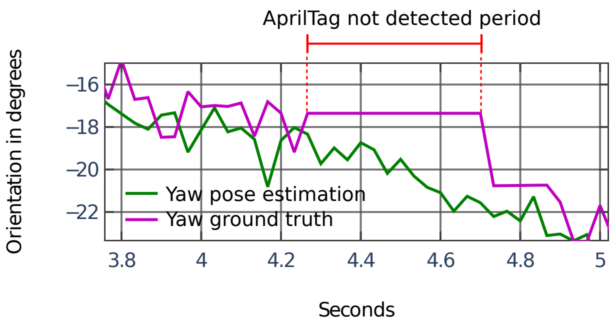


Figure 6.22: Period where the AprilTag is not detected, verified by screen recording.

Table 6.4: Key error points for Figure 6.21 with added yaw ground truth and AprilTag status. Similar error validation applies here as for Table 6.3.

Time point in seconds	Position error in millimeter	Axis	Yaw ground truth in degrees	AprilTag detected
1.33	-13.3	y	-1.61	Yes
1.33	-24.4	z	-1.61	Yes
2.20	80.7	z	-8.17	Yes
4.43	9.0	x	-17.35	No
4.53	34.3	y	-17.35	No
5.57	-31.6	x	-29.28	Yes

6.4 Chapter Summary

Chapter 6 presents two different experiment scenarios. Scenario one is the standstill test, where 14 cases between 0° and 90° at a two-meter distance from the pallet are evaluated. Out of the 14 test cases, the best and worst case for each DOF is based on the μ error between the estimated pose vector and ground truth. The following two paragraphs summarize the best and worst-case results from scenario one and scenario two.

The standstill position error shows that the best case error for the x and y position is within 0.3 millimeters, and the worst case is within 7.5 millimeters. For the z position, the best case is 1.6 millimeters, and the worst case is 28.6 millimeters. The pitch orientation has a best-case error of 3.65°, while the worst is 5.21°. On the other hand, the yaw angle has a best case of 0.86° and a worst case of 2.64°. Issues during the standstill test included the RGB to BGR bug, which interfered with the YOLOX-S object detection algorithm, creating frames with false x and y positions.

Scenario two is the moving test that takes a continuous capture of the same path as scenario one over 25 seconds. All positions are stable until 6 seconds, when they start to drift at a ground truth yaw angle of -31.64°. During the initial 6 seconds, the x and y positions are within 34.3 millimeters, while the z position is within 80.7 millimeters. As for the pitch orientation, the error is within 5.09°, while the yaw orientation is within 16.04°. Issues during the moving test are that the AprilTag was not detected while moving, resulting in false error spikes. Similar to scenario one, the RGB to BGR bug occurred at around 18 seconds.

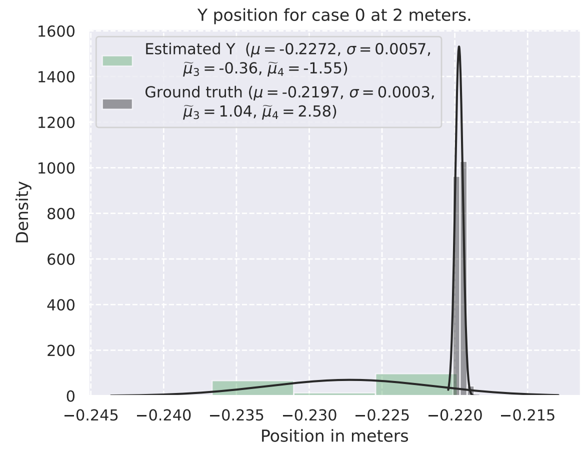
Chapter 7 will discuss the results further with suggested solutions and identified sources of errors.

7. Further Discussions

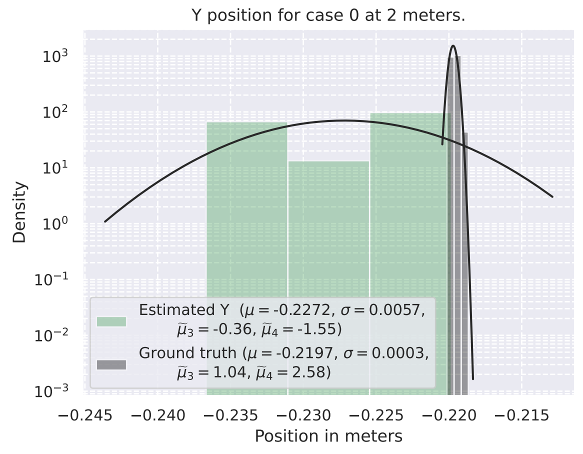
Through the development of the system solution a few efforts have been made, but the results have not been included in the final design. These efforts are presented in this chapter, followed by some of the experiences from the testing and experimenting phase and challenges faced through the system design and solution development journey.

7.1 Density Estimation and Data Representation

To get a representation of the generated standstill results, a density histogram has been generated together with a Gaussian distribution fit. By representing the bins as density instead of frequency, enables comparison of data with different sample sizes [88][89, p.10].



(a) Density plot for the case 0 Y position with a linear y axis scale using 3 bins.



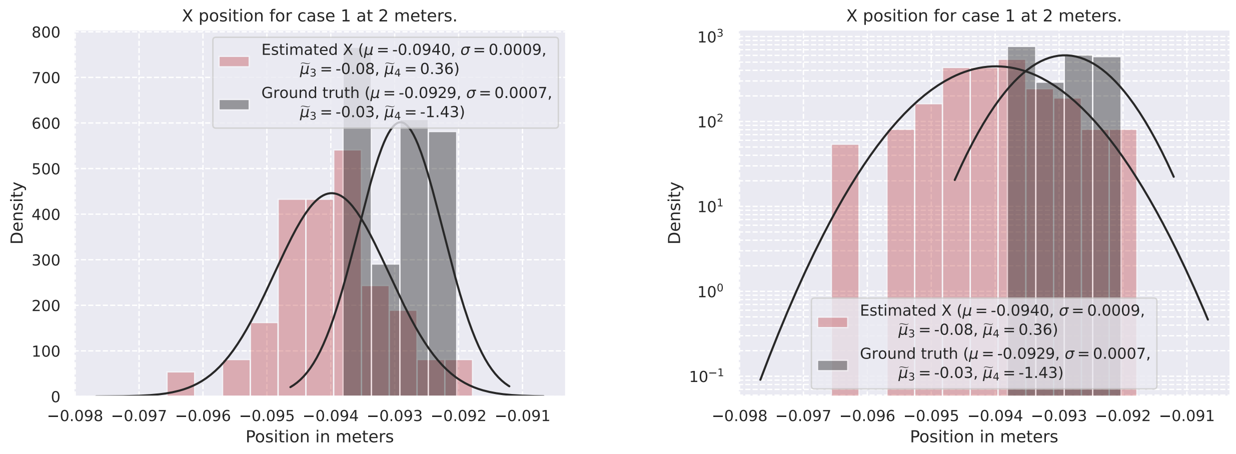
(b) Density plot for the case 0 Y position with logarithmic y axis scale using 3 bins.

Figure 7.1: Two equal density plots with a Gaussian distribution fit, the only difference being the y-axis scale, where Figure 7.1a) has a linear y-axis scale versus Figure 7.1b) that has a logarithmic y-axis scale.

Figure 7.1 shows two density plots with a Gaussian distribution fit. As it is hard to compare the data from the pose vector and ground truth solution, the y-axis in Figure 7.1a) has been converted to the logarithmic scale to get the data more comparable against each other, as seen in Figure 7.1b). Another attempted measure has been adjusting the number of bins to make the data more equally distributed, but with no major effect. Seen from Figure 7.1b) Furthermore, in a few other plots, the fit with the converted y-axis does not directly represent a Gaussian distribution, although the data is much easier to compare.

Consequently, the logarithmic manipulated Gaussian representation has shown to vary for different samples as it does not sufficiently capture all the needed data attributes for all cases.

On the other hand, the Gaussian distribution fit, with logarithmic y-axis scale, has proven to be suitable for some cases. As shown in Figure 7.2, the histogram in Figure 7.2a) represents the data being well represented with a good Gaussian distribution fit. Figure 7.2b) is the same plot as Figure 7.2a) the only difference is that the y-axis is logarithmic scale, it is possible to evaluate the outliers for both plots.



(a) Density plot for the case 1 X position with a linear y axis scale.

(b) Density plot for the case 1 X position with logarithmic y axis scale.

Figure 7.2: Two equal density plots with a good Gaussian distribution fit, the only difference being the y-axis scale. 7.2a has a linear y-axis scale, while 7.2b has a logarithmic y-axis scale.

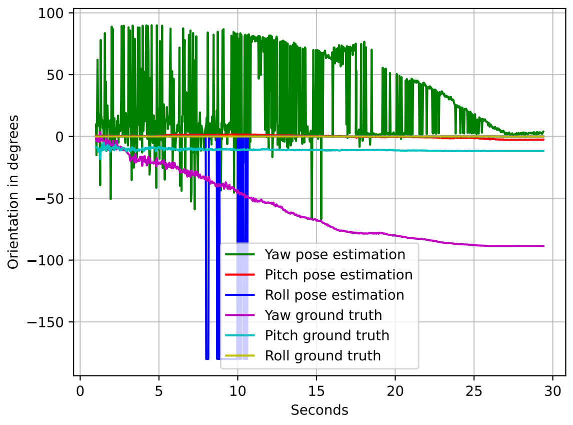
However, as mentioned in Subsection 6.2.1, the statistical evaluation metrics are directly calculated using functions from the SciPy [80] python library. Thus, the calculation of σ, μ, skew and kurtosis is not affected by the data's representation as they are found by separate functions. Histograms continue to be a proper tool for displaying data, but at the same time it is worth considering other available density estimates in order to illustrate the obtained data in a more presentable and easily perceivable way for all cases [89, p.11].

7.2 Additional Experiments

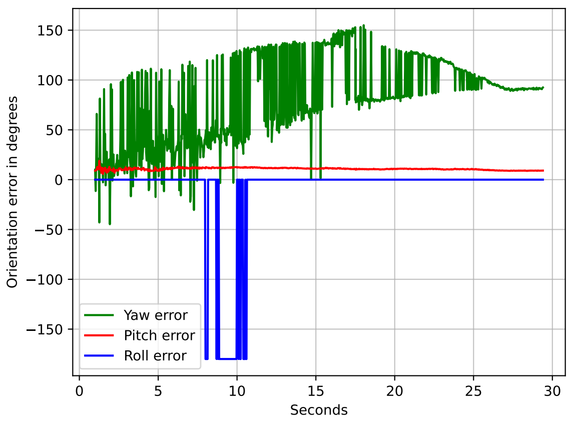
For the project's expectations presented in Section 3.2 two additional experiments have been executed. They include tests in a real-world scenario and a test of the algorithm at 3-meters distance from the pallet. A brief introduction to the two scenarios is presented in the following subsections, followed by comments on the system design's performance for the two scenarios.

7.2.1 Moving Test at 3 Meters

The system design has also been tested in motion at 3 meters distance, similar to the test scenario presented in Subsection 6.1.2. The purpose of the test is to evaluate the system design's performance at longer distances, which will be important before the final industrialized system design is integrated into real-life applications. The test results are seen in Figure 7.3 and Figure 7.4.



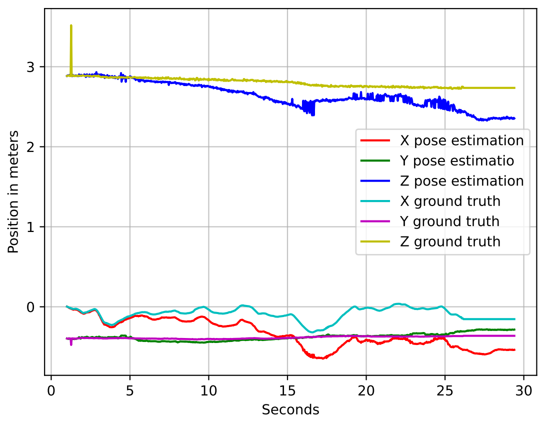
(a) All orientations.



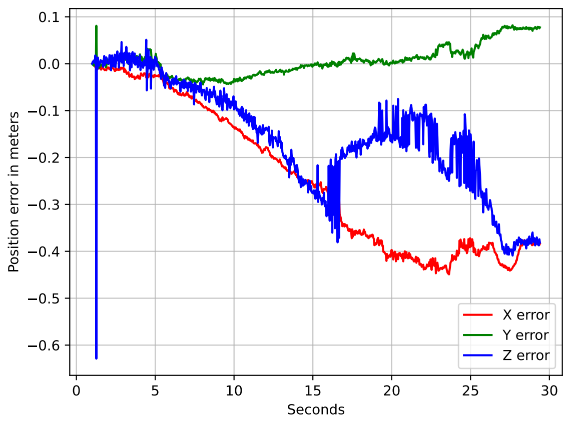
(b) Orientation error.

Figure 7.3: Orientation and orientation error for the 3 meter moving test.

The orientation as shown in Figure 7.3a) the yaw pose estimated is susceptible to severe noise. No points are used as the ground plane, and the expected floor plane is set to zero. However, at 15 seconds or 75 yaw ground truth, the estimated yaw managed to estimate the angle within reasonable error.



(a) All positions.



(b) Position error.

Figure 7.4: Positions and position error for the 3 meter moving test.

As for the translation, as shown in Figure 7.4a) the pallet is translated 3 meter away from the camera. The X position is following a similar trajectory as in Figure 6.20a). For the translation error shown in Figure 7.4b) does the x and z diverge from zero at a similar rate; the z error contains more noise than the X error. On the other hand, the y error decreases and then increases, crossing 0 after 15 seconds or 75° for yaw ground truth.

7.2.2 On-Site Prototype Experiment

To test the performance of the object detection and pose estimation model in a real-life context it was also tested in the environment where it is supposed to be operated when the final product is developed. Gathering experience from a real-life test scenario is important for adapting the developed model to a real-life use case, as also desired from Red Rock as introduced in Section 3.2. The operation where the model has been tested consists of the following steps: positioning for lifting the pallet, lifting the pallet, and unloading the pallet. The operation is repeated twice and is performed by an operator doing all the tasks manually. An image of the lifting phase is seen in Figure 7.5. The purpose of the test was to gather experiences related to the performance of the algorithm in a real-world context.

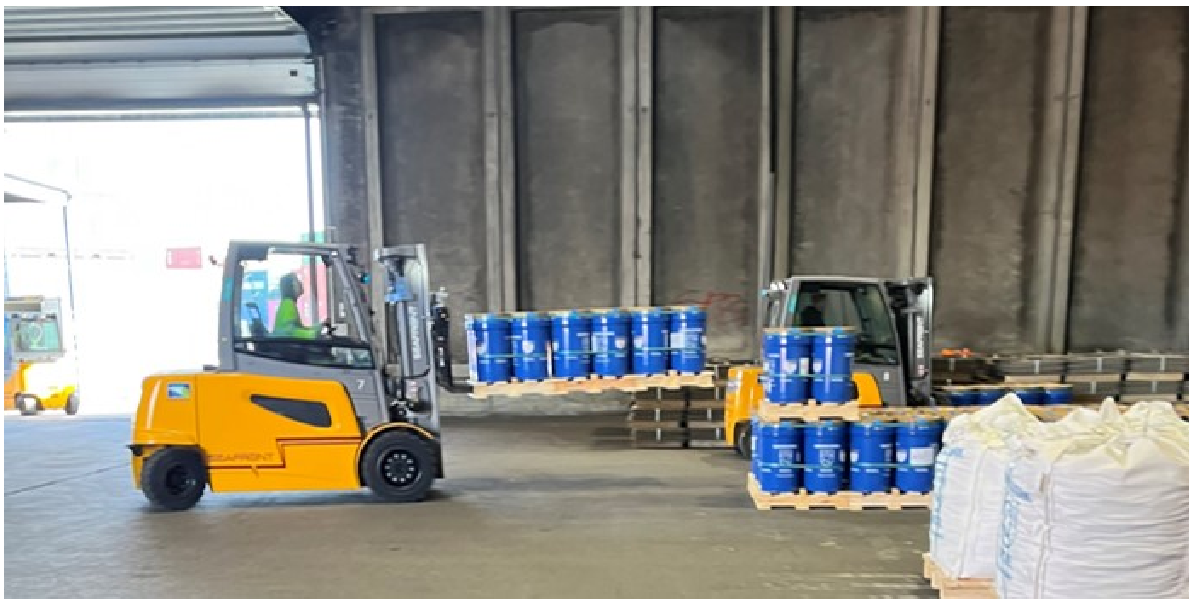
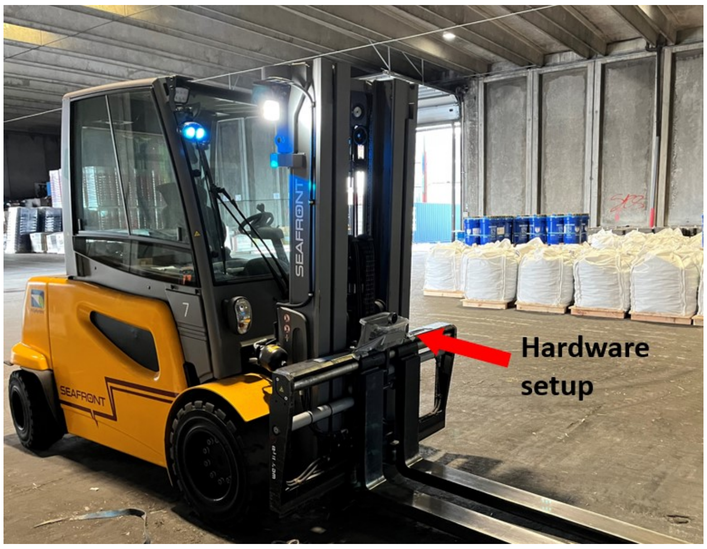
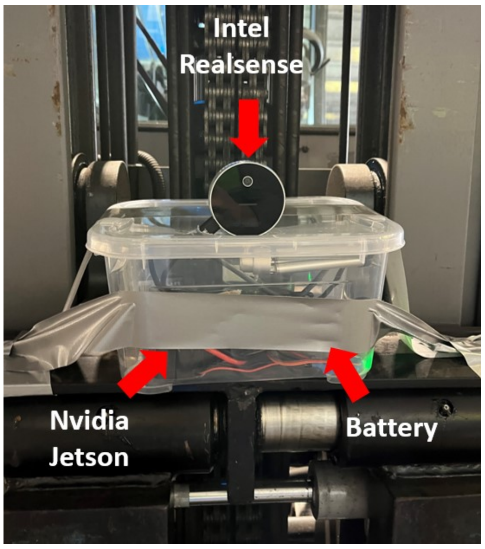


Figure 7.5: Real-life forklift operation used for testing.

To perform the test, a setup of hardware components Was mounted to the forklift's forks. To achieve this, the Intel Realsense RGB-D camera was connected to a Nvidia Jetson Nano SoM and a 81.4 Wh LiPo battery. The data generated from the Intel Realsense was stored locally on the Jetson, and after sampling transferred to an external PC. The three components were placed in a box that is in sufficient size to not disturb the sight of the forklift operator. An image of the complete setup is seen in Figure 7.6b). The complete setup was then mounted to the mask of the forklift, seen in Figure 7.6a), with the purpose of generating a front view from the perspective of the forklift.



(a) Overview of hardware setup mounted to forklift.



(b) Front view of mounted hardware setup.

Figure 7.6: Overview of hardware components setup.

One of the challenges faced when implementing the solutions in real life was mounting an AprilTag marker to the pallet as it would disturb the general operations. Consequently, there was no suitable method of evaluating the solution's performance in terms of accuracy, precision, and robustness. Another challenge was that multiple different pallets were used in the real-life environment. These come in different shapes and forms and are not consistent. With the current model trained on foremost euro pallets, a more generalized data set would be needed to make the model suitable for the real-life application and enable it to detect various pallet types.

The Jetson was connected remotely to a laptop through a mobile IEEE 802.11ac WiFi hot spot. RealSense Viewer is installed on the Jetson, and X11 forwarding over ssh was enabled to run GUI applications remotely, which enables to view a live feed and start and stop recording. The Ros-bag recording from the RealSense Viewer was stored locally. However, the recording paused when the connection between the laptop and Jetson was lost. When the connection was regained, did the recording continue. The final recorded Ros-bag was possible to play but with corrupted sequences.

7.3 Further Challenges for Object Detection

The first part is related to some of the findings when developing the object detection phase. To be enable the forklift two lift the pallet an exact understanding of where the pallet holes are would be needed. The current design is finding an approximated center position of the pallet, but not the exact pallet hole position. A design to achieve this has been developed and tested with results and is described in the following subsection.

As explained in Section 3.2, one of the expectations behind the project was to test the algorithm beyond the requirement of 45°. In the following subsection also some of the identified limitations of the system design beyond 45° are discussed.

