Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Espen Kolberg Grødem
Magnus Christensen Myklegard

Pipeline for Monocular 6D Pose Estimation of Shipping Containers

Master's thesis in Engineering and ICT
Supervisor: Christian Holden
Co-supervisor: Sveinung Johan Ohrem and Bent Haugaløkken
June 2022

Espen Kolberg Grødem
Magnus Christensen Myklegard

Pipeline for Monocular 6D Pose Estimation of Shipping Containers

Master's thesis in Engineering and ICT
Supervisor: Christian Holden
Co-supervisor: Sveinung Johan Ohrem and Bent Haugaløkken
June 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

NTNU
Kunnskap for en bedre verden

Faculty of Engineering

TMM4935 - Industrial ICT

Master Thesis

Pipeline for Monocular 6D Pose Estimation of Shipping Containers

Author:
Espen Kolberg Grødem
Magnus Christensen Myklegard

Supervisor:
Christian Holden

Co-Supervisors from SINTEF Ocean:
Sveinung Johan Ohrem
Bent Haugaløkken

June, 2022

Preface

This master's thesis concludes our 5-year masters degree program at Norwegian University of Science and Technology (NTNU). The thesis was conducted at the Department of Mechanical and Industrial Engineering, and the thesis' authors are two students from the program Engineering and ICT.

We would like to express our sincere gratitude towards our supervisor; Associate Professor Christian Holden, for guidance and continued feedback during the whole semester. The thesis was carried out in collaboration with SINTEF Ocean, and we would like to give a special thanks to our co-supervisors Sveinung Johan Ohrem and Bent Haugaløkken for their support and suggestions.

Trondheim, Norway - June 11, 2022

Espen Kolberg Grødem
Magnus Christensen Myklegard

Abstract

In order to increase safety, and factors such as costs, time- and energy usage and emission, SINTEF Ocean has expressed interest in investigating the possibilities of retrofitting a hydraulic crane with a monocular camera system to increase the degree of autonomy in crane operations. The achieve this, real-time detection of the object pose of the payload is critical for the crane to be able to interact with its 3D environment. As the crane operates mostly on cargo that is of a cuboid shape, a software able to estimate the pose of a general cuboid shape is applicable, and therefore beneficial. Hence, a shipping container is chosen, as it is of cuboid shape as well as being an object heavily related to the maritime sector.

As the quality of a computer vision projects are heavily dependent on the data presented during training, combined with the challenges of annotating real life images, a synthetic dataset was created for this thesis. The dataset was constructed and rendered in the 3D modelling software Unity.

This thesis explores a pipeline for object pose estimation on shipping containers in an offshore environment. The pipeline consists of two segments; the detection segment and the pose estimation segment. The detection segment predicts the position of the shipping container using semantic segmentation and the U-Net network. The original image is then cropped around the prediction with a confidence score of 60% and above, and returned. The pose estimation segment takes in the cropped image and feeds it to the custom made DVFnet for regression of pixel-wise unit vectors pointing towards the keypoints. Random Sample Consensus voting process is then applied for keypoint localization. Finally, OpenCVs solvePnP() is used with the predicted keypoints, in order solve the PnP problem, and estimate the containers attitude and position in the 3D world.

Testing of the proposed pipeline was conducted on the custom created synthetic SINTEF 6DPE ISO container dataset. The results from the test shows that the pipeline is inadequate for use in real-time. The detection segment of the proposed pipeline achieved good results with the use of U-Net for segmentation. The trained U-Net reached a dice score of 82% on the test data and as such provided adequate predictions for the shipping container. DVFnet however, showed signs of overfitting that crippled its performances in its task to regress the unit vector fields for the keypoints. The network does not estimate a usable pose on using the predicted vector field, while the pose estimated on ground truth data struggles with systematical inaccuracies. Both the use of limited augmentations and the symmetrical properties of the container objects were identified as a possible source of error, as the network is not able to learn object features.

Sammendrag

For å øke sikkerheten, og faktorer som kostnader, tids- og energibruk og utslipp, har SINTEF Ocean uttrykt interesse for å undersøke mulighetene for å ettermontere en hydraulisk kran med ett mono kamerasystem for å øke graden av autonomi i kranoperasjoner. For å oppnå en sanntidsdeteksjonen av last er objektposisjon og orientasjon avgjørende for at kranen skal kunne samhandle med sitt 3D-miljø. Siden kranen for det meste opererer på last som har en kubisk form, er ett system som er i stand til å estimere posisjon og orientasjon til en generell kubisk form anvendelig, og derfor fordelaktig. Derfor velges en container, da den har en kubisk form i tillegg til å være et objekt som er sterkt relatert til den maritime sektoren.

Siden kvaliteten på et datasynsprosjekt er sterkt avhengig av dataene som presenteres under trening, kombinert med utfordringene med å annotere bilder, ble det laget et syntetisk datasett for denne oppgaven. Datasettet ble konstruert og gjengitt i 3D-modelleringsprogramvaren Unity.