7.3.1 Detecting Pallet Holes

The current assumption for identifying the pallet holes is that as long as the developed solution finds the center of the pallet's front plane and the pallet holes are located in a fixed position with an equal distance from the center, the forklift would only need the center location to correctly position its forks. On the other hand, as a part of the initial system design, detecting the pallet holes where the pallet will insert the forks to lift the pallet, was also included in the end-to-end pipeline. Due to a few challenges this part has rather between included in the discussion section, presented along with the obstacles followed to achieve the task and some of the identified opportunities to actually achieve pallet hole detection.

Proposed Pallet Hole Detection Model

For detection of pallet holes, the same YOLOX-S model as for the one detecting pallets is used, but additionally, transfer learning is used to train the network on labeled pallet holes, as seen by the red boxes in Figure 7.7. Accordingly, the same LOCO dataset is pre-labeled with pallets, however, to detect the pallet holes, some manual labeling is required to find the two 3D points where the forks can be inserted. The pallet holes are labeled across the largest diagonal of the hole, and only pallets that are directly in contact with the floor are labeled. The bounding box will overlap both on the floor and onto the pallet, but since only the center of the bounding box is relevant does the overlapping have no concern.

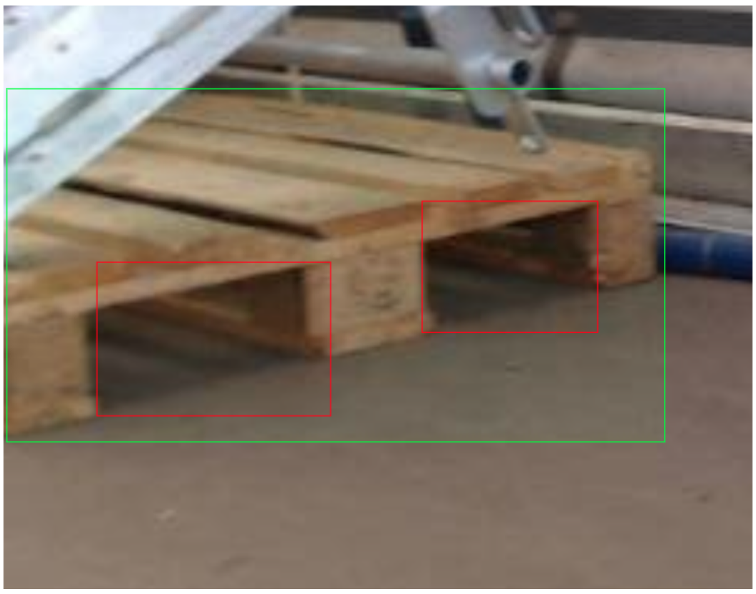
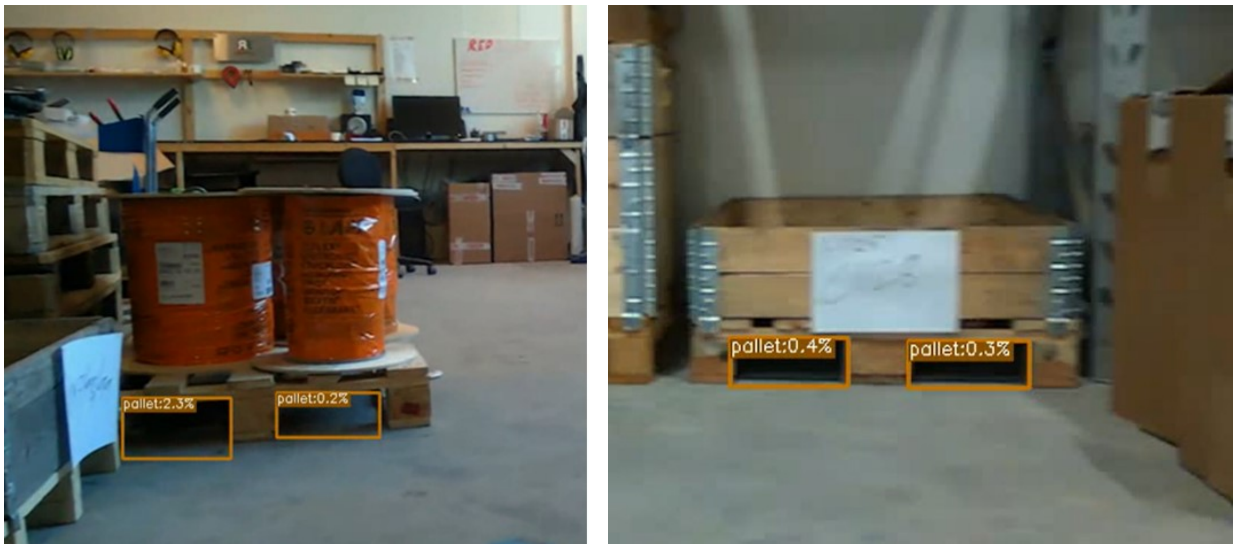


Figure 7.7: Multiple objects including both pallet and pallet holes located in one grid cell.

A total of 1900 pallet holes annotations have been manually annotated, refereed to as voids, split into 1261 training and 639 validation annotations for training. The ML algorithm manage to detect pallet holes, however with a very low confidence bellow 1%, as shown in Figure 7.8. CVAT support auto labeling with OpenVINO [90] that speed up the labeling process to make the model more confident. Since the pallet void dataset only contain 1900 pallet void annotations compared to 120 000 pallet annotations in the complete LOCO dataset [34], is it recommended by [59] to do more augmentation during training to improve the model.

The training parameters is located in Table A.2, and all of the training results; loss curve, average precision and average recall is presented in Appendix A as Figure A.4, Figure A.5 and Figure A.6, respectively.



(a) Inference evaluation of test image 1. (b) Inference evaluation of test image 2.

Figure 7.8: Inference of the YOLOX-s pallet void detection model.

7.3.2 Multiple Objects within the same bounding box

One of the challenges faced when trying to detect pallet holes is found in the YOLOX object detection algorithm when multiple objects are located in the same cell grid. This issue was experienced when trying to detect both the pallet and the pallet holes within the same iteration shown in Figure 7.7, as YOLOX was only capable of detecting one of them. This is a known issue in YOLOX and is one of the limitations of the ML model [52]. A potential solution for this issue is to separate the detection of multiple objects that are located closely together into two different problems and have one model for each. Another solution is to evaluate other ML models such as Mask R-CNN or SSD that are using different techniques. The R-CNN model could be an option as it is capable of both object classification and instance segmentation, where the object's can be identified on pixel level and separated into semantic parts, also proposed in [31]. This solution has been considered, but due to Mask R-CNN's AP and FPS performance, which is not as god as YOLOX, it was not taken into further consideration.

7.3.3 Switching Planes

Another challenge related to plane detection is switching planes. As the RANSAC algorithm is based on finding the largest vertical plane based out of the point cloud, issues, when the camera is moving from 0° to 90° under the moving test, are identified. One of the issues happens for the yaw rotation at approximately -70° with a transition angle of 1.6°, where the algorithm detects the side plane of the pallet as the largest vertical plane, seen in Figure 7.9. This makes the calculated vector find the pose of the side plane instead of the front plane.

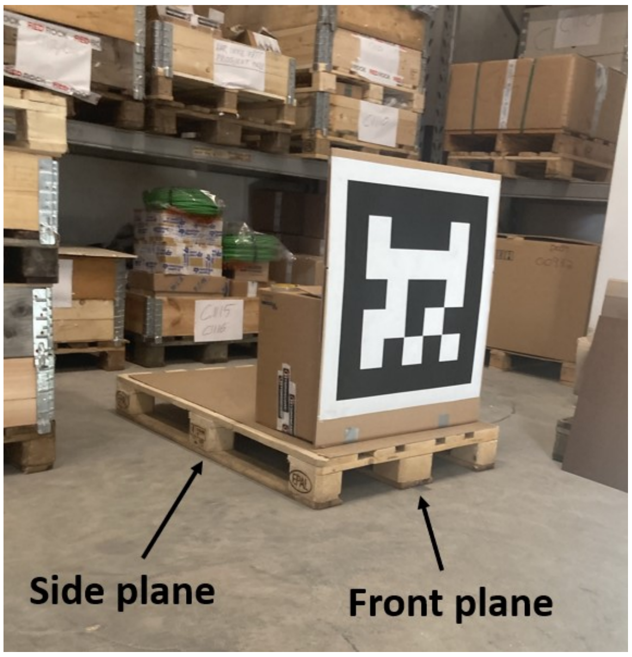
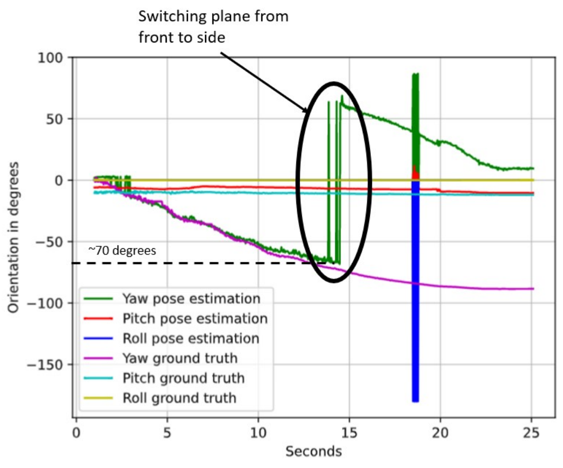
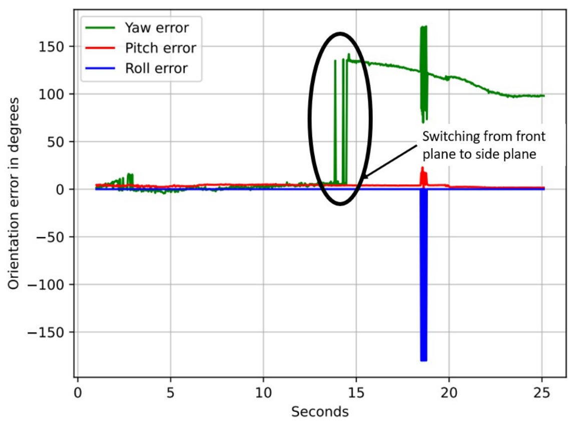


Figure 7.9: Switching planes between the front and side plane.

The swap in the detected plane also affects the results, where the spike in yaw value shown in the generated Figure 7.10a) happens because of the pose estimation vector calculating the vector of the side plane instead of the front plane, and the value goes from approximately -70° to approximately 65°, as the ground truth vector suddenly is compared to the side plane vector. Consequently, the error is spiking as well shown in Figure 7.10b) going from a stable value around 0° to approximately 140°. From the expectations presented in Section 3.2 related to the limitations of the algorithm when the angle becomes greater than ±45°, it is found that the algorithm is capable of detecting the front plane vector to approximately -70°, at this stage its switches to the side plane.



(a) Estimation result when switching planes.



(b) Error due to the switching of planes.

Figure 7.10: Switching plane from front to side.

7.4 Pose Estimation Sources of Errors

Through the procedures of testing and experimenting, some points on potential sources of errors have been noted. In the following subsections are a few highlighted sources of errors presented and described.

7.4.1 AprilTag Marker Error

The ground truth in this project is defined based on the parameters generated from the AprilTag marker. When considering the results from the AprilTag marker, potential sources of error should be taken into consideration. Key features of AprilTag are the detection of the marker's corners that define the center, and the black and white patterns which show the marker's id. Under varying light conditions or occlusion, these results might be affected and the performance of the AprilTag marker will decrease. This can result in both missing values and a drifting ground-truth value not showing the actual values. However, multiple studies related to AprilTag markers as ground truth shows that AprilTag is a proper tool when placed in closed environments, where external influences are controllable [91, 92]. This applies to the stand still and moving tests performed for this project. On the other side, is the performance of AprilTag more unstable when placed in outside and uncontrollable environments, which should be considered in future experiments.

7.4.2 Deviations in Orientation of the Camera

One challenge related to the sampling sequence is the potential deviation of the camera's orientation between each sampling angle. When recording the moving test, the authors have followed a marked path trying to generate a sample as accurately as possible and with stable orientations of the camera for each sample following the 90° half circle path. At the same time, is it expected that the orientation of the camera might deviate some for each sample, illustrated in Figure 7.11, where the dark points show the sample and the arrow shows the orientation for the camera. As the camera's origin might slightly vary for each sample in the pitch, roll, and yaw dimensions, potential deviations in the final results need to be considered. Accordingly, these varying samples might affect the generated curve showing both the pose of the pallet and the error of the generated vector compared to the ground truth AprilTag marker.

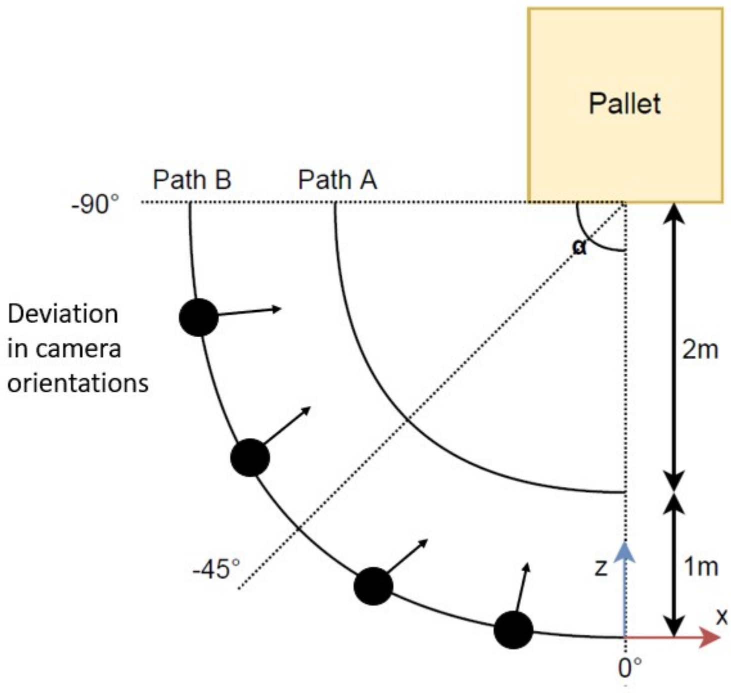


Figure 7.11: Potential varying camera orientations.

However, the key evaluation parameters are compared to the ground truth of the AprilTag marker, which also will be a result of the small deviations in the camera's orientation. Accordingly, the potential small deviation because of the camera's orientation will be compensated for in the relation between the pose vector and the AprilTag marker.

7.4.3 RANSAC Algorithm

As the RANSAC algorithm is programmed to detect the plane with the largest point density in the vertical direction, potential external influences on point density will affect the algorithm. An example of this scenario is when cargo is placed on top of the pallet, where the RANSAC algorithm also will consider the points from the load. This can also be identified in the scenario, where the AprilTag marker is placed on top of the pallet, and the RANSAC algorithm includes parts of the marker in its search for the largest vertical plane seen in Figure 7.12. Consequently, the importance of the YOLOX bounding box precision is underscored, where the more precise the bounding box the fewer external points are included in the point cloud analysis.

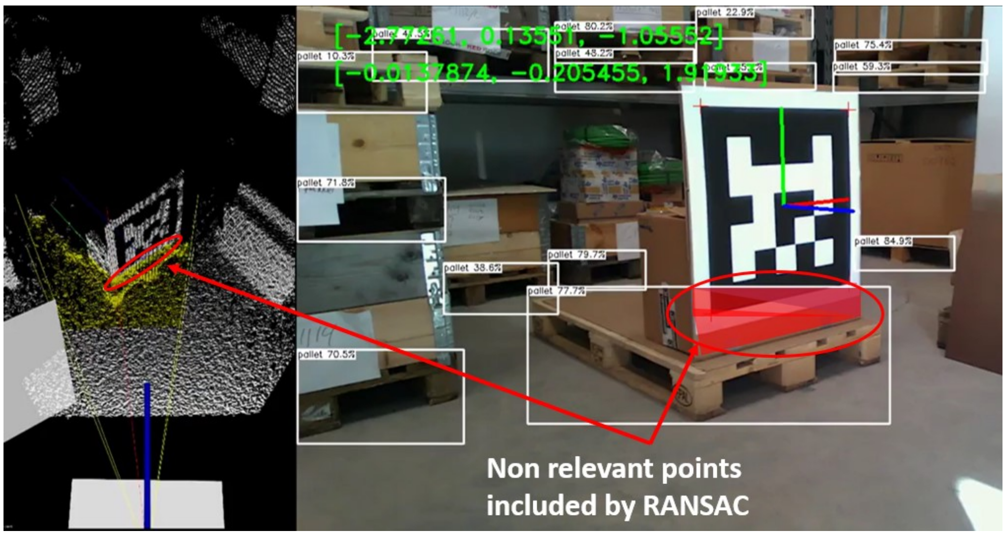


Figure 7.12: RANSAC detecting parts of the cargo on top of the pallet.

8. Conclusions and Future Work

Several conclusions are drawn from the requirements, expectations, and results of experimenting and testing. They are presented in this chapter, followed by the project's contributions to the topic of object detection and pose estimation in an industrial environment. The conclusions also present our answers to the research questions raised in Section 1.2. Finally, a few steps for future work are highlighted, indicating the work required to step forward from proof-of-concept to real-life operations.

8.1 Conclusions

The project developed a system solution capable of detecting the 6D pose of pallets. By creating a unique co-ordinate system that includes both (x,y) values generated from the YOLOX-S ML object detection algorithm, and (z) as well as (pitch, roll, yaw) values from automated arithmetic operations, a 3D point and 3D vector is calculated that can be used by forklifts to maneuver to the correct position for lifting pallets. For each of the four highlighted phases needed to achieve the system implementation, concrete outcomes are provided, presenting the performance of the developed solution. In the following paragraphs, conclusions for each of the project's four phases is presented.

By utilizing the COCO pre-trained YOLOX-S object detection algorithm and transfer learning with the LOCO dataset, a model capable of identifying pallets in industrial environments is developed. The ML algorithm is also documented in delivering Throughput of $\geq$30 FPS on GPU and AP of $\geq$40%, which satisfies the requirements presented in Section 3.2. To improve the model, optimization measures by Intel OpenVINO are applied, improving the model's performance from 191.53 to 70.49 in average latency on CPU. From the object detection, 2D x and y coordinates for 6D pose estimation are found, and they are further exported to the pose estimation model.

From the inputted 2D image from the object detection model, a 3D representation of the environment is generated from the Intel Realsense RGB-D camera. The depth data creates a point cloud, which is analyzed by the developed pose estimation model. The model takes advantage of arithmetic calculation and the RANSAC algorithm to calculate the object's 6D pose. The solution proves sufficient results in regards to the requirements defined in Subsection 3.2.1. The results from the standstill experiments show that the algorithm can estimate the position within 0.3 and 7.5 millimeters for the x- and y-axes, while the z-axis managed to be kept within 1.6 and 28.6 millimeters. The pitch orientation was kept within 3.65° and 5.21°, and the yaw orientation managed to be within 0.86° and 2.64°.

The limitations and capabilities of the algorithm were also evaluated. From the moving test, the algorithm is capable of estimating the pallet position up to -31.64°, after which the position starts to drift. As for orientation, the algorithm can estimate the yaw up to -68.85°. The capabilities correspond well to the standstill scenario where the plane reaches its limit after case 8 at -71.22°, beyond which it switches to the side plane and is no longer capable of detecting the desired front plane.

For evaluating the developed pose estimation solution an AprilTag marker has been integrated into the design to represent ground truth. Based on a proper literature study, presented in Subsection 3.4.8, AprilTag is suitable for the desired task as it has proven precise and robust results in closed environments without considerable external influences. The marker's flexibility has also made it suitable for the prototyping phase and under testing of the developed system solution.

8.2 Contributions

From transfer learning of the object detection network and development of the pose estimation model, as well as testing of a real-life application and development of pallet hole detection solution, a few contributions have been made. The contributions are summarized in the following list.

- The design of a step-by-step object detection and pose estimation system which includes an object detection algorithm, a pose estimation algorithm, and a validation method.

- Transfer learning of the state-of-the-art object detection model YOLOX on the LOCO dataset, followed through inference optimization by Intel OpenVINO for industrial robustness and performance improvement.

- The implementation of a pose estimation method targeting at a single pallet which combines arithmetic operations to calculate orientation and depth translation.

- The validation and experiments of the developed object detection and pose estimation system through 6D parameters based on real-life scenarios and requirements.

8.3 Future Work

To fully realize the system solution in real-life operations, a number of steps for further work are identified. They emphasize the need for network robustness, a sufficient amount of relevant data for the specific task, and the development of new techniques for network efficiency. The topics for future work are envisaged in the following list.

- Further development and training of the pallet hole detection network, to enable the design to identify both the pose of the pallet and the exact position of the pallet holes.

- Improve the performance of the solution in on-site scenarios, by training the YOLOX-S network on a more generalized dataset of pallets, with pallets of different sorts.

- Implement accelerated learning such as auto-labeling for ML, that will improve the performance of the network to make the bounding boxes more precise.

Bibliography

[1] *Powered Industrial Truck Operator Training.* URL: https://www.osha.gov/laws-regs/federalregister/1995-03-14.

[2] R. B. Rusu and S. Cousins. “3D is here: Point Cloud Library (PCL)”. In: *2011 IEEE International Conference on Robotics and Automation.* 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.

[3] R. Szeliski. *Computer Vision: Algorithms and Applications.* Second. Springer London Dordrecht Heidelberg New York, ISBN: 1868-0941, 2011.

[4] Intel. *Intel RealSense SDK 2.0.* https://www.intelrealsense.com/sdk-2/. 2021.

[5] M. Kang and N. J. Jameson. “Machine Learning: Fundamentals”. In: *Prognostics and Health Management of Electronics.* John Wiley & Sons, Ltd, 2018. Chap. 4, pp. 85–109. ISBN: 9781119515326. DOI: 10.1002/9781119515326.ch4.

[6] T. P. Trappenberg. In: *Fundamentals of Machine Learning.* Oxford: Oxford University Press, 2019. Chap. 4, pp. 66–86. ISBN: 9781119515326. DOI: 10.1093/oso/9780198828044.001.0001.

[7] L. Ulrich, E. Vezzetti, S. Moos, and F. Marcolin. “Analysis of RGB-D camera technologies for supporting different facial usage scenarios”. In: *Multimedia Tools and Applications* 79 (Oct. 2020). DOI: 10.1007/s11042-020-09479-0.

[8] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald. “Real-time large-scale dense RGB-D SLAM with volumetric fusion”. In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 598–626. DOI: 10.1177/0278364914551008.

[9] N. Yu and S. Wang. “Enhanced Autonomous Exploration and Mapping of an Unknown Environment with the Fusion of Dual RGB-D Sensors”. In: *Engineering* 5.1 (2019), pp. 164–172. ISSN: 2095-8099. DOI: 10.1016/j.eng.2018.11.014.

[10] *Point Cloud Library.* https://pointclouds.org/.

[11] *OpenCV.* https://opencv.org/.

[12] S. Brahmbhatt. In: *Practical OpenCV.* Apress Berkeley, 2013, p. 244. ISBN: 978-1-4302-6080-6. DOI: 10.1007/978-1-4302-6080-6.

[13] K. Suzuki. *Artificial Neural Networks.* Rijeka: IntechOpen, 2011. DOI: 10.5772/644.

[14] *Supervised Learning.* URL: https://www.uio.no/studier/emner/matnat/fys/HON1000/v20/studentblogg/veiledet-lering.html (visited on 07/14/2022).

[15] L. N. Smith. *A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay.* 2018. DOI: 10.48550/ARXIV.1803.09820.

[16] Q. Zhang, M. Zhang, T. Chen, Z. Sun, Y. Ma, and B. Yu. “Recent advances in convolutional neural network acceleration”. In: *Neurocomputing* 323 (2019), pp. 37–51. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.09.038.

[17] S. Albawi, T. A. Mohammed, and S. Al-Zawi. “Understanding of a convolutional neural network”. In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[18] M. Z. Alom, T. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, B. Essen, A. Awwal, and V. Asari. “A State-of-the-Art Survey on Deep Learning Theory and Architectures”. In: *Electronics* 8 (Mar. 2019), p. 292. DOI: 10.3390/electronics8030292.

[19] X. Ma, W. A. Najjar, and A. K. Roy-Chowdhury. “Evaluation and Acceleration of High-Throughput Fixed-Point Object Detection on FPGAs”. In: *IEEE Transactions on Circuits and Systems for Video Technology* 25.6 (2015), pp. 1051–1062. DOI: 10.1109/TCSVT.2014.2360030.

[20] D. Chicco. “Ten quick tips for machine learning in computational biology”. In: *BioData Mining* 10 (Dec. 2017), p. 35. DOI: 10.1186/s13040-017-0155-3.

[21] D. Bolya, S. Foley, J. Hays, and J. Hoffman. “TIDE: A General Toolbox for Identifying Object Detection Errors”. In: *Computer Vision – ECCV 2020*. Ed. by A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm. Cham: Springer International Publishing, 2020, pp. 558–573. ISBN: 978-3-030-58580-8.

[22] J. Davis and M. Goadrich. “The Relationship between Precision-Recall and ROC Curves”. In: *Proceedings of the 23rd International Conference on Machine Learning*. ICML ’06. Pittsburgh, Pennsylvania, USA: Association for Computing Machinery, 2006, pp. 233–240. ISBN: 1595933832. DOI: 10.1145/1143844.1143874.

[23] R. Padilla, S. L. Netto, and E. A. B. da Silva. “A Survey on Performance Metrics for Object-Detection Algorithms”. In: *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*. 2020, pp. 237–242. DOI: 10.1109/IWSSIP48289.2020.9145130.

[24] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár. *Microsoft COCO: Common Objects in Context*. 2014. DOI: 10.48550/ARXIV.1405.0312.

[25] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez. “Automatic generation and detection of highly reliable fiducial markers under occlusion”. In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. ISSN: 0031-3203. DOI: 10.1016/j.patcog.2014.01.005.

[26] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios. “Fiducial Markers for Pose Estimation: Overview, Applications and Experimental Comparison of the ARTag, AprilTag, ArUco and STag Markers”. In: *Journal of Intelligent and Robotic Systems: Theory and Applications* 101 (4 Apr. 2021). ISSN: 15730409. DOI: 10.1007/s10846-020-01307-9.

[27] Z. Zhang, Y. Hu, G. Yu, and J. Dai. *DeepTag: A General Framework for Fiducial Marker Design and Detection*. 2021. DOI: 10.48550/ARXIV.2105.13731.

[28] R. Girshick, J. Donahue, T. Darrell, and J. Malik. *Rich feature hierarchies for accurate object detection and semantic segmentation*. 2014. arXiv: 1311.2524 [cs.CV].

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. *You Only Look Once: Unified, Real-Time Object Detection.* 2016. arXiv: 1506.02640 [cs.CV].

[30] T. Li, B. Huang, C. Li, and M. Huang. "Application of convolution neural network object detection algorithm in logistics warehouse". In: *The Journal of Engineering* 2019 (Mar. 2019). DOI: 10.1049/joe.2018.9180.

[31] R. Iinuma, Y. Hori, H. Onoyama, T. Fukao, and Y. Kubo. "Pallet Detection and Estimation for Fork Insertion with RGB-D Camera". In: Institute of Electrical and Electronics Engineers Inc., Aug. 2021, pp. 854–859. ISBN: 9781665441001. DOI: 10.1109/ICMA52036.2021.9512641.

[32] T. Li, Q. Jin, B. Huang, C. Li, and M. Huang. "Cargo pallets real-time 3D positioning method based on computer vision". In: *The Journal of Engineering* 2019 (Mar. 2019). DOI: 10.1049/joe.2018.9053.

[33] Intel. *LiDAR Camera L515 Datasheet.* 2021. URL: https://dev.intelrealsense.com/docs/lidar-camera-l515-datasheet.

[34] C. Mayershofer, D.-M. Holm, B. Molter, and J. Fottner. "LOCO: Logistics objects in context". In: *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA).* IEEE. 2020, pp. 612–617. DOI: 10.1109/ICMLA51294.2020.00102.

[35] A. Z. B. Sekachev and N. Manovich. *Computer Vision Annotation Tool: A Universal Approach to Data Annotation.* https://www.intel.com/content/www/us/en/developer/articles/technical/computer-vision-annotation-tool-a-universal-approach-to-data-annotation.html. 2019.

[36] *PyTorch.* URL: https://pytorch.org/.

[37] Y. Gorbachev, M. Fedorov, I. Slavutin, A. Tugarev, M. Fatekhov, and Y. Tarkan. "OpenVINO Deep Learning Workbench: Comprehensive Analysis and Tuning of Neural Networks Inference". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops.* Oct. 2019.

[38] *Intelő Distribution of OpenVINO Toolkit.* URL: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html (visited on 04/13/2022).

[39] *What is OpenVINO? The Ultimate Overview.* URL: https://viso.ai/computer-vision/intel-openvino-toolkit-overview/#:~:text=The%20OpenVINO%20toolkit%20covers%20both,well%20as%20pre%2Doptimized%20kernels. (visited on 04/13/2022).

[40] K. Shabalina, A. Sagitov, L. Sabirova, H. Li, and E. Magid. "ARTag, AprilTag and CALTag Fiducial Systems Comparison in a Presence of Partial Rotation: Manual and Automated Approaches". In: Jan. 2020, pp. 536–558. ISBN: 978-3-030-11291-2. DOI: 10.1007/978-3-030-11292-9_27.

[41] K. G. Derpanis. "Overview of the RANSAC Algorithm". In: 2005.

[42] *Robust line model estimation using RANSAC.* URL: https://scikit-image.org/docs/stable/auto_examples/transform/plot_ransac.html.

[43] *Data format.* 2022. URL: https://cocodataset.org/#format-data.

[44] *CVAT and More Video Annotation Tools.* URL: https://medium.com/@iowt.uva/cvat-and-more-video-annotation-tools-85e93ddcd0a41 (visited on 04/10/2022).

[45] openvinotoolkit. *Computer Vision Annotation Tool (CVAT)*. https://github.com/openvinotoolkit/cvat. commit:967b0fe. Nov. 2021.

[46] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He. "A Comprehensive Survey on Transfer Learning". In: *Proceedings of the IEEE* 109.1 (2021), pp. 43–76. DOI: 10.1109/JPROC.2020.3004555.

[47] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[48] R. Girshick. *Fast R-CNN*. 2015. DOI: 10.48550/ARXIV.1504.08083.

[49] S. Ren, K. He, R. Girshick, and J. Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016. arXiv: 1506.01497 [cs.CV].

[50] J. Du. "Understanding of Object Detection Based on CNN Family and YOLO". In: *Journal of Physics: Conference Series* 1004 (Apr. 2018), p. 012029. DOI: 10.1088/1742-6596/1004/1/012029.

[51] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma. "A Review of Yolo Algorithm Developments". In: *Procedia Computer Science* 199 (2022). The 8th International Conference on Information Technology and Quantitative Management: Developing Global Digital Economy after COVID-19, pp. 1066–1073. ISSN: 1877-0509. DOI: 10.1016/j.procs.2022.01.135.

[52] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun. "YOLOX: Exceeding YOLO Series in 2021". In: *arXiv preprint arXiv:2107.08430* (2021).

[53] A. Sharma. *Introduction to the YOLO Family*. https://pyimagesearch.com/2022/04/04/introduction-to-the-yolo-family. 2022.

[54] FateScript, Joker316701882, zhiqwang, tonysy, GOATmessi7, PieroMacaluso, imneonizer, SWHL, and ankandrew. *YOLOX Exceeding YOLO series in 2021*. commit:ca9bc37. Jan. 2022. URL: https://github.com/Megvii-BaseDetection/YOLOX.

[55] B. Liu, J. Huang, S. Lin, Y. Yang, and Y. Qi. "Improved YOLOX-S Abnormal Condition Detection for Power Transmission Line Corridors". In: *2021 IEEE 3rd International Conference on Power Data Science (ICPDS)*. 2021, pp. 13–16. DOI: 10.1109/ICPDS54746.2021.9690074.

[56] *Object Detection on COCO test-dev*. https://paperswithcode.com/sota/object-detection-on-coco?tag_filter=15.

[57] S. Ma, H. Lu, Y. Wang, and H. Xue. "YOLOX-Mobile: A Target Detection Algorithm More Suitable for Mobile Devices". In: *Journal of Physics: Conference Series* 2203.1 (Feb. 2022), p. 012030. DOI: 10.1088/1742-6596/2203/1/012030.

[58] A. Balasubramaniam and S. Pasricha. *Object Detection in Autonomous Vehicles: Status and Open Challenges*. 2022. DOI: 10.48550/ARXIV.2201.07706.

[59] FateScript, Joker316701882, zhiqwang, tonysy, GOATmessi7, PieroMacaluso, imneonizer, SWHL, and ankandrew. *Train Custom Data*. commit:c7c5ccf. Jan. 2022. URL: https://github.com/Megvii-BaseDetection/YOLOX/blob/main/docs/train_custom_data.md.

[60] FateScript and SWHL. *YOLOX-S Custom Exp File*. commit:b861b22. Jan. 2022. URL: https://github.com/Megvii-BaseDetection/YOLOX/blob/main/exps/example/custom/yolox_s.py.

[61] A. Demidovskij, A. Tugaryov, M. Fatekhov, E. Aidova, E. Stepyreva, M. Shevtsov, and Y. Gorbachev. "Accelerating Object Detection Models Inference within Deep Learning Workbench". In: Institute of Electrical and Electronics Engineers Inc., 2021. ISBN: 9781665427142. DOI: 10.1109/ICEET53442.2021.9659634.

[62] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner. "AI Accelerator Survey and Trends". In: Institute of Electrical and Electronics Engineers Inc., 2021. ISBN: 9781665423694. DOI: 10.1109/HPEC49654.2021.9622867.

[63] S. Kim, S. Na, B. Y. Kong, J. Choi, and I. C. Park. "Real-Time SSDLite Object Detection on FPGA". In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29 (6 June 2021), pp. 1192–1205. ISSN: 15579999. DOI: 10.1109/TVLSI.2021.3064639.

[64] *Convert model with Model Optimizer*. URL: https://docs.openvino.ai/latest/openvino_docs_MO_DG_Deep_Learning_Model_Optimizer_DevGuide.html.

[65] J. Bai, F. Lu, K. Zhang, et al. *ONNX: Open Neural Network Exchange*. https://github.com/onnx/onnx. 2019.

[66] FateScript and SWHL. *YOLOX-ONNXRuntime in Python*. commit:6b49fd2. Jan. 2022. URL: https://github.com/Megvii-BaseDetection/YOLOX/tree/main/demo/ONNXRuntime.

[67] *Converting an ONNX Model*. 2021. URL: https://docs.openvino.ai/latest/openvino_docs_MO_DG_prepare_model_convert_model_Convert_Model_From_ONNX.html.

[68] Intel. *LiDAR Camera L515 Datasheet*. https://dev.intelrealsense.com/docs/lidar-camera-l515-datasheet. 2021.

[69] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004.

[70] A. Lutica. "CCD and CMOS image sensors in new HD cameras". In: *Proceedings Elmar - International Symposium Electronics in Marine* September (2011), pp. 133–136. ISSN: 13342630.

[71] D. Griffiths and J. Boehm. "A Review on deep learning techniques for 3D sensed data classification". In: *Remote Sensing* 11 (12 June 2019). ISSN: 20724292. DOI: 10.3390/rs11121499.

[72] W. Gellert, M. Hellwich, H. Kastner, and H. Kustner. *The VNR concise encyclopedia of mathematics*. Springer Science and Business Media, 2012. ISBN: 978-94-011-69844. DOI: 10.1007/978-94-011-6982-0.

[73] D. Sunday. *Practical Geometry Algorithms: With C++ Code*. 2021.

[74] J. Wang and E. Olson. "AprilTag 2: Efficient and robust fiducial detection". In: vol. 2016-November. Institute of Electrical and Electronics Engineers Inc., Nov. 2016, pp. 4193–4198. ISBN: 9781509037629. DOI: 10.1109/IROS.2016.7759617.

[75] M. Ben-Ezra and S. K. Nayar. "Motion-based motion deblurring". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26 (6 June 2004), pp. 689–698. ISSN: 01628828. DOI: 10.1109/TPAMI.2004.1.

[76] C. Cai, A. Liu, and B. Zhang. "Motion deblurring from a single image". In: Institute of Electrical and Electronics Engineers Inc., Sept. 2016, pp. 406–410. ISBN: 9781509019151. DOI: 10.1109/CSCWD.2016.7566023.

[77] Y. Xu, X. Hu, and S. Peng. "Blind motion deblurring using optical flow". In: *Optik* 126 (1 2015), pp. 87–94. ISSN: 00304026. DOI: 10.1016/j.ijleo.2014.08.139.

[78] A. Kaehler and G. Bradski. *Learning OpenCV 3: computer vision in C++ with the OpenCV library.* " O'Reilly Media, Inc.", 2016. ISBN: 978-1491937990.

[79] M. L. Waskom. "Seaborn: statistical data visualization". In: *Journal of Open Source Software* 6.60 (2021), p. 3021. DOI: 10.21105/joss.03021.

[80] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, et al. "SciPy 1.0: fundamental algorithms for scientific computing in Python". In: *Nature methods* 17.3 (2020), pp. 261–272.

[81] L. T. DeCarlo. "On the meaning and use of kurtosis." In: *Psychological methods* 2.3 (1997), p. 292.

[82] B. Peter and A. Bruce. *Practical statistics for data scientists.* 2017.

[83] D. P. Doane and L. E. Seward. "Measuring skewness: a forgotten statistic?" In: *Journal of statistics education* 19.2 (2011). DOI: 10.1080/10691898.2011.11889611.

[84] *Counterbalanced Forklift 3,500 lbs. Capacity KBET18 48V Electric Forklift Truck.* 2019. URL: https://www.kion-na.com/KION-NA-Master/Downloads/Spec-Sheets/Baoli/baoli_spec_KBET18_single-pg_red.pdf.

[85] *Diesel Forklift Trucks 13,000 to 17,500 lbs. Capacity H60D, H70D, H80D, H80D-900, and H80D-1100 SERIES.* 2013. URL: https://www.kion-na.com/KION-NA-Master/Downloads/Spec-Sheets/396-03-D-Spec-Sheet.pdf.

[86] *Engine powered forklift trucks 4.0 - 5.0 tonnes.* 2008. URL: https://qpsearch.bt-forklifts.com/PDFSearch/GetPDF.asp?artno=745562-040.

[87] *Electric powered forklift 2.0 - 5.0 ton.* 2014. URL: https://www.toyota-equiposindustriales.com.uy/catalogos/Catalogo-Tecnico-Autoelevador-Electrico-Toyota-8FBE-1020.pdf.

[88] H. Deng and H. Wickham. "Density estimation in R". In: *Electronic publication* (2011).

[89] B. W. Silverman. *Density estimation for statistics and data analysis.* Routledge, 2018.

[90] *Automatic annotation.* Sept. 2021. URL: https://openvinotoolkit.github.io/cvat/docs/manual/advanced/automatic-annotation/.

[91] M. Fiala. "Designing Highly Reliable Fiducial Markers". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.7 (2010), pp. 1317–1324. DOI: 10.1109/TPAMI.2009.146.

[92] M. Kalaitzakis, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios. "Experimental Comparison of Fiducial Markers for Pose Estimation". In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS).* 2020, pp. 781–789. DOI: 10.1109/ICUAS48674.2020.9213977.

Appendices

A. Machine Learning Training Results

A.1 YOLOX-S Only Pallet with LOCO

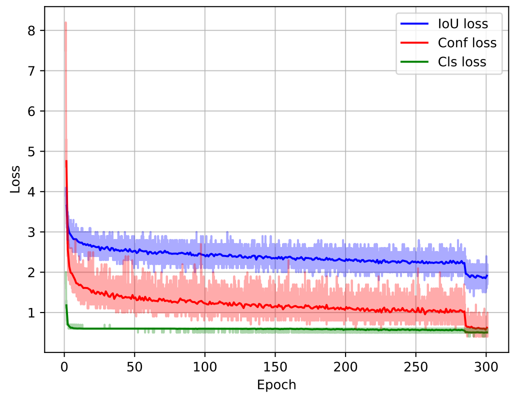


Figure A.1: Training loss of YOLOX-S LOCO only pallet.

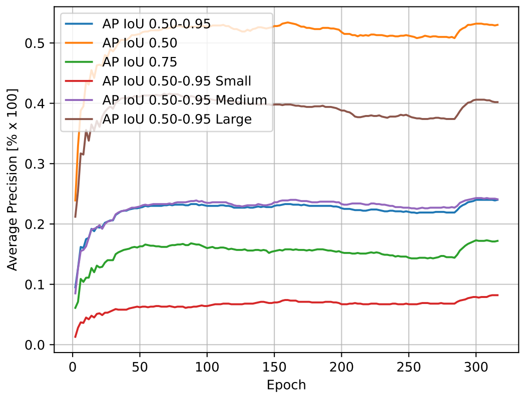


Figure A.2: Average Precision per epoch of YOLOX-S LOCO only pallet.

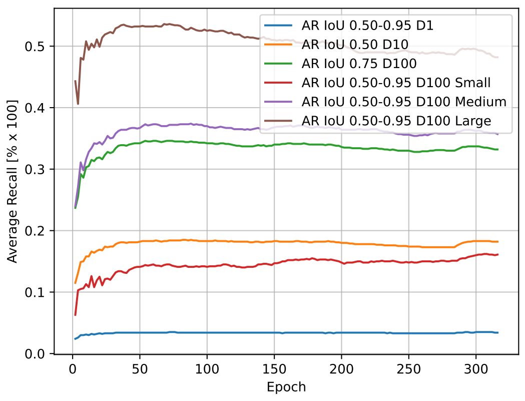


Figure A.3: Average Recall per epoch of YOLOX-S LOCO only pallet.

Table A.1: Training parameters for YOLOX-S LOCO only pallet.