Denne oppgaven utforsker ett system for estimering av objektposisjon og orientasjon på containere i et offshoremiljø. Rørledningen består av to segmenter; deteksjonssegmentet og segmentet som estimere posisjon og orientasjon. Deteksjonssegmentet forutsier posisjonen til containeren ved hjelp av semantisk segmentering og U-Net-nettverket. Deretter beskjæres og returneres det originale bildet rundt prediksjonen, dersom en konfidensverdi er over 60% . Systemets andre segmentet tar inn det beskårne bildet og mater det til det spesiallagde DVFnet for regresjon av enhetsvektorer som peker mot nøkkelpunktene. Random Sample Consensus, en konsensus-stemmeprosess blir deretter brukt for lokalisering av nøkkelpunkter. Til slutt brukes OpenCVs solvePnP() med de predikerte nøkkelpunktene, for å løse PnP-problemet, og estimere beholderens posisjon og orientasjon i 3D-verdenen.

Testing av den foreslåtte systemet ble utført på det spesiallagde syntetiske SINTEF 6DPE ISO-conatiner datasettet. Resultatene fra testen viser at systemet er utilstrekkelig for bruk i sanntid. Deteksjonssegmentet til dette foreslåtte systmet oppnådde gode resultater ved bruk av U-Net for segmentering. Det trente U-Net nådde en treffsikkerhet på 82% på testdataene og ga som sådan tilstrekkelige prediksjoner for containere. DVFnet viste imidlertid tegn på overtilpasning som lammet ytelsen i oppgaven med å regressere enhetsvektorfeltene for nøkkelpunktene. Nettverket estimerer ikke en brukbar positur ved bruk av det forutsagte vektorfeltet, mens posisjonen estimert på ground truth data sliter med systematiske unøyaktigheter. Både bruken av begrensede visuelle endringer på treningsdataen og de symmetriske egenskapene til containerobjektene ble identifisert som mulige feilkilder, da nettverket ikke er i stand til å lære karakteristikker for en container.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The offshore environment is a hazardous scene which greatly raises the risk of damage on material or personnel. Vessel collisions, dropped object hazards, dangerous payload sway trajectories or extreme environmental conditions are some of the hazards listed by (Ozguc 2021), which assesses the damages caused by accidental dropped cargo from offshore cranes. These conditions can lead to economical losses or even worse, the loss of lives. As described in Fenstad (2021), accidents occur during loading and unloading, and especially workers on fishing vessels and cargo ships are at risk of injuries related to these types of operations. The article further describes how introducing an autonomous crane system could decrease the number of accidents. The same crane system is referred to in Nordal (2021), stating that there is a need to further streamline the process of loading and unloading ships in a safe way, with as few personnel as possible. Making this process more efficient and autonomous will not only increase safety. As the loading and unloading process requires large amounts of energy to operate the crane and other relevant systems, such as maintaining a stable position of the vessel, further streamlining the process will decrease time- and energy usage. Hence reducing costs and potentially reducing emissions (Nordal 2021).

The responsibility of crane operators in these conditions are extensive (Fang et al. 2020), and the need for computer assisted aid or complete autonomy could be vital to secure both the safety of personnel and material. In fact, MUNIN (2016), a rationale of autonomous ships, proposes that autonomy will help increase safety along with efficiency in the maritime sector, which is sorely needed as crane operations are regarded as bottlenecks in the maritime supply chain (Ngo and Hong 2012). The rationale also proposes that autonomy even helps lessen the environmental footprint and the damage on local marine habitats because of fewer accidents on board. To reach autonomy in such crane systems however, the crane needs to know the pose of the object of interest, meaning the position and orientation in the world.

The ability to identify and compute object poses in images has been an interesting research problem for decades. However, the benefits of going from the 2D image domain to identification in the 3D domain have become more evident and applicable during the last decade. This is due to the advancement in computer technology and the monumental increase in both computational power and availability, which enables computers to handle 3D data and sensors more efficiently (Sølund 2017). For a mechanical system, i.e a crane system, to interact with and manipulate an object in 3D space it needs methods in place to detect the object and estimate the objects 3D position and attitude. There exists numerous technologies for informing the system of an objects presence, such as LiDAR, *Light Detection and Ranging* (Opromolla et al. 2015), SLAM, *simultaneous localization and mapping* (Eade and Drummond 2006), sensor fusion, combining wheel encoders, gyroscope, accelerometer and electronic compasses (Nasir and Roth 2012), or IMUs, *Inertial Measurement Unit* (Hol et al. 2009). Most of these technologies use senor fusion to detect and calculated the pose of an object, but later research have moved on to utilizing the benefit of deep learning and computer vision methods. Computer Vision (CV) has been a trending topic since it was first introduced in the 1970s as a branch of artificial intelligence (Gooodfellow et al. 2016), but attempts on object detection and pose estimation using CV were sparse before the revolution of deep learning started back in 2012 (Buntine 2020). The main goal of CV, according to Gooodfellow et al. (2016) is to enable computers, devices, robots or machines to interpret their surroundings in the real world through vision, much like humans, given by a camera(s). To do this, CV methods breaks down the world portrayed through a 2D image plane into different attributes. These attributes could be anything from shapes, colors, contrasts, corners, lines or depth, to name a few. CV's versatility and cost efficiency has led to its usage in a great number of applications and industries, as shown on the graph of market revenues in figure 1.1 produced by Statista below.