Keys	**Values**
batch_size	8
fp16	TRUE
num_machines	1
occupy	TRUE
exp_file	exps/example/custom/yolox_s_loco_cvat_1245.py'
ckpt	models/yolo_s_only_pallet_mars.pth
cache	FALSE
seed	None
output_dir	./YOLOX_outputs'10
print_interval	10
eval_interval	2
num_classes	1
depth	0.33
width	0.5
act	silu
data_num_workers	4
input_size	(640, 640)
multiscale_range	5
data_dir	datasets/loco2020_only_pallet_1245'
train_ann	instances_Train.json'
val_ann	instances_Validation.json'
test_ann	instances_default.json'
mosaic_prob	1.0
mixup_prob	1.0
hsv_prob	1.0
flip_prob	0.5
degrees	10
translate	0.1
mosaic_scale	0.1
mixup_scale	0.5
shear	2
enable_mixup	TRUE
warmup_epochs	5
max_epoch	300
warmup_lr	0
basic_lr_per_img	0.00015625
scheduler	yoloxwarmcos
no_aug_epochs	15
min_lr_ratio	0.05
ema	TRUE
weight_decay	0.0005
momentum	0.9
exp_name	yolox_s_loco_cvat_1245
test_size	(640, 640)
test_conf	0.01
nmsthre	0.65

A.2 YOLOX-S Pallet Void with LOCO

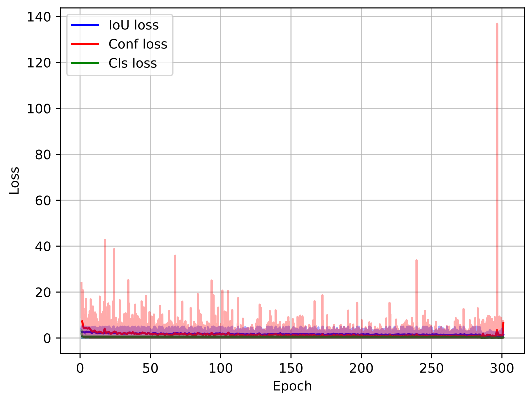


Figure A.4: Training loss of YOLOX-S LOCO pallet void.

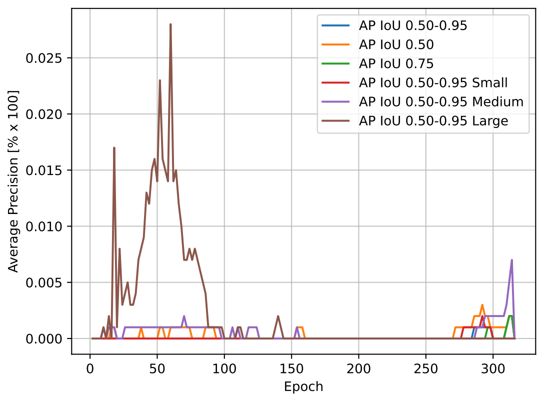


Figure A.5: Average Precision per epoch of YOLOX-S LOCO pallet void.

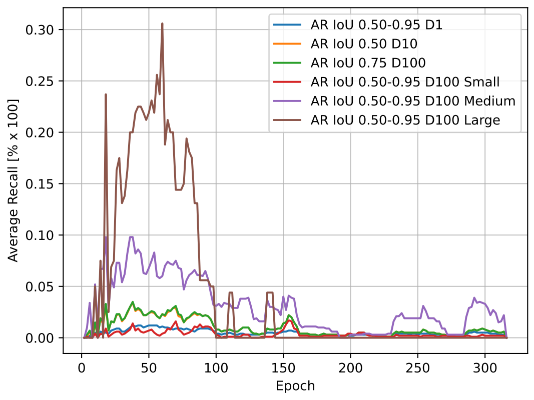


Figure A.6: Average Recall per epoch of YOLOX-S LOCO pallet void.

Table A.2: Training parameters for YOLOX-S LOCO pallet void.

Keys	**Values**
batch_size	8
fp16	TRUE
num_machines	1
occupy	TRUE
exp_file	exps/example/custom/yolox_s_only_pallet_void.py
ckpt	models/yolox_s_only_pallet_void.pth
cache	FALSE
seed	None
output_dir	./YOLOX_outputs'
print_interval	10
eval_interval	2
num_classes	1
depth	0.33
width	0.5
act	silu
data_num_workers	4
input_size	(640, 640)
multiscale_range	5
data_dir	datasets/loco2020_only_pallet_void_limited'
train_ann	instances_Train.json'
val_ann	instances_Validation.json'
test_ann	instances_default.json'
mosaic_prob	1.0
mixup_prob	1.0
hsv_prob	1.0
flip_prob	0.5
degrees	10
translate	0.1
mosaic_scale	0.1
mixup_scale	0.5
shear	2
enable_mixup	TRUE
warmup_epochs	5
max_epoch	300
warmup_lr	0
basic_lr_per_img	0.00015625
scheduler	yoloxwarmcos
no_aug_epochs	15
min_lr_ratio	0.05
ema	TRUE
weight_decay	0.0005
momentum	0.9
exp_name	yolox_s_only_pallet_void'
test_size	(640, 640)
test_conf	0.01
nmsthre	0.65

B. Case Plot Selection

	yaw_mu_ground_truth	yaw_sigma_ground_truth	x_mu	x_sigma	x_mu_ground_truth	x_mu_error	x_mu_error_mm
0	-0.211	0.027	0.08635239355	0.004919263191	0.08774982581	**0.0014**	1.4
1	-18.611	0.019	-0.09398585294	0.003295737741	-0.09290794706	0.0011	1.1
2	-25.110	0.045	0.001671427778	0.002378945942	0.0003252486444	0.0013	1.3
3	-35.424	0.064	0.02331372688	0.001389868201	0.02454562688	0.0012	1.2
4	-43.308	0.064	-0.008932800538	0.001389868201	-0.008664833656	**0.0003**	0.3
5	-52.644	0.159	-0.03889319425	0.007833917532	-0.03836106092	0.0005	0.5
6	-61.423	0.067	-0.09130973418	0.00348703692	-0.09174033671	0.0004	0.4
7	-66.076	0.168	-0.01489213542	0.00134954856	-0.01399380417	0.0009	0.9
8	-71.219	0.177	-0.002394190233	0.001344800279	-0.005261072791	0.0029	2.9
9	-77.503	0.140	-0.03744935399	0.004330054341	-0.03223499387	0.0052	5.2
10	-81.386	0.561	-0.02266833494	0.0102138571	-0.0212172241	0.0015	1.5
11	-84.738	0.366	-0.03701207381	0.002234353848	-0.03907897143	0.0021	2.1
12	-86.964	0.757	-0.03099355062	0.0008945286305	-0.03805313827	0.0071	7.1
13	-88.280	0.581	-0.144067561	0.001928267318	-0.1271180244	0.0169	16.9

	yaw_mu_ground_truth	yaw_sigma_ground_truth	x_sigma_ground_truth	y_mu	y_sigma	y_mu_ground_truth
0	-0.211	0.027	0.00060	-0.2271744839	0.0009520852902	-0.2197104731
1	-18.611	0.019	0.00043	-0.2134558353	0.0007054229874	-0.2155289412
2	-25.110	0.045	0.00043	-0.2128397333	0.001180715625	-0.2115160556
3	-35.424	0.064	0.00041	-0.2111212258	0.0007529336416	-0.2109381075
4	-43.308	0.064	0.00041	-0.20761	0.0007529336416	-0.2080700645
5	-52.644	0.159	0.00995	-0.2113226552	0.001602162416	-0.2112806207
6	-61.423	0.067	0.00030	-0.2083526835	0.001711486887	-0.2071208734
7	-66.076	0.168	0.00024	-0.2070823125	0.0009305723408	-0.2061544063
8	-71.219	0.177	0.00041	-0.203958907	0.001164037849	-0.2040371395
9	-77.503	0.140	0.00028	-0.202640773	0.002499923814	-0.202597816
10	-81.386	0.561	0.00069	-0.2007039398	0.009347926171	-0.2007346024
11	-84.738	0.366	0.00021	-0.1995975714	0.001262819894	-0.1995609524
12	-86.964	0.757	0.00066	-0.2002107037	0.001056176922	-0.1994276543
13	-88.280	0.581	0.00032	-0.200646122	0.005696802708	-0.197444622

	yaw_mu_ground_truth	yaw_sigma_ground_truth	y_mu_error	y_mu_error_mm	y_sigma_ground_truth	z_mu
0	-0.211	0.027	**0.007464010753**	7.5	0.0005892317578	1.956482688
1	-18.611	0.019	0.002073105882	2.1	0.0002139555256	1.980525647
2	-25.110	0.045	0.001323677778	1.3	0.000286829006	1.968620778
3	-35.424	0.064	**0.0001831182796**	0.2	0.0003811707058	1.95993086
4	-43.308	0.064	0.0004600645161	0.5	0.0003811707058	1.947544946
5	-52.644	0.159	0.00004203448276	0.0	0.0008598518865	1.950023908
6	-61.423	0.067	0.001231810127	1.2	0.0001597805299	1.931866582
7	-66.076	0.168	0.00092790625	0.9	0.0005090609733	1.921885313
8	-71.219	0.177	0.00007823255814	0.1	0.000383938595	1.918030116
9	-77.503	0.140	0.00004295705522	0.0	0.0001561867613	1.908956687
10	-81.386	0.561	0.0000306626506	0.0	0.0003060747961	1.911791687
11	-84.738	0.366	0.00003661904762	0.0	0.0002201251342	1.889777262
12	-86.964	0.757	0.0007830493827	0.8	0.0004457767142	1.88281
13	-88.280	0.581	0.0032015	3.2	0.0002605011544	1.875870488

	yaw_mu_ground_truth	yaw_sigma_ground_truth	z_sigma	z_mu_ground_truth	z_mu_error	z_sigma_ground_truth	yaw_mu
0	-0.211	0.027	0.003076966317	1.98508871	**0.02860602151**	0.002437056389	1.311519221
1	-18.611	0.019	0.002703418047	1.971471765	0.009053882353	0.001576754184	-19.80082306
2	-25.110	0.045	0.00764986949	1.966129556	0.002491222222	0.002348576123	-25.99526887
3	-35.424	0.064	0.002863889772	1.961561828	**0.001630967742**	0.001858223827	-32.78753843
4	-43.308	0.064	0.002863889772	1.956763656	0.009218709677	0.001858223827	-41.29974717
5	-52.644	0.159	0.004907787608	1.94581931	0.004204597701	0.001855928045	-49.49750118
6	-61.423	0.067	0.007223289609	1.930682025	0.001184556962	0.00116482327	-56.77079688
7	-66.076	0.168	0.005641211814	1.922459583	0.0005742708333	0.00226429529	-61.01226689
8	-71.219	0.177	0.006583323174	1.918124651	0.00009453488373	0.001350568919	-65.47315058
9	-77.503	0.140	0.0103528555	1.907147975	0.001808711656	0.001107117821	55.49255377
10	-81.386	0.561	0.03755153201	1.909184458	0.002607228916	0.002454801652	47.10675316
11	-84.738	0.366	0.00623735133	1.897054048	0.007276785714	0.001018185653	35.83309521
12	-86.964	0.757	0.00536374769	1.884287654	0.001477654321	0.001552213733	27.76756079
13	-88.280	0.581	0.043479957	1.867710244	0.008160243902	0.0007835241943	13.76689299

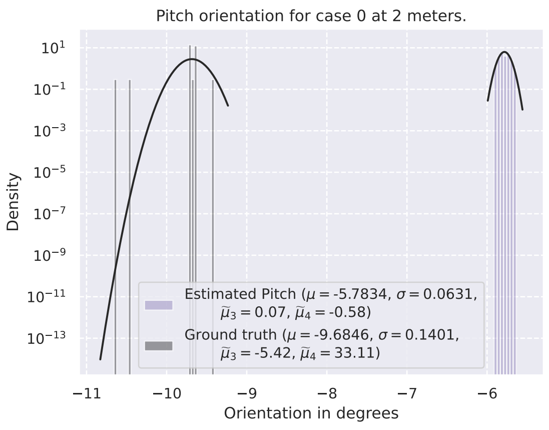
	yaw_mu_ground_truth	yaw_sigma_ground_truth	yaw_sigma	yaw_mu_ground_truth	yaw_mu_error	yaw_sigma_ground_truth
0	-0.211	0.027	0.4763810176	-0.2105085642	1.522027785	0.02693963573
1	-18.611	0.019	0.4430343975	-18.611131	1.189692057	0.01918801449
2	-25.110	0.045	0.4590529713	-25.11014697	**0.8851219093**	0.04494403406
3	-35.424	0.064	0.5685968158	-35.42371776	**2.636179332**	0.06433145431
4	-43.308	0.064	0.5685968158	-43.30750665	2.007759481	0.06433145431
5	-52.644	0.159	0.6727874294	-52.64440506	3.146903882	0.1590618948
6	-61.423	0.067	1.008389263	-61.42265932	4.651862442	0.06678472567
7	-66.076	0.168	1.072548497	-66.0755377	5.063270811	0.1681003217
8	-71.219	0.177	1.209910235	-71.21867496	5.745524389	0.1767142977
9	-77.503	0.140	0.7383375713	-77.50320984	132.9957636	0.1398385974
10	-81.386	0.561	1.130489897	-81.38604888	128.492802	0.5609767389
11	-84.738	0.366	0.4617065491	-84.7376044	120.5706996	0.366456443
12	-86.964	0.757	0.3804700198	-86.96377881	114.7313396	0.756525719
13	-88.280	0.581	1.006254181	-88.27966113	102.0465541	0.5809325828

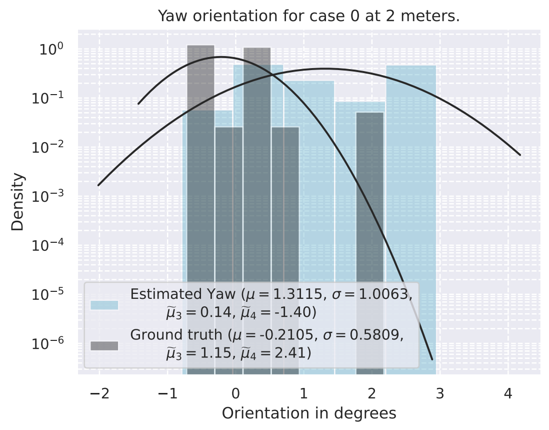
	yaw_mu_ground_truth	yaw_sigma_ground_truth	pitch_mu	pitch_sigma	pitch_mu_ground_truth	pitch_mu_error
0	-0.211	0.027	-5.783382895	0.03382072776	-9.684582664	3.901199769
1	-18.611	0.019	-6.391673585	0.02940684814	-10.07876874	3.687095151
2	-25.110	0.045	-6.559087779	0.02159653537	-10.0288156	**3.469727818**
3	-35.424	0.064	-5.487683683	0.02444343128	-10.03594392	4.548260234
4	-43.308	0.064	-5.090661158	0.02444343128	-10.29748123	**5.206820074**
5	-52.644	0.159	-5.250034046	0.05400984994	-10.33136071	5.081326666
6	-61.423	0.067	-5.790735291	0.05746798228	-10.8418273	5.051092008
7	-66.076	0.168	-6.143564491	0.02900097619	-10.72937383	4.585809337
8	-71.219	0.177	-6.567549172	0.03755589186	-10.90008576	4.332536583
9	-77.503	0.140	-6.980616038	0.07982674381	-11.26022433	4.27960829
10	-81.386	0.561	-7.290726251	0.3054334385	-11.42518573	4.134459478
11	-84.738	0.366	-7.472478478	0.05360196392	-11.61113158	4.138653104
12	-86.964	0.757	-9.33613265	0.04331734412	-11.85845074	2.52231809
13	-88.280	0.581	-10.13926731	0.06306684394	-12.02987627	1.890608958

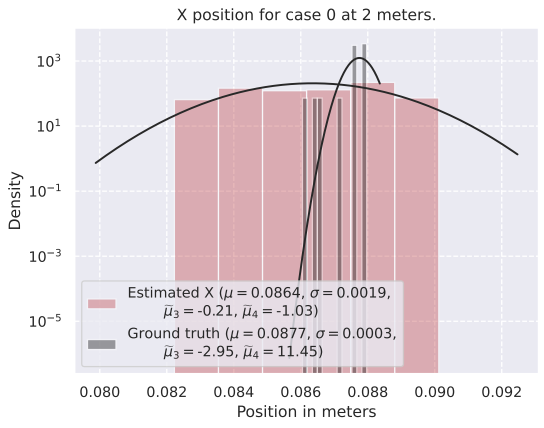
	yaw_mu_ground_truth	yaw_sigma_ground_truth	pitch_sigma_ground_truth
0	-0.211	0.027	0.1044188789
1	-18.611	0.019	0.05443435327
2	-25.110	0.045	0.06690764144
3	-35.424	0.064	0.06003364467
4	-43.308	0.064	0.06003364467
5	-52.644	0.159	0.08480460368
6	-61.423	0.067	0.0235790179
7	-66.076	0.168	0.05265469122
8	-71.219	0.177	0.05730897157
9	-77.503	0.140	0.09948938124
10	-81.386	0.561	0.162375241
11	-84.738	0.366	0.2608922712
12	-86.964	0.757	0.458499121
13	-88.280	0.581	0.1400864649

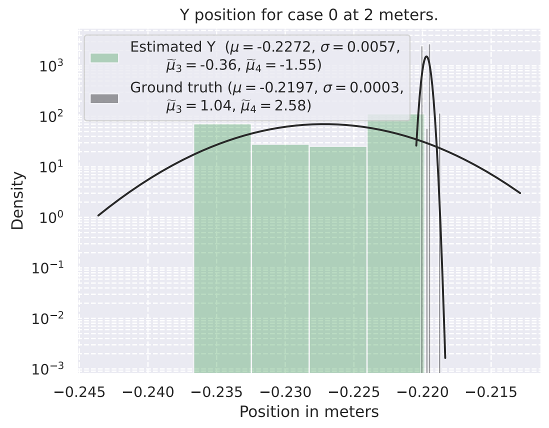
Table B.1: All case plots with error.

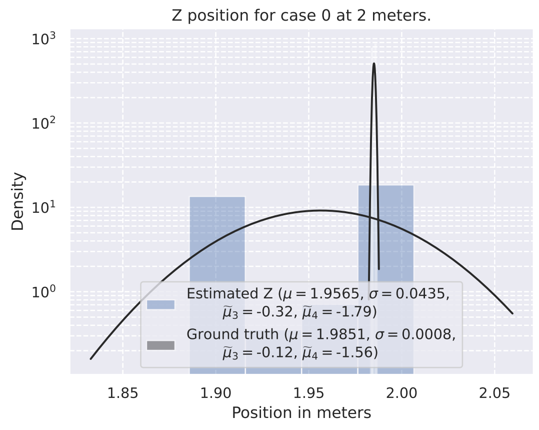
C. Standstill Density Plots from Scenario 1

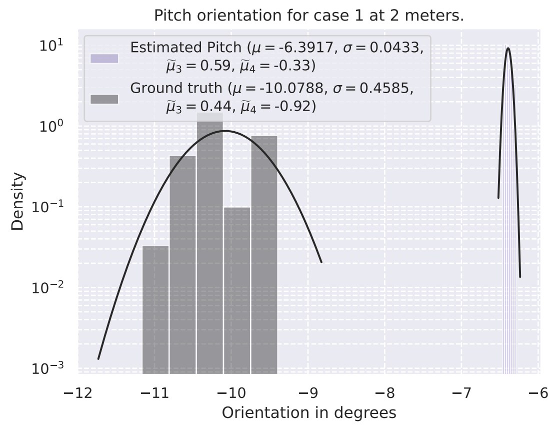


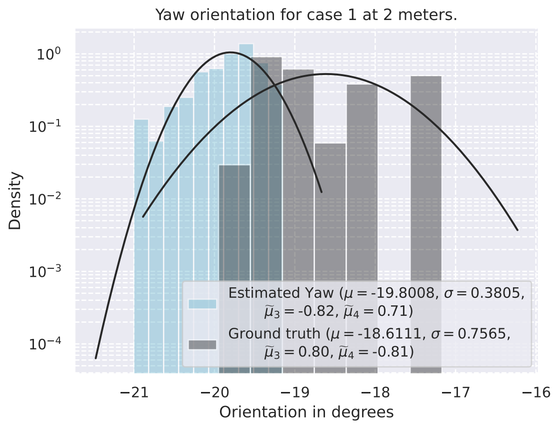


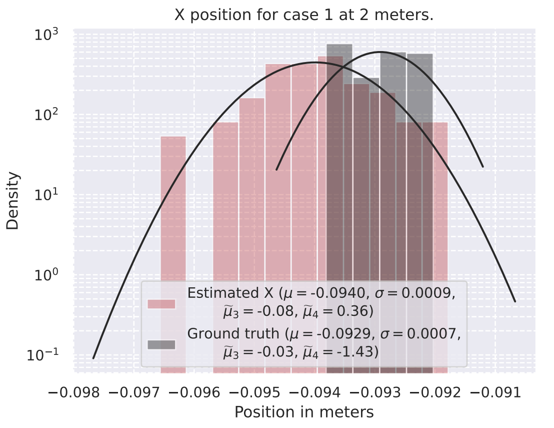


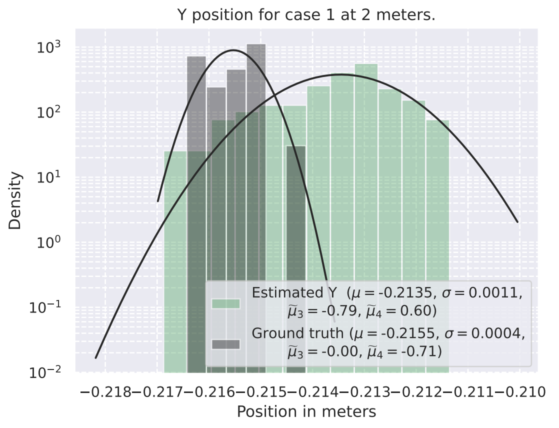


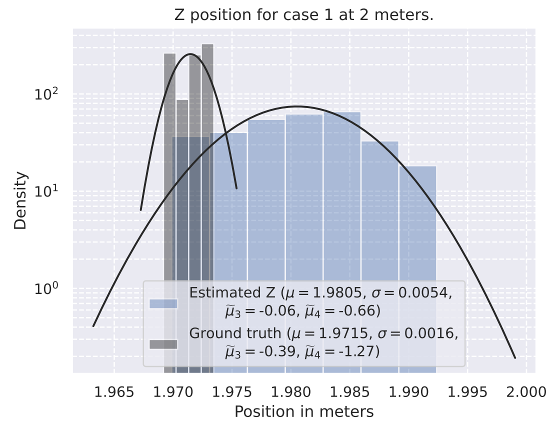


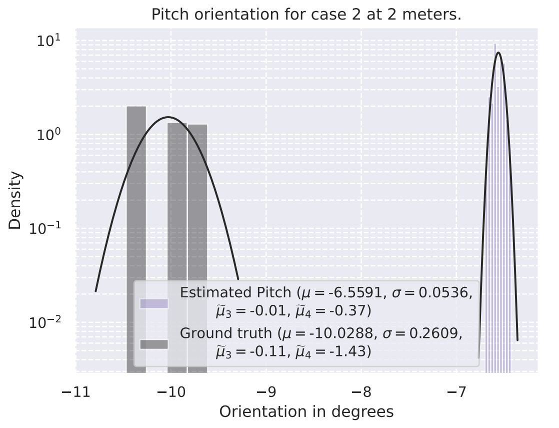


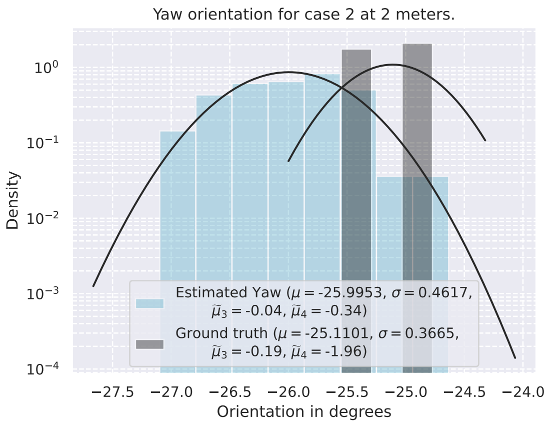

Yaw orientation for case 1 at 2 meters.
Density
Orientation in degrees
Estimated Yaw ($\mu = -19.8008$, $\sigma = 0.3805$, $\widetilde{\mu}_3 = -0.82$, $\widetilde{\mu}_4 = 0.71$)
Ground truth ($\mu = -18.6111$, $\sigma = 0.7565$, $\widetilde{\mu}_3 = 0.80$, $\widetilde{\mu}_4 = -0.81$)

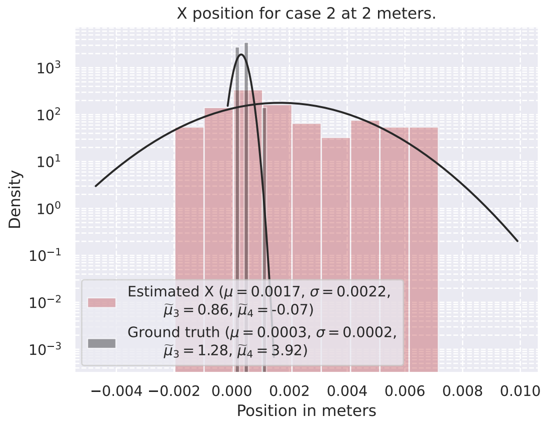

X position for case 1 at 2 meters.
Density
Position in meters
Estimated X ($\mu = -0.0940$, $\sigma = 0.0009$, $\widetilde{\mu}_3 = -0.08$, $\widetilde{\mu}_4 = 0.36$)
Ground truth ($\mu = -0.0929$, $\sigma = 0.0007$, $\widetilde{\mu}_3 = -0.03$, $\widetilde{\mu}_4 = -1.43$)

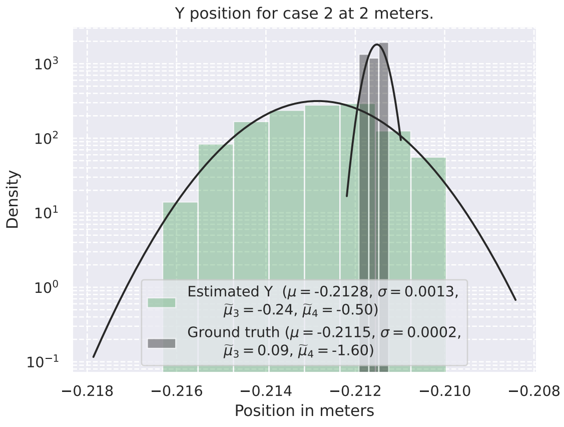

Y position for case 1 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2135$, $\sigma = 0.0011$, $\widetilde{\mu}_3 = -0.79$, $\widetilde{\mu}_4 = 0.60$)
Ground truth ($\mu = -0.2155$, $\sigma = 0.0004$, $\widetilde{\mu}_3 = -0.00$, $\widetilde{\mu}_4 = -0.71$)

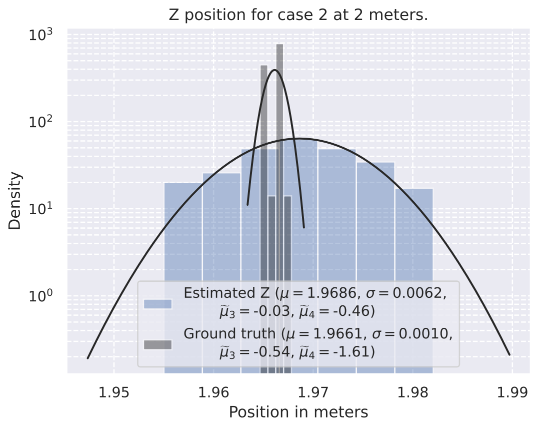

Z position for case 1 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.9805$, $\sigma = 0.0054$, $\widetilde{\mu}_3 = -0.06$, $\widetilde{\mu}_4 = -0.66$)
Ground truth ($\mu = 1.9715$, $\sigma = 0.0016$, $\widetilde{\mu}_3 = -0.39$, $\widetilde{\mu}_4 = -1.27$)

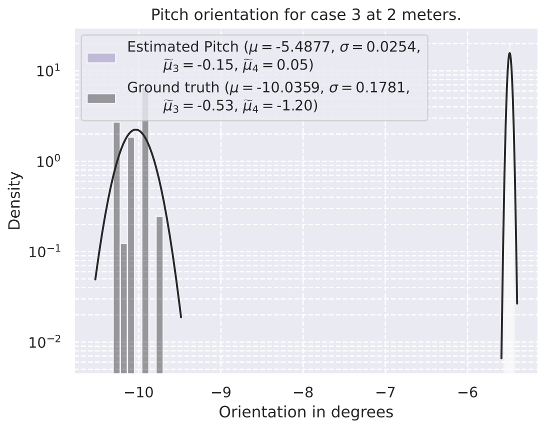

Pitch orientation for case 2 at 2 meters.
Density
Orientation in degrees
Estimated Pitch ($\mu = -6.5591$, $\sigma = 0.0536$, $\widetilde{\mu}_3 = -0.01$, $\widetilde{\mu}_4 = -0.37$)
Ground truth ($\mu = -10.0288$, $\sigma = 0.2609$, $\widetilde{\mu}_3 = -0.11$, $\widetilde{\mu}_4 = -1.43$)

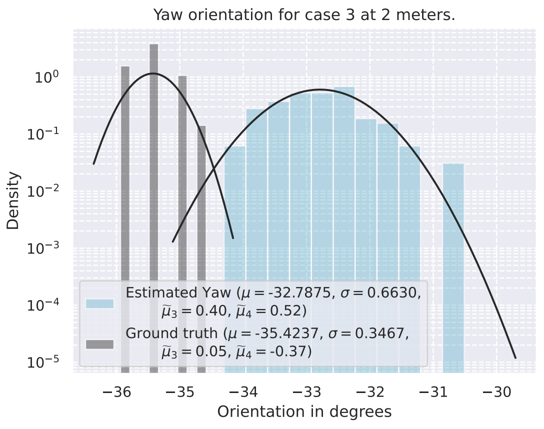

Yaw orientation for case 2 at 2 meters.
Density
Orientation in degrees
Estimated Yaw ($\mu = -25.9953$, $\sigma = 0.4617$, $\widetilde{\mu}_3 = -0.04$, $\widetilde{\mu}_4 = -0.34$)
Ground truth ($\mu = -25.1101$, $\sigma = 0.3665$, $\widetilde{\mu}_3 = -0.19$, $\widetilde{\mu}_4 = -1.96$)

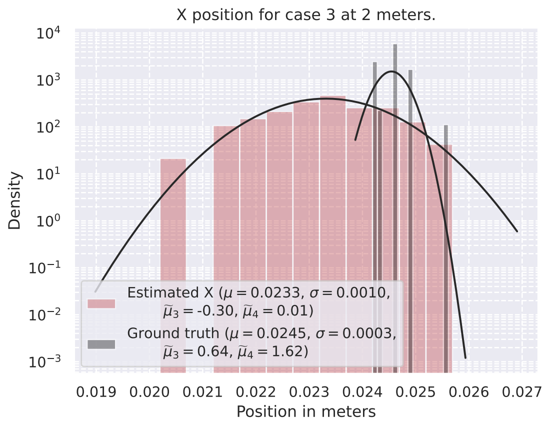

X position for case 2 at 2 meters.
Density
Position in meters
Estimated X ($\mu = 0.0017$, $\sigma = 0.0022$, $\widetilde{\mu}_3 = 0.86$, $\widetilde{\mu}_4 = -0.07$)
Ground truth ($\mu = 0.0003$, $\sigma = 0.0002$, $\widetilde{\mu}_3 = 1.28$, $\widetilde{\mu}_4 = 3.92$)

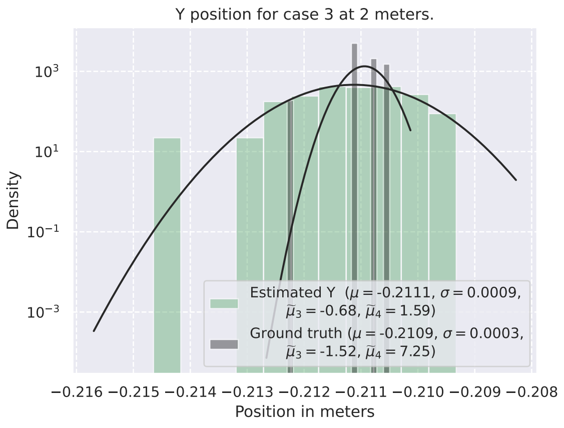

Y position for case 2 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2128$, $\sigma = 0.0013$, $\widetilde{\mu}_3 = -0.24$, $\widetilde{\mu}_4 = -0.50$)
Ground truth ($\mu = -0.2115$, $\sigma = 0.0002$, $\widetilde{\mu}_3 = 0.09$, $\widetilde{\mu}_4 = -1.60$)

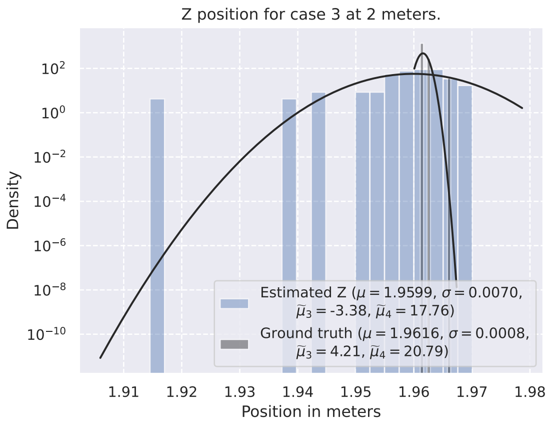

Z position for case 2 at 2 meters.
Estimated Z ($\mu = 1.9686$, $\sigma = 0.0062$, $\widetilde{\mu}_3 = -0.03$, $\widetilde{\mu}_4 = -0.46$)
Ground truth ($\mu = 1.9661$, $\sigma = 0.0010$, $\widetilde{\mu}_3 = -0.54$, $\widetilde{\mu}_4 = -1.61$)
Density
Position in meters

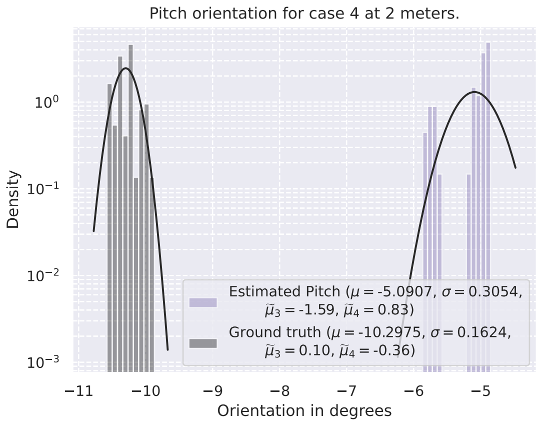

Pitch orientation for case 3 at 2 meters.
Estimated Pitch ($\mu = -5.4877$, $\sigma = 0.0254$, $\widetilde{\mu}_3 = -0.15$, $\widetilde{\mu}_4 = 0.05$)
Ground truth ($\mu = -10.0359$, $\sigma = 0.1781$, $\widetilde{\mu}_3 = -0.53$, $\widetilde{\mu}_4 = -1.20$)
Density
Orientation in degrees

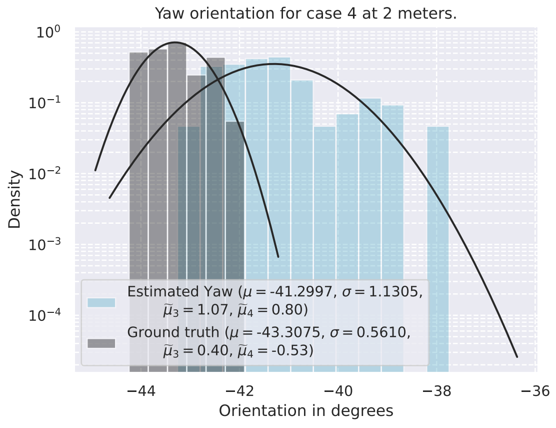

Yaw orientation for case 3 at 2 meters.
Estimated Yaw ($\mu = -32.7875$, $\sigma = 0.6630$, $\widetilde{\mu}_3 = 0.40$, $\widetilde{\mu}_4 = 0.52$)
Ground truth ($\mu = -35.4237$, $\sigma = 0.3467$, $\widetilde{\mu}_3 = 0.05$, $\widetilde{\mu}_4 = -0.37$)
Density
Orientation in degrees

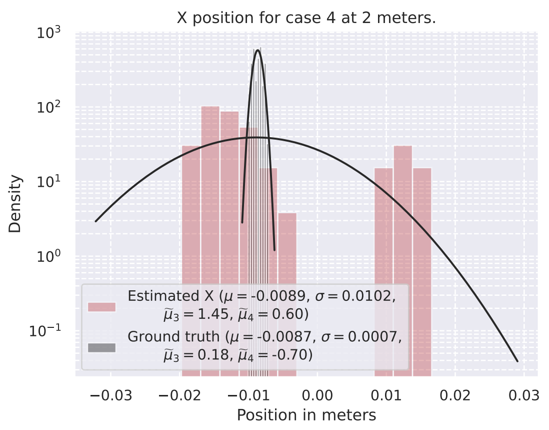

X position for case 3 at 2 meters.
Estimated X ($\mu = 0.0233$, $\sigma = 0.0010$, $\widetilde{\mu}_3 = -0.30$, $\widetilde{\mu}_4 = 0.01$)
Ground truth ($\mu = 0.0245$, $\sigma = 0.0003$, $\widetilde{\mu}_3 = 0.64$, $\widetilde{\mu}_4 = 1.62$)
Density
Position in meters

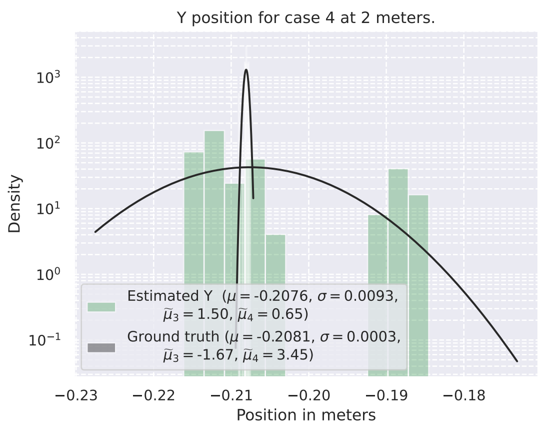

Y position for case 3 at 2 meters.
Estimated Y ($\mu = -0.2111$, $\sigma = 0.0009$, $\widetilde{\mu}_3 = -0.68$, $\widetilde{\mu}_4 = 1.59$)
Ground truth ($\mu = -0.2109$, $\sigma = 0.0003$, $\widetilde{\mu}_3 = -1.52$, $\widetilde{\mu}_4 = 7.25$)
Density
Position in meters

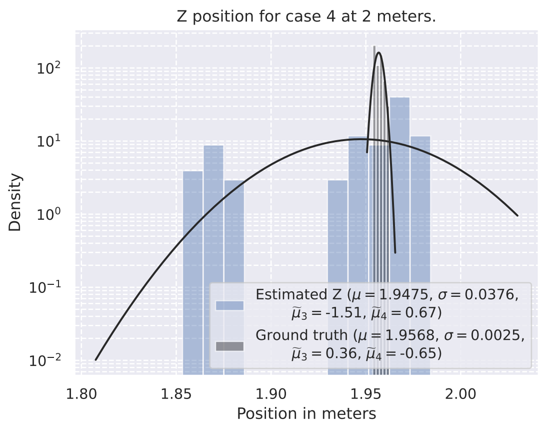

Z position for case 3 at 2 meters.
Estimated Z ($\mu = 1.9599$, $\sigma = 0.0070$, $\widetilde{\mu}_3 = -3.38$, $\widetilde{\mu}_4 = 17.76$)
Ground truth ($\mu = 1.9616$, $\sigma = 0.0008$, $\widetilde{\mu}_3 = 4.21$, $\widetilde{\mu}_4 = 20.79$)
Density
Position in meters

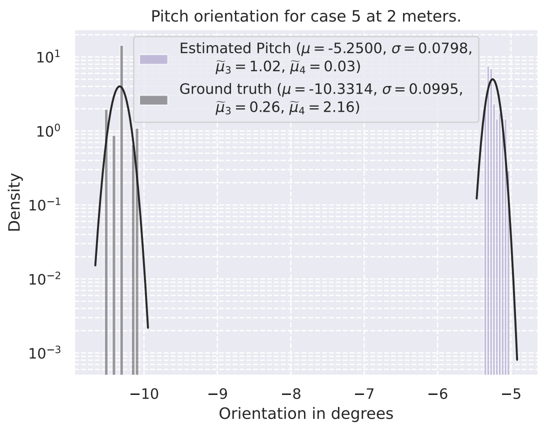

Pitch orientation for case 4 at 2 meters.
Estimated Pitch ($\mu = -5.0907$, $\sigma = 0.3054$, $\widetilde{\mu}_3 = -1.59$, $\widetilde{\mu}_4 = 0.83$)
Ground truth ($\mu = -10.2975$, $\sigma = 0.1624$, $\widetilde{\mu}_3 = 0.10$, $\widetilde{\mu}_4 = -0.36$)
Density
Orientation in degrees

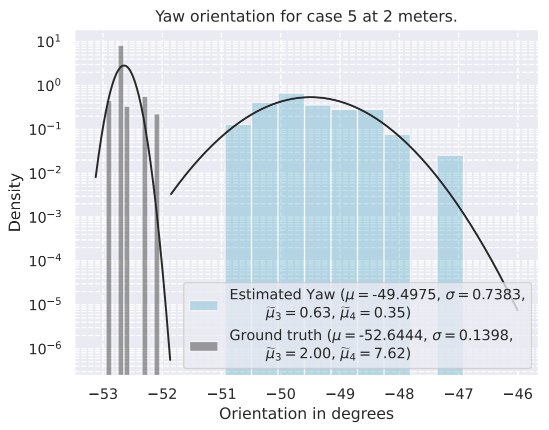

Yaw orientation for case 4 at 2 meters.
Estimated Yaw ($\mu = -41.2997$, $\sigma = 1.1305$, $\widetilde{\mu}_3 = 1.07$, $\widetilde{\mu}_4 = 0.80$)
Ground truth ($\mu = -43.3075$, $\sigma = 0.5610$, $\widetilde{\mu}_3 = 0.40$, $\widetilde{\mu}_4 = -0.53$)
Density
Orientation in degrees

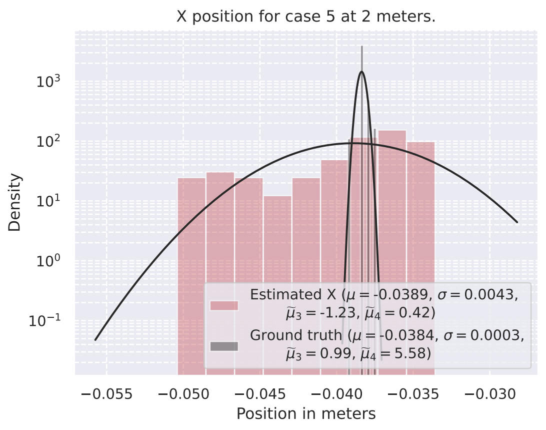

X position for case 4 at 2 meters.
Density
Position in meters
Estimated X ($\mu = -0.0089$, $\sigma = 0.0102$, $\tilde{\mu}_3 = 1.45$, $\tilde{\mu}_4 = 0.60$)
Ground truth ($\mu = -0.0087$, $\sigma = 0.0007$, $\tilde{\mu}_3 = 0.18$, $\tilde{\mu}_4 = -0.70$)

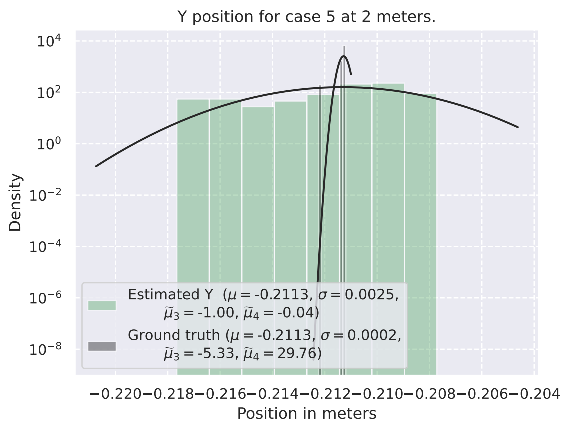

Y position for case 4 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2076$, $\sigma = 0.0093$, $\tilde{\mu}_3 = 1.50$, $\tilde{\mu}_4 = 0.65$)
Ground truth ($\mu = -0.2081$, $\sigma = 0.0003$, $\tilde{\mu}_3 = -1.67$, $\tilde{\mu}_4 = 3.45$)

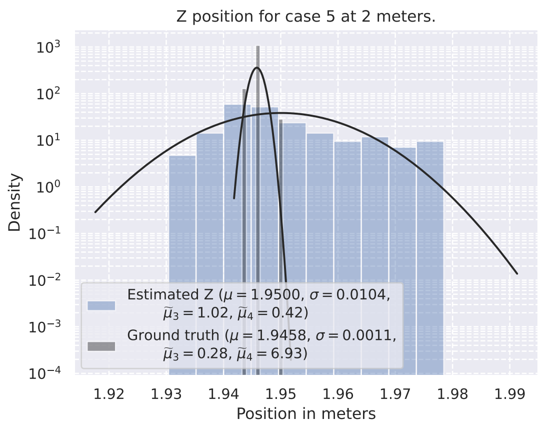

Z position for case 4 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.9475$, $\sigma = 0.0376$, $\tilde{\mu}_3 = -1.51$, $\tilde{\mu}_4 = 0.67$)
Ground truth ($\mu = 1.9568$, $\sigma = 0.0025$, $\tilde{\mu}_3 = 0.36$, $\tilde{\mu}_4 = -0.65$)

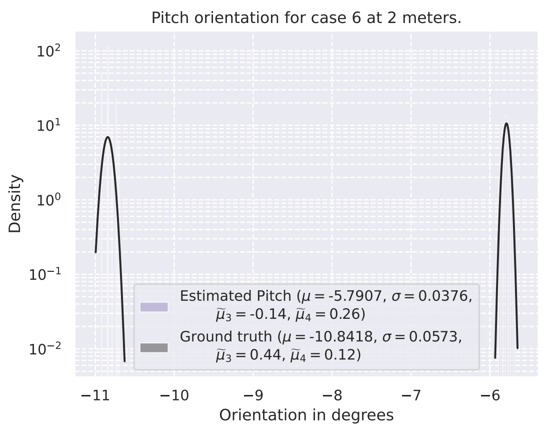

Pitch orientation for case 5 at 2 meters.
Density
Orientation in degrees
Estimated Pitch ($\mu = -5.2500$, $\sigma = 0.0798$, $\tilde{\mu}_3 = 1.02$, $\tilde{\mu}_4 = 0.03$)
Ground truth ($\mu = -10.3314$, $\sigma = 0.0995$, $\tilde{\mu}_3 = 0.26$, $\tilde{\mu}_4 = 2.16$)

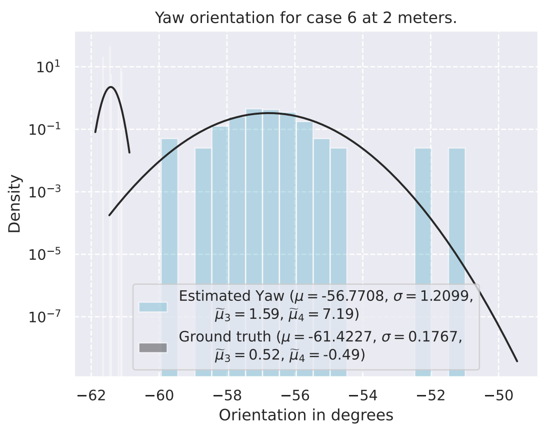

Yaw orientation for case 5 at 2 meters.
Density
Orientation in degrees
Estimated Yaw ($\mu = -49.4975$, $\sigma = 0.7383$, $\tilde{\mu}_3 = 0.63$, $\tilde{\mu}_4 = 0.35$)
Ground truth ($\mu = -52.6444$, $\sigma = 0.1398$, $\tilde{\mu}_3 = 2.00$, $\tilde{\mu}_4 = 7.62$)

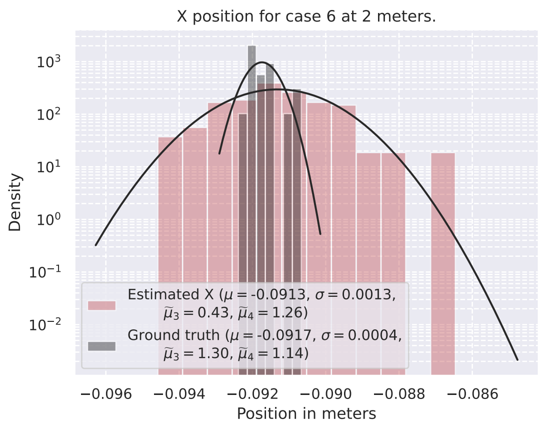

X position for case 5 at 2 meters.
Density
Position in meters
Estimated X ($\mu = -0.0389$, $\sigma = 0.0043$, $\tilde{\mu}_3 = -1.23$, $\tilde{\mu}_4 = 0.42$)
Ground truth ($\mu = -0.0384$, $\sigma = 0.0003$, $\tilde{\mu}_3 = 0.99$, $\tilde{\mu}_4 = 5.58$)

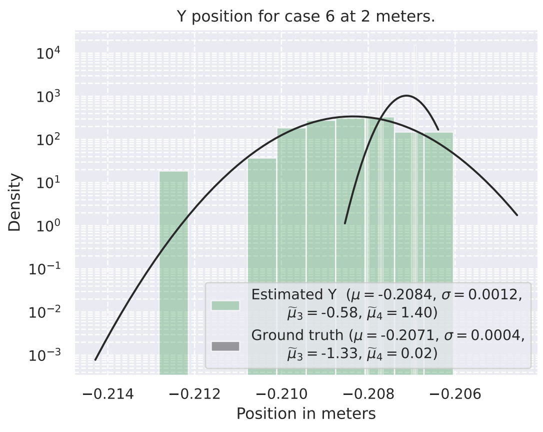

Y position for case 5 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2113$, $\sigma = 0.0025$, $\tilde{\mu}_3 = -1.00$, $\tilde{\mu}_4 = -0.04$)
Ground truth ($\mu = -0.2113$, $\sigma = 0.0002$, $\tilde{\mu}_3 = -5.33$, $\tilde{\mu}_4 = 29.76$)

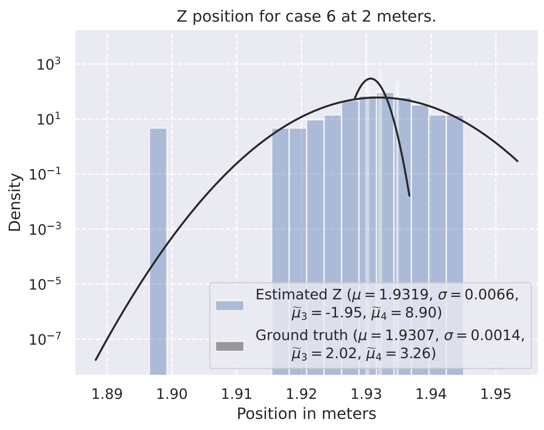

Z position for case 5 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.9500$, $\sigma = 0.0104$, $\tilde{\mu}_3 = 1.02$, $\tilde{\mu}_4 = 0.42$)
Ground truth ($\mu = 1.9458$, $\sigma = 0.0011$, $\tilde{\mu}_3 = 0.28$, $\tilde{\mu}_4 = 6.93$)

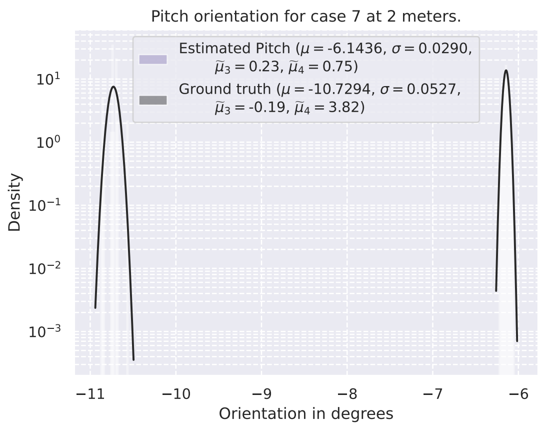

Pitch orientation for case 6 at 2 meters.
Estimated Pitch ($\mu = $ -5.7907, $\sigma = 0.0376$, $\tilde{\mu}_3 = $ -0.14, $\tilde{\mu}_4 = 0.26$)
Ground truth ($\mu = $ -10.8418, $\sigma = 0.0573$, $\tilde{\mu}_3 = 0.44$, $\tilde{\mu}_4 = 0.12$)
Density
Orientation in degrees

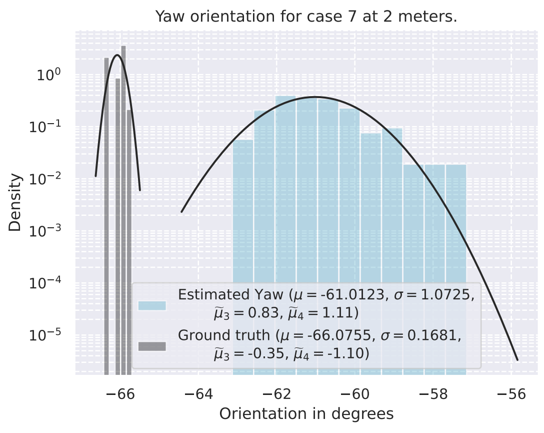

Yaw orientation for case 6 at 2 meters.
Estimated Yaw ($\mu = $ -56.7708, $\sigma = 1.2099$, $\tilde{\mu}_3 = 1.59$, $\tilde{\mu}_4 = 7.19$)
Ground truth ($\mu = $ -61.4227, $\sigma = 0.1767$, $\tilde{\mu}_3 = 0.52$, $\tilde{\mu}_4 = $ -0.49)
Density
Orientation in degrees

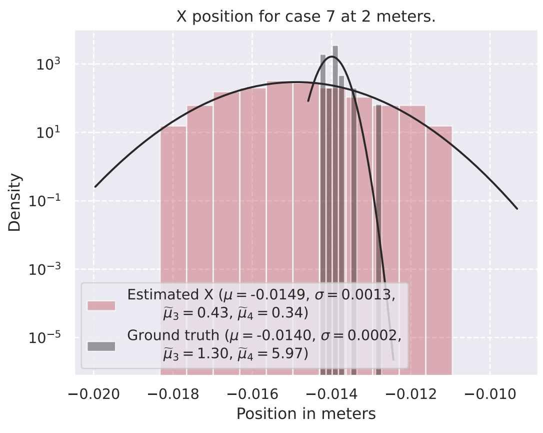

X position for case 6 at 2 meters.
Estimated X ($\mu = $ -0.0913, $\sigma = 0.0013$, $\tilde{\mu}_3 = 0.43$, $\tilde{\mu}_4 = 1.26$)
Ground truth ($\mu = $ -0.0917, $\sigma = 0.0004$, $\tilde{\mu}_3 = 1.30$, $\tilde{\mu}_4 = 1.14$)
Density
Position in meters

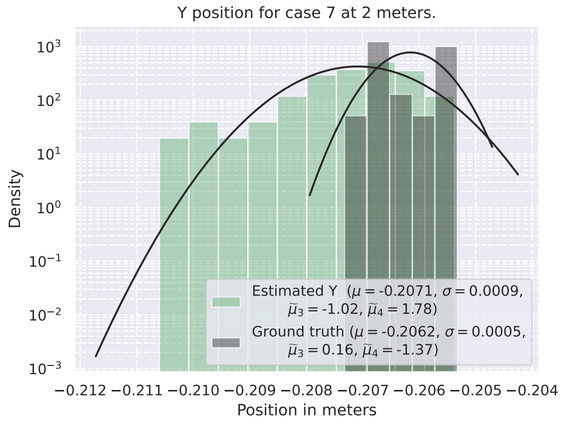

Y position for case 6 at 2 meters.
Estimated Y ($\mu = $ -0.2084, $\sigma = 0.0012$, $\tilde{\mu}_3 = $ -0.58, $\tilde{\mu}_4 = 1.40$)
Ground truth ($\mu = $ -0.2071, $\sigma = 0.0004$, $\tilde{\mu}_3 = $ -1.33, $\tilde{\mu}_4 = 0.02$)
Density
Position in meters

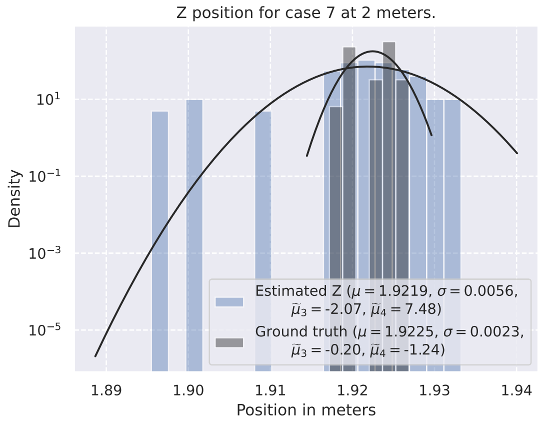

Z position for case 6 at 2 meters.
Estimated Z ($\mu = 1.9319$, $\sigma = 0.0066$, $\tilde{\mu}_3 = $ -1.95, $\tilde{\mu}_4 = 8.90$)
Ground truth ($\mu = 1.9307$, $\sigma = 0.0014$, $\tilde{\mu}_3 = 2.02$, $\tilde{\mu}_4 = 3.26$)
Density
Position in meters

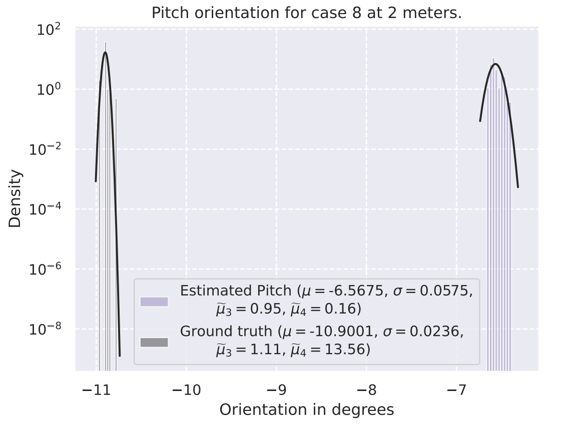

Pitch orientation for case 7 at 2 meters.
Estimated Pitch ($\mu = $ -6.1436, $\sigma = 0.0290$, $\tilde{\mu}_3 = 0.23$, $\tilde{\mu}_4 = 0.75$)
Ground truth ($\mu = $ -10.7294, $\sigma = 0.0527$, $\tilde{\mu}_3 = $ -0.19, $\tilde{\mu}_4 = 3.82$)
Density
Orientation in degrees

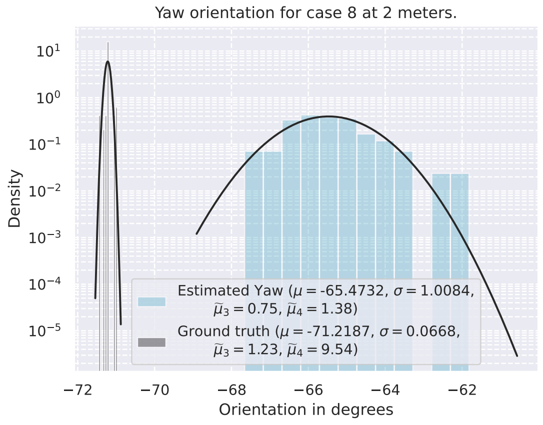

Yaw orientation for case 7 at 2 meters.
Estimated Yaw ($\mu = $ -61.0123, $\sigma = 1.0725$, $\tilde{\mu}_3 = 0.83$, $\tilde{\mu}_4 = 1.11$)
Ground truth ($\mu = $ -66.0755, $\sigma = 0.1681$, $\tilde{\mu}_3 = $ -0.35, $\tilde{\mu}_4 = $ -1.10)
Density
Orientation in degrees

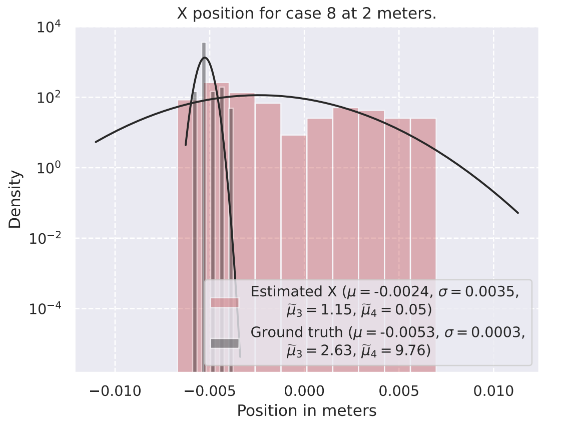

X position for case 7 at 2 meters.
Estimated X ($\mu = $ -0.0149, $\sigma = 0.0013$, $\tilde{\mu}_3 = 0.43$, $\tilde{\mu}_4 = 0.34$)
Ground truth ($\mu = $ -0.0140, $\sigma = 0.0002$, $\tilde{\mu}_3 = 1.30$, $\tilde{\mu}_4 = 5.97$)
Density
Position in meters

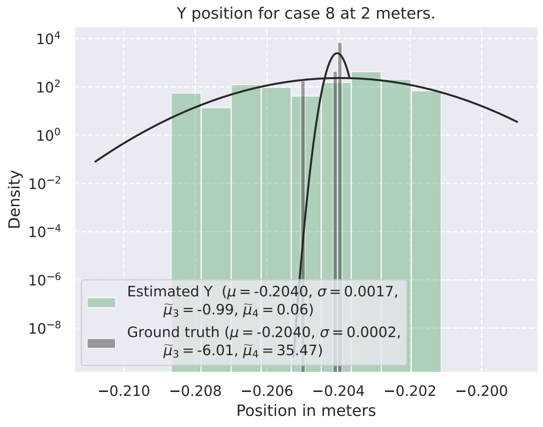

Y position for case 7 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2071$, $\sigma = 0.0009$, $\tilde{\mu}_3 = -1.02$, $\tilde{\mu}_4 = 1.78$)
Ground truth ($\mu = -0.2062$, $\sigma = 0.0005$, $\tilde{\mu}_3 = 0.16$, $\tilde{\mu}_4 = -1.37$)

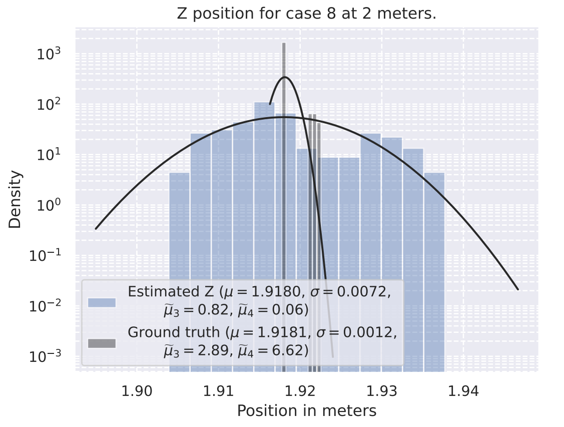

Z position for case 7 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.9219$, $\sigma = 0.0056$, $\tilde{\mu}_3 = -2.07$, $\tilde{\mu}_4 = 7.48$)
Ground truth ($\mu = 1.9225$, $\sigma = 0.0023$, $\tilde{\mu}_3 = -0.20$, $\tilde{\mu}_4 = -1.24$)

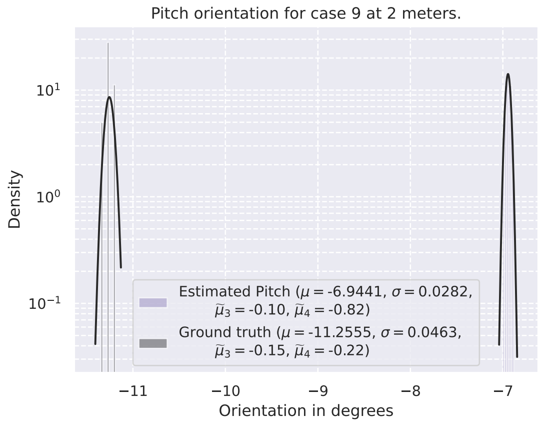

Pitch orientation for case 8 at 2 meters.
Density
Orientation in degrees
Estimated Pitch ($\mu = -6.5675$, $\sigma = 0.0575$, $\tilde{\mu}_3 = 0.95$, $\tilde{\mu}_4 = 0.16$)
Ground truth ($\mu = -10.9001$, $\sigma = 0.0236$, $\tilde{\mu}_3 = 1.11$, $\tilde{\mu}_4 = 13.56$)

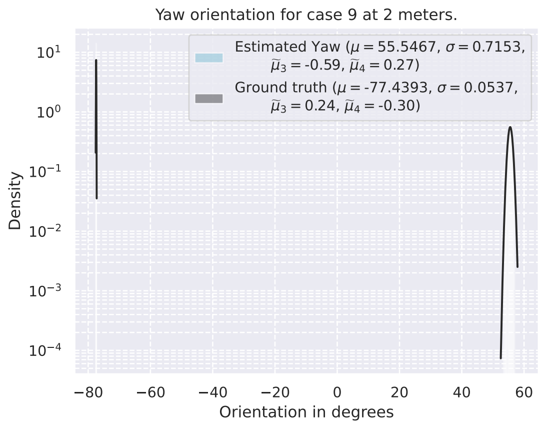

Yaw orientation for case 8 at 2 meters.
Density
Orientation in degrees
Estimated Yaw ($\mu = -65.4732$, $\sigma = 1.0084$, $\tilde{\mu}_3 = 0.75$, $\tilde{\mu}_4 = 1.38$)
Ground truth ($\mu = -71.2187$, $\sigma = 0.0668$, $\tilde{\mu}_3 = 1.23$, $\tilde{\mu}_4 = 9.54$)

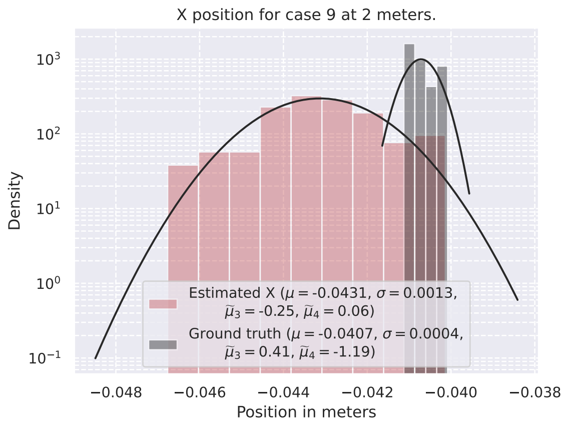

X position for case 8 at 2 meters.
Density
Position in meters
Estimated X ($\mu = -0.0024$, $\sigma = 0.0035$, $\tilde{\mu}_3 = 1.15$, $\tilde{\mu}_4 = 0.05$)
Ground truth ($\mu = -0.0053$, $\sigma = 0.0003$, $\tilde{\mu}_3 = 2.63$, $\tilde{\mu}_4 = 9.76$)

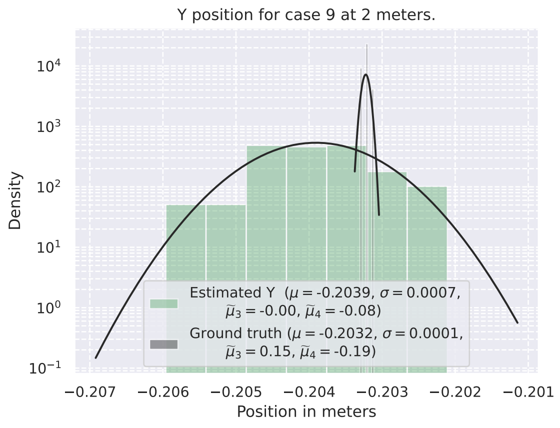

Y position for case 8 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2040$, $\sigma = 0.0017$, $\tilde{\mu}_3 = -0.99$, $\tilde{\mu}_4 = 0.06$)
Ground truth ($\mu = -0.2040$, $\sigma = 0.0002$, $\tilde{\mu}_3 = -6.01$, $\tilde{\mu}_4 = 35.47$)

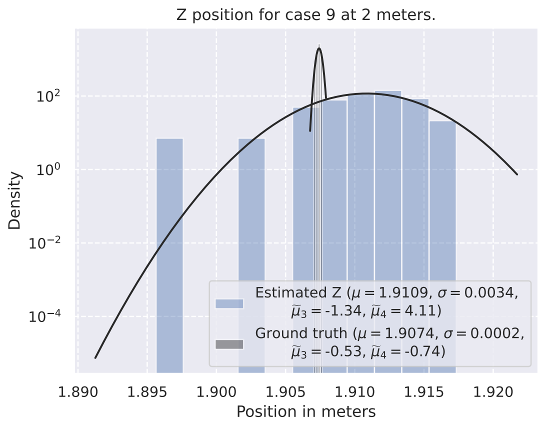

Z position for case 8 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.9180$, $\sigma = 0.0072$, $\tilde{\mu}_3 = 0.82$, $\tilde{\mu}_4 = 0.06$)
Ground truth ($\mu = 1.9181$, $\sigma = 0.0012$, $\tilde{\mu}_3 = 2.89$, $\tilde{\mu}_4 = 6.62$)

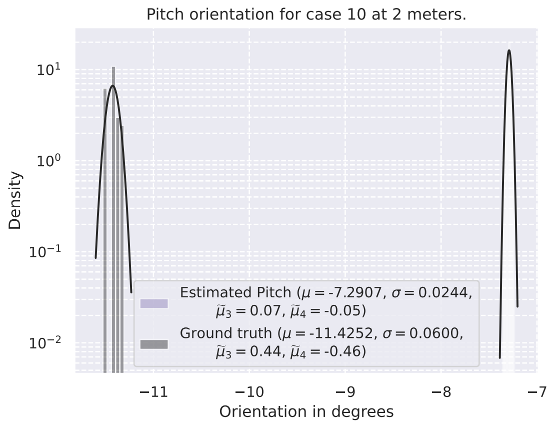

Pitch orientation for case 9 at 2 meters.
Density
Orientation in degrees
Estimated Pitch ($\mu = -6.9441$, $\sigma = 0.0282$, $\tilde{\mu}_3 = -0.10$, $\tilde{\mu}_4 = -0.82$)
Ground truth ($\mu = -11.2555$, $\sigma = 0.0463$, $\tilde{\mu}_3 = -0.15$, $\tilde{\mu}_4 = -0.22$)

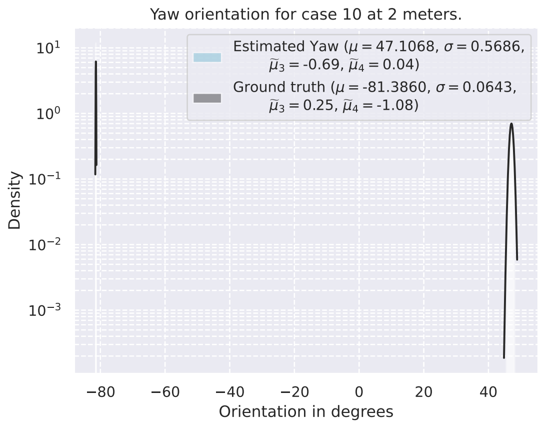

Yaw orientation for case 9 at 2 meters.
Estimated Yaw ($\mu = 55.5467$, $\sigma = 0.7153$, $\tilde{\mu}_3 = -0.59$, $\tilde{\mu}_4 = 0.27$)
Ground truth ($\mu = -77.4393$, $\sigma = 0.0537$, $\tilde{\mu}_3 = 0.24$, $\tilde{\mu}_4 = -0.30$)
Density
Orientation in degrees

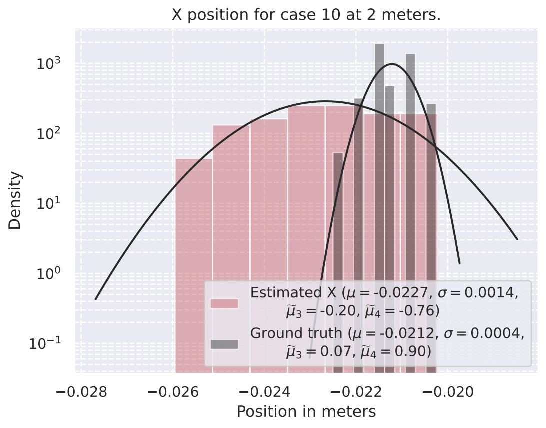

X position for case 9 at 2 meters.
Estimated X ($\mu = -0.0431$, $\sigma = 0.0013$, $\tilde{\mu}_3 = -0.25$, $\tilde{\mu}_4 = 0.06$)
Ground truth ($\mu = -0.0407$, $\sigma = 0.0004$, $\tilde{\mu}_3 = 0.41$, $\tilde{\mu}_4 = -1.19$)
Density
Position in meters

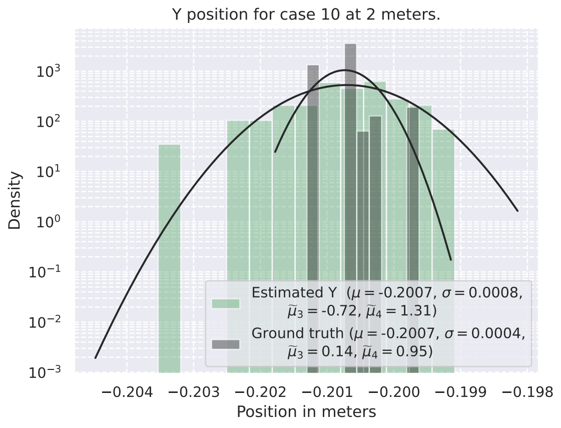

Y position for case 9 at 2 meters.
Estimated Y ($\mu = -0.2039$, $\sigma = 0.0007$, $\tilde{\mu}_3 = -0.00$, $\tilde{\mu}_4 = -0.08$)
Ground truth ($\mu = -0.2032$, $\sigma = 0.0001$, $\tilde{\mu}_3 = 0.15$, $\tilde{\mu}_4 = -0.19$)
Density
Position in meters

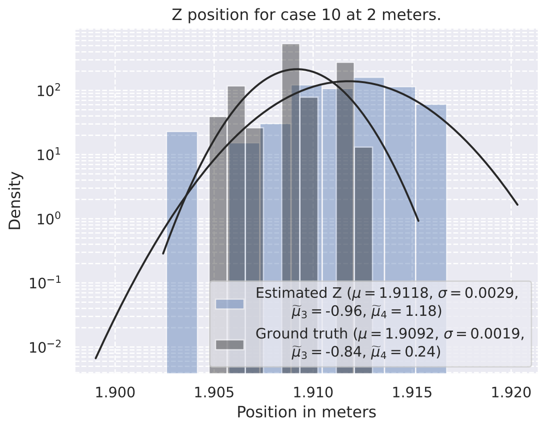

Z position for case 9 at 2 meters.
Estimated Z ($\mu = 1.9109$, $\sigma = 0.0034$, $\tilde{\mu}_3 = -1.34$, $\tilde{\mu}_4 = 4.11$)
Ground truth ($\mu = 1.9074$, $\sigma = 0.0002$, $\tilde{\mu}_3 = -0.53$, $\tilde{\mu}_4 = -0.74$)
Density
Position in meters


Pitch orientation for case 10 at 2 meters.
Estimated Pitch ($\mu = -7.2907$, $\sigma = 0.0244$, $\tilde{\mu}_3 = 0.07$, $\tilde{\mu}_4 = -0.05$)
Ground truth ($\mu = -11.4252$, $\sigma = 0.0600$, $\tilde{\mu}_3 = 0.44$, $\tilde{\mu}_4 = -0.46$)
Density
Orientation in degrees

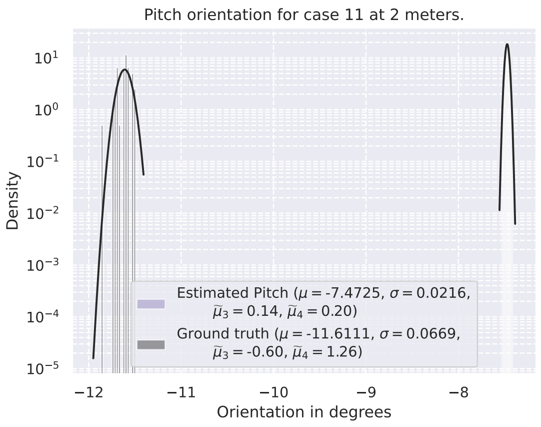

Yaw orientation for case 10 at 2 meters.
Estimated Yaw ($\mu = 47.1068$, $\sigma = 0.5686$, $\tilde{\mu}_3 = -0.69$, $\tilde{\mu}_4 = 0.04$)
Ground truth ($\mu = -81.3860$, $\sigma = 0.0643$, $\tilde{\mu}_3 = 0.25$, $\tilde{\mu}_4 = -1.08$)
Density
Orientation in degrees


X position for case 10 at 2 meters.
Estimated X ($\mu = -0.0227$, $\sigma = 0.0014$, $\tilde{\mu}_3 = -0.20$, $\tilde{\mu}_4 = -0.76$)
Ground truth ($\mu = -0.0212$, $\sigma = 0.0004$, $\tilde{\mu}_3 = 0.07$, $\tilde{\mu}_4 = 0.90$)
Density
Position in meters

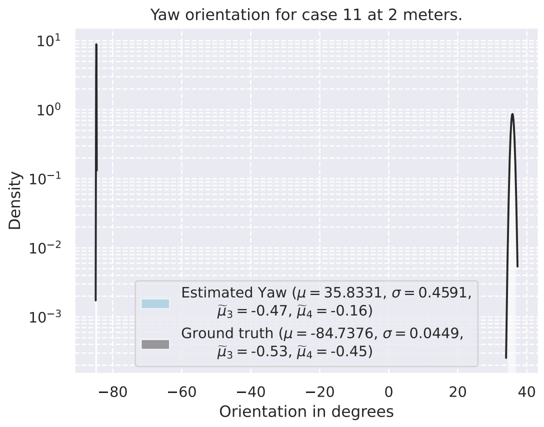

Y position for case 10 at 2 meters.
Estimated Y ($\mu = -0.2007$, $\sigma = 0.0008$, $\tilde{\mu}_3 = -0.72$, $\tilde{\mu}_4 = 1.31$)
Ground truth ($\mu = -0.2007$, $\sigma = 0.0004$, $\tilde{\mu}_3 = 0.14$, $\tilde{\mu}_4 = 0.95$)
Density
Position in meters



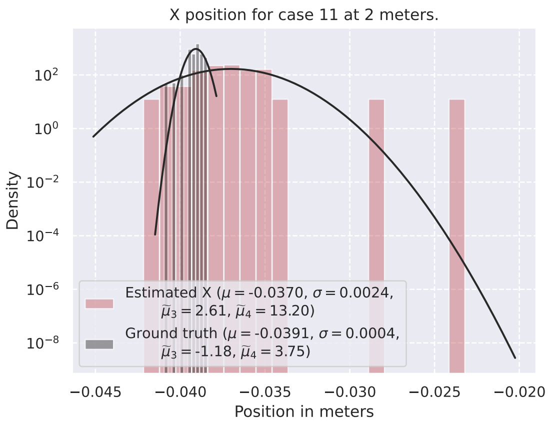
Z position for case 10 at 2 meters.
Estimated Z ($\mu = 1.9118$, $\sigma = 0.0029$, $\tilde{\mu}_3 = -0.96$, $\tilde{\mu}_4 = 1.18$)
Ground truth ($\mu = 1.9092$, $\sigma = 0.0019$, $\tilde{\mu}_3 = -0.84$, $\tilde{\mu}_4 = 0.24$)
Density
Position in meters



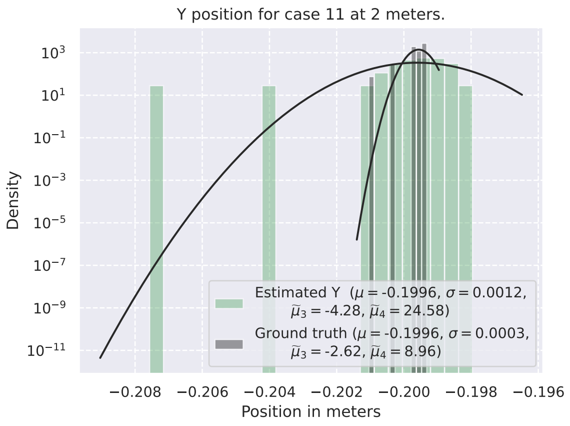
Pitch orientation for case 11 at 2 meters.
Estimated Pitch ($\mu = -7.4725$, $\sigma = 0.0216$, $\tilde{\mu}_3 = 0.14$, $\tilde{\mu}_4 = 0.20$)
Ground truth ($\mu = -11.6111$, $\sigma = 0.0669$, $\tilde{\mu}_3 = -0.60$, $\tilde{\mu}_4 = 1.26$)
Density
Orientation in degrees



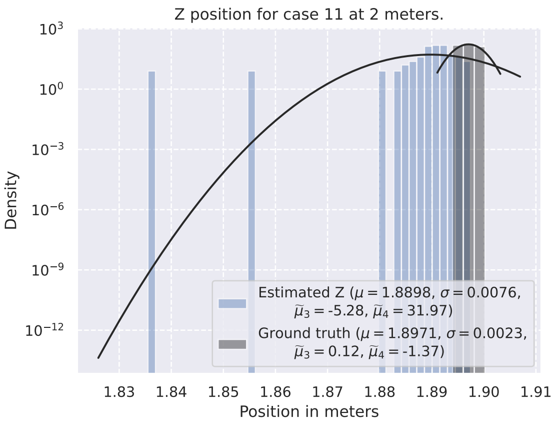
Yaw orientation for case 11 at 2 meters.
Estimated Yaw ($\mu = 35.8331$, $\sigma = 0.4591$, $\tilde{\mu}_3 = -0.47$, $\tilde{\mu}_4 = -0.16$)
Ground truth ($\mu = -84.7376$, $\sigma = 0.0449$, $\tilde{\mu}_3 = -0.53$, $\tilde{\mu}_4 = -0.45$)
Density
Orientation in degrees



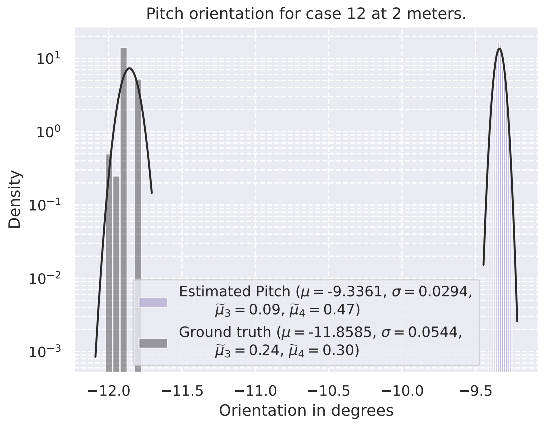
X position for case 11 at 2 meters.
Estimated X ($\mu = -0.0370$, $\sigma = 0.0024$, $\tilde{\mu}_3 = 2.61$, $\tilde{\mu}_4 = 13.20$)
Ground truth ($\mu = -0.0391$, $\sigma = 0.0004$, $\tilde{\mu}_3 = -1.18$, $\tilde{\mu}_4 = 3.75$)
Density
Position in meters



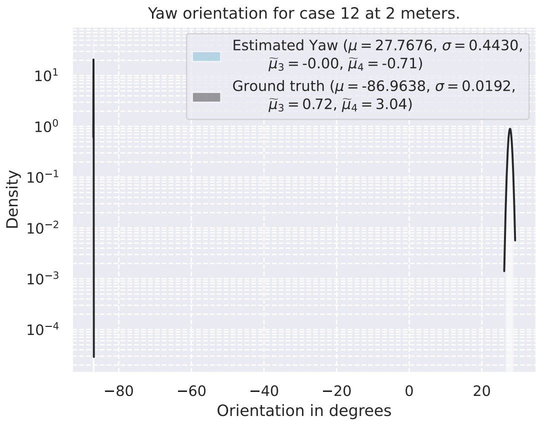
Y position for case 11 at 2 meters.
Estimated Y ($\mu = -0.1996$, $\sigma = 0.0012$, $\tilde{\mu}_3 = -4.28$, $\tilde{\mu}_4 = 24.58$)
Ground truth ($\mu = -0.1996$, $\sigma = 0.0003$, $\tilde{\mu}_3 = -2.62$, $\tilde{\mu}_4 = 8.96$)
Density
Position in meters



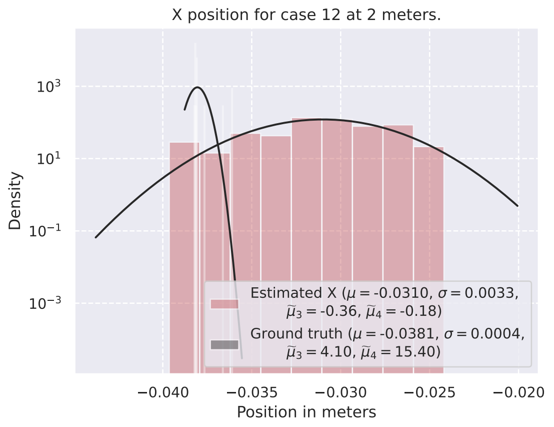
Z position for case 11 at 2 meters.
Estimated Z ($\mu = 1.8898$, $\sigma = 0.0076$, $\tilde{\mu}_3 = -5.28$, $\tilde{\mu}_4 = 31.97$)
Ground truth ($\mu = 1.8971$, $\sigma = 0.0023$, $\tilde{\mu}_3 = 0.12$, $\tilde{\mu}_4 = -1.37$)
Density
Position in meters

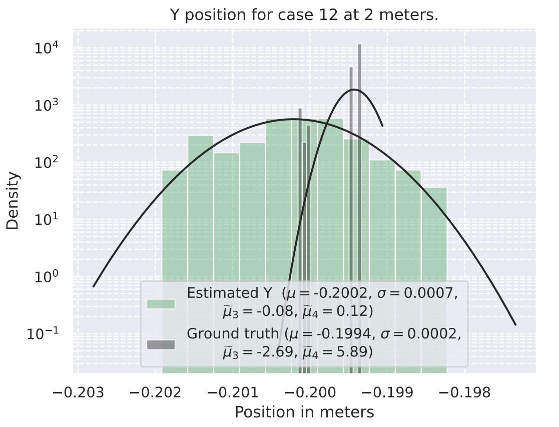


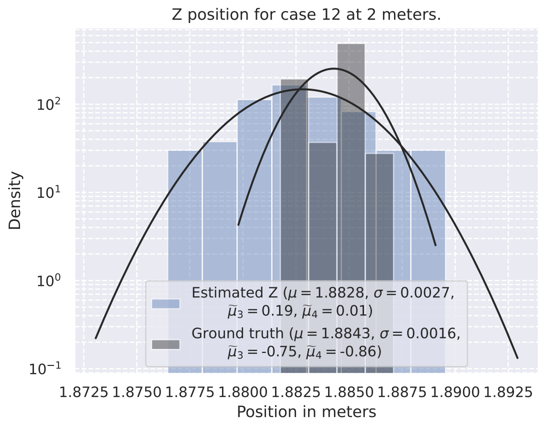
Pitch orientation for case 12 at 2 meters.
Estimated Pitch ($\mu = -9.3361$, $\sigma = 0.0294$, $\tilde{\mu}_3 = 0.09$, $\tilde{\mu}_4 = 0.47$)
Ground truth ($\mu = -11.8585$, $\sigma = 0.0544$, $\tilde{\mu}_3 = 0.24$, $\tilde{\mu}_4 = 0.30$)
Density
Orientation in degrees

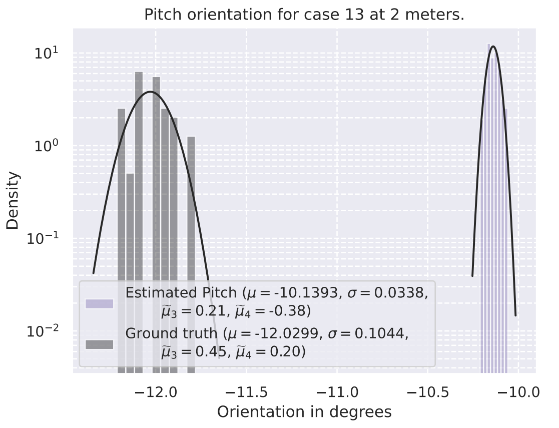


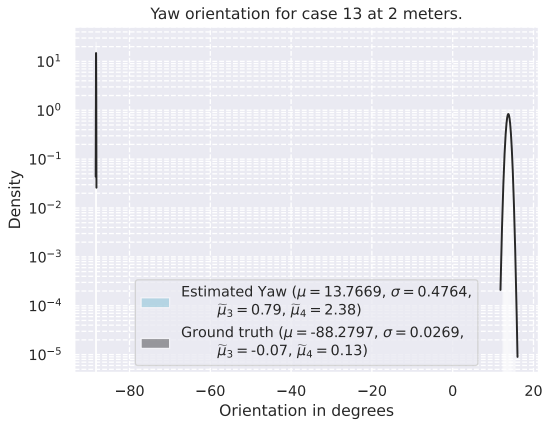
Yaw orientation for case 12 at 2 meters.
Estimated Yaw ($\mu = 27.7676$, $\sigma = 0.4430$, $\tilde{\mu}_3 = -0.00$, $\tilde{\mu}_4 = -0.71$)
Ground truth ($\mu = -86.9638$, $\sigma = 0.0192$, $\tilde{\mu}_3 = 0.72$, $\tilde{\mu}_4 = 3.04$)
Density
Orientation in degrees

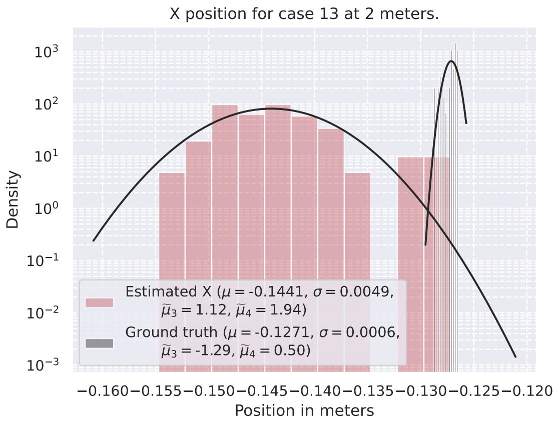

X position for case 12 at 2 meters.
Density
Position in meters
Estimated X ($\mu = -0.0310$, $\sigma = 0.0033$, $\tilde{\mu}_3 = -0.36$, $\tilde{\mu}_4 = -0.18$)
Ground truth ($\mu = -0.0381$, $\sigma = 0.0004$, $\tilde{\mu}_3 = 4.10$, $\tilde{\mu}_4 = 15.40$)

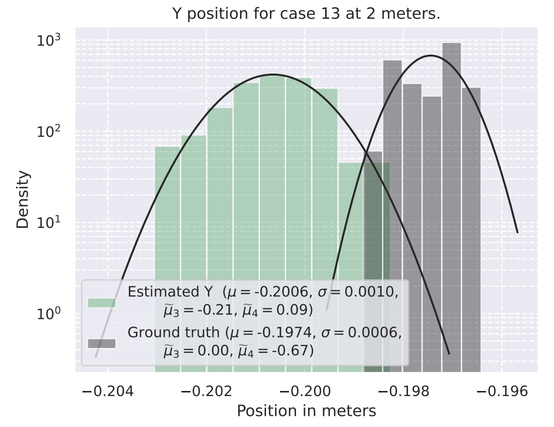

Y position for case 12 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2002$, $\sigma = 0.0007$, $\tilde{\mu}_3 = -0.08$, $\tilde{\mu}_4 = 0.12$)
Ground truth ($\mu = -0.1994$, $\sigma = 0.0002$, $\tilde{\mu}_3 = -2.69$, $\tilde{\mu}_4 = 5.89$)

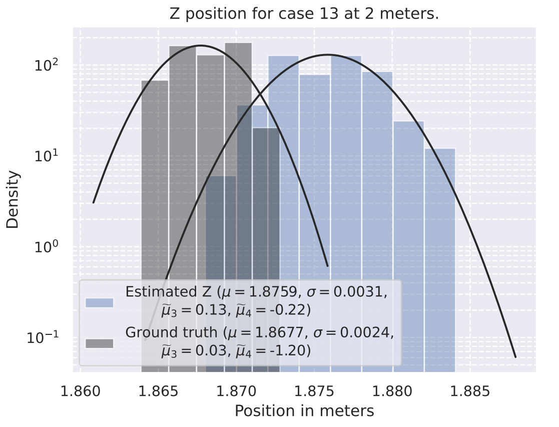

Z position for case 12 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.8828$, $\sigma = 0.0027$, $\tilde{\mu}_3 = 0.19$, $\tilde{\mu}_4 = 0.01$)
Ground truth ($\mu = 1.8843$, $\sigma = 0.0016$, $\tilde{\mu}_3 = -0.75$, $\tilde{\mu}_4 = -0.86$)


Pitch orientation for case 13 at 2 meters.
Density
Orientation in degrees
Estimated Pitch ($\mu = -10.1393$, $\sigma = 0.0338$, $\tilde{\mu}_3 = 0.21$, $\tilde{\mu}_4 = -0.38$)
Ground truth ($\mu = -12.0299$, $\sigma = 0.1044$, $\tilde{\mu}_3 = 0.45$, $\tilde{\mu}_4 = 0.20$)


Yaw orientation for case 13 at 2 meters.
Density
Orientation in degrees
Estimated Yaw ($\mu = 13.7669$, $\sigma = 0.4764$, $\tilde{\mu}_3 = 0.79$, $\tilde{\mu}_4 = 2.38$)
Ground truth ($\mu = -88.2797$, $\sigma = 0.0269$, $\tilde{\mu}_3 = -0.07$, $\tilde{\mu}_4 = 0.13$)


X position for case 13 at 2 meters.
Density
Position in meters
Estimated X ($\mu = -0.1441$, $\sigma = 0.0049$, $\tilde{\mu}_3 = 1.12$, $\tilde{\mu}_4 = 1.94$)
Ground truth ($\mu = -0.1271$, $\sigma = 0.0006$, $\tilde{\mu}_3 = -1.29$, $\tilde{\mu}_4 = 0.50$)


Y position for case 13 at 2 meters.
Density
Position in meters
Estimated Y ($\mu = -0.2006$, $\sigma = 0.0010$, $\tilde{\mu}_3 = -0.21$, $\tilde{\mu}_4 = 0.09$)
Ground truth ($\mu = -0.1974$, $\sigma = 0.0006$, $\tilde{\mu}_3 = 0.00$, $\tilde{\mu}_4 = -0.67$)


Z position for case 13 at 2 meters.
Density
Position in meters
Estimated Z ($\mu = 1.8759$, $\sigma = 0.0031$, $\tilde{\mu}_3 = 0.13$, $\tilde{\mu}_4 = -0.22$)
Ground truth ($\mu = 1.8677$, $\sigma = 0.0024$, $\tilde{\mu}_3 = 0.03$, $\tilde{\mu}_4 = -1.20$)

D. Code Listings

D.1 Code Listings for Object Detection

```

class Exp(MyExp):
    def __init__(self):
        super(Exp, self).__init__()
        self.depth = 0.33
        self.width = 0.50
        self.exp_name = ...
            os.path.split(os.path.realpath(__file__))[1].split(".")[0]

        # Define dataset path
        self.data_dir = "datasets/loco2020_only_pallet_1245"
        self.train_ann = "instances_Train.json"
        self.val_ann = "instances_Validation.json"
        self.test_ann = "instances_default.json"

        self.num_classes = 1

        self.max_epoch = 300
        self.data_num_workers = 4
        self.eval_interval = 2
```

Listing D.1: YOLOX exp-file with the LOCO dataset training on pallets.

```
python tools/train.py -f exps/example/custom/yolox\_s\_loco\_cvat\_1245.py ...
    -d 0 -b 8 --fp16 -o -c models/yolo\_s\_only\_pallet\_mars.pth
```

Listing D.2: Running train.py with input arguments.

```
python3 tools/export\_onnx.py --output-name ...
    yolox\_s\_pallet\_1245\_o10.onnx -o 10 -f ...
    exps/example/custom/yolox\_s\_loco\_cvat\_1245.py -c ...
    YOLOX\_outputs/yolox\_s\_loco\_cvat\_1245/best\_ckpt.pth
```

Listing D.3: Export to ONNX with opt version 10.

```
python3 mo --input_model models/yolox_s_only_pallet_294epoch_o10.onnx
```

Listing D.4: Converting ONNX to OpenVINO.

```
object_detection_object_.set_model_path(
"models/yolox_s_only_pallet_294epoch_o10/yolox_s_only_pallet_294epoch_o10.xml");

object_detection_object_.set_object_detection_settings(0.3,0.75);
object_detection_object_.setup_object_detection();
```

Listing D.5: Object detection configuration.

D.2 Code Listings for Pose Estimation

```
    if (auto p = dev.as<rs2::playback>()){
      p.set_real_time(0); //! Doesn't skip frames.
    }
```

Listing D.6: Enable realtime playback if playing from ROS-bag.

```
double angle_zy = std::acos((z_inverse.dot(center_frustum_zy))/
                            (z_inverse.norm()*center_frustum_zy.norm()));
double angle_zx = std::acos((z_inverse.dot(center_frustum_zx))/
                            (z_inverse.norm()*center_frustum_zx.norm()));
```

Listing D.7: Calculation of camera pose.

```
  fov_h_rad_ = std::acos(dot_01/(len_sq_0*len_sq_1));
  fov_v_rad_ = std::acos(dot_02/(len_sq_0*len_sq_2));
```

Listing D.8: Calculation of field-of-view.

```
pcl::FrustumCulling<pcl::PointXYZ> frustum_filter;


frustum_filter.setInputCloud(local_cloud);
frustum_filter.setCameraPose(camera_pose*rotation_red_pos_ccw*
                             rotation_green_pos_cw);
frustum_filter.setNearPlaneDistance(0);
frustum_filter.setFarPlaneDistance(15);
frustum_filter.setVerticalFOV(fov_v_rad_ * 57.2958);
frustum_filter.setHorizontalFOV(fov_h_rad_ * 57.2958);
frustum_filter.filter(*local_pallet);
frustum_filter.filter(frustum_filter_inliers_);
```

Listing D.9: Frustum filter settings.

```
// Sampling surface normal's
pcl::SamplingSurfaceNormal<pcl::PointNormal> sample_surface_normal;
sample_surface_normal.setInputCloud(input_cloud_with_normals);
sample_surface_normal.setSample(50); // TODO(simon) Setting that is ...
    required to be a parameter.
sample_surface_normal.setRatio(0.5); // TODO(simon) Setting that is ...
    required to be a parameter.
sample_surface_normal.filter(*output_cloud_with_normals);
```

Listing D.10: Sampling of surface normal's settings.

```
// RANSAC
pcl::SACSegmentationFromNormals<pcl::PointNormal,pcl::PointNormal> ...
    segmentation;
pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients);
pcl::PointIndices::Ptr inliers (new pcl::PointIndices);

segmentation.setModelType(pcl::SACMODEL_NORMAL_PLANE);
segmentation.setMethodType(pcl::SAC_RANSAC);
segmentation.setOptimizeCoefficients(true);
segmentation.setMaxIterations(500);
segmentation.setDistanceThreshold(0.1);

segmentation.setAxis(Eigen::Vector3f(0,1,0)); // Y axis
segmentation.setEpsAngle(0.1);
segmentation.setInputCloud(output_cloud_with_normals_);
segmentation.setInputNormals(output_cloud_with_normals_);

segmentation.segment (*inliers, *coefficients);
```

Listing D.11: RANSAC of first plane settings.

```
pcl::PointCloud<pcl::PointNormal>::Ptr extracted_cloud_with_normals(new ...
    pcl::PointCloud<pcl::PointNormal>);
//Extract all points
pcl::ExtractIndices<pcl::PointNormal> extract_filter;
extract_filter.setInputCloud(output_cloud_with_normals_);
extract_filter.setNegative(true);
extract_filter.setIndices(inliers);
extract_filter.filter(*extracted_cloud_with_normals);
```

Listing D.12: Extract remaining indicies settings.

```
  second_segmentation.setAxis(Eigen::Vector3f(0,0,1)); // Z axis
  second_segmentation.setEpsAngle(0.1);
```

Listing D.13: Second RANSAC settings changes.

```
  plane_orgin.x() = ...
     -ransac_model_coefficients_.at(0)*ransac_model_coefficients_.at(3);
  plane_orgin.y() = ...
     -ransac_model_coefficients_.at(1)*ransac_model_coefficients_.at(3);
  plane_orgin.z() = ...
     -ransac_model_coefficients_.at(2)*ransac_model_coefficients_.at(3);
```

Listing D.14: Plane origin in space.

```
distance_scalar = (plane_orgin.dot(plane_normal_vector))/
                   (center_frustum_vector.dot(plane_normal_vector));

plane_vector_intersect = center_frustum_vector * distance_scalar;
```

Listing D.15: Calculating the distance scalar and intersection point.

```
cv::Ptr<cv::aruco::Dictionary> dictionary_ = ...
   getPredefinedDictionary(cv::aruco::DICT_APRILTAG_25h9);
cv::Ptr<cv::aruco::DetectorParameters> parameters_ = ...
   cv::aruco::DetectorParameters::create();
std::vector<std::vector<cv::Point2f>> markerCorners_, rejectedCandidates_;
std::vector<int> markerIds_;

std::vector<cv::Vec<double,3>> rvecs_;
std::vector<cv::Vec<double,3>> tvecs_;

float example_camera_matrix_data[9] = ...
   {907.114,0,662.66,0,907.605,367.428,0,0,1};
cv::Mat example_camera_matrix_ = ...
   cv::Mat(3,3,CV_32F,example_camera_matrix_data);

float example_dist_coefficients_data[5] = ...
   {0.157553,-0.501105,-0.00164696,0.000623876,0.466404};
cv::Mat example_dist_coefficients_ = ...
   cv::Mat(1,5,CV_32F,example_dist_coefficients_data);
```

Listing D.16: April tag marker configuration.

```
cv::aruco::detectMarkers(image_,
                         dictionary_,
                         markerCorners_,
                         markerIds_,
                         parameters_,
                         rejectedCandidates_);
```

Listing D.17: Detection of marker functuon.

```
cv::aruco::estimatePoseSingleMarkers(markerCorners_,
                                     0.535, // TODO(simon) configuration ...
                                        value should be set from a ...
                                        config-file.
                                     example_camera_matrix_,
                                     example_dist_coefficients_,
                                     rvecs,
                                     tvecs,
                                     object_points);
```

Listing D.18: Estimation of pose for the April tag.

D.3 Code Listings for Figure Generation

```
import matplotlib.pyplot as plt
import tikzplotlib
import scipy
import scipy.stats as stats
import seaborn as sns
import numpy as np

(mu, sigma) = stats.norm.fit(data)
skew = stats.skew(data)
kurtosis = stats.kurtosis(data,fisher=True) # Fisher kurtosis
save_case_plot_data(mu,sigma,i,test_number_label,False)
plt.figure(i)
ax = sns.distplot(data, #TODO(simon) Use displot or histplot as distplot ...
    is soon depreicated.
                  norm_hist=True,
                  kde=False,
                  bins=3,
                  fit=stats.norm,
                  color=data_colors_estimation[i],
                  label=f"{density_plot_labels[i]} ($\mu=${mu:.4f}, ...
                      $\sigma=${sigma:.4f},\n\t$\widetilde\mu_3=${skew:.2f}, ...
                      $\widetilde\mu_4=${kurtosis:.2f})",
                  axlabel=x_axis_labels[i])
```

Listing D.19: Generating denisity plot uisng Seaborn and SciPy.