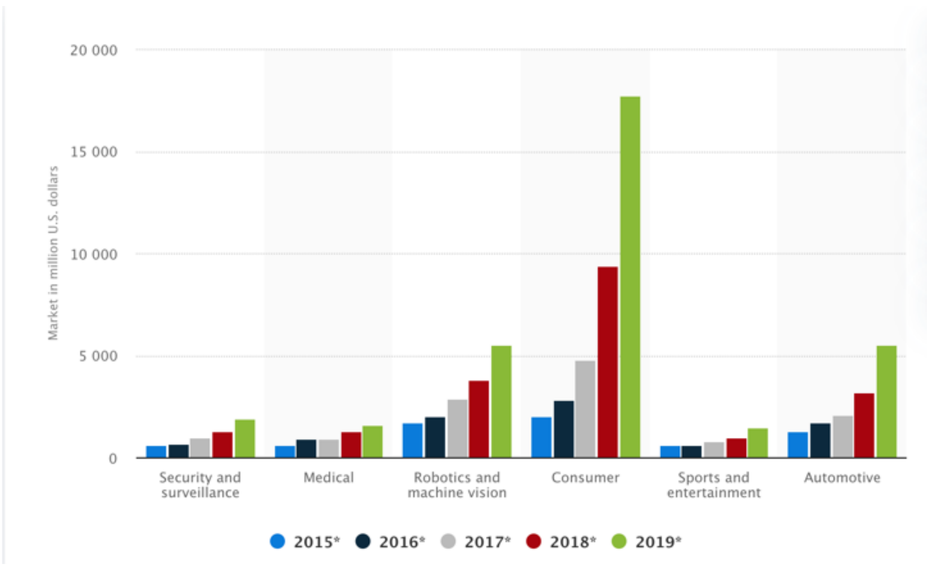
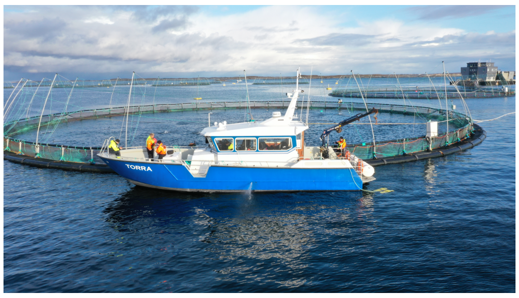


Figure 1.1: The rise in CV systems in different industries, here shown by the increase of market revenue for CV applications. (Irina Peregud 2020)

For robots or autonomous cranes, being able to the detect objects in the visual inputs is not enough to interact with the world however. The attributes garnered from a 2D image plane are constrained to the 2D work space and needs to be translated to the 3D work space. To accomplish this a CV system can make use of object pose estimation to find the position and orientation of objects, along with their texture and color information. Given the orientation and position in the world coordinate system of both the autonomous crane and the objects in its environment, the crane can interact with and manipulate its surroundings (Code 2017).

This thesis has been completed in collaboration with SINTEF Ocean, one of Europe's largest independent researching facilities, that focuses on innovation and research in the maritime sector. To conduct tests and research SINTEF Ocean employ their research vessel M/S Torra. The vessel is used for the development and testing of new ocean technology and sports a variety of instruments and utilities (Bremvåg 2015). One such instrument is the hydraulic crane attached on the stern of the vessel, used for loading and unloading payloads from land and water. Figure 1.2 below shows the M/S Torra and her hydraulic crane next to a aquaculture facility.


(a) Vessel with the crane on the stern

(b) Vessel shown from above

Figure 1.2: The vessel and the crane outside of a aquaculture facility.
Courtesy of Magnus Pedersen, SINTEF Ocean.

1.1 Problem Description

Crane operations are important in aquaculture industry and used in almost all aspects of operations that involve service vessels. However, cranes used in aquaculture are often hydraulic cranes with very low degree of automation, few sensors and automatic systems involved in the operational control. Ships and crane systems are also often completely disconnected from each other (SINTEF 2021). Autonomy in crane operations could contribute in reducing personnel risk and injuries, in addition to reduced time- and energy usage, leading to reduced costs and potentially lower emissions. The potential to increase automation does not only apply to cases within aquaculture, and could contribute to the increase in safety and efficiency in the broader maritime sector as well.

In order to achieve a higher degree of automation, SINTEF Ocean is interested in investigating the potential for retrofitting the hydraulic crane in Figure 1.2, with sensors and systems to automate existing crane operations. As a part of this thesis, object detection and object pose estimation are described as necessities for reaching autonomy. This statement is based on the fact that object detection and object pose estimation are essential for the crane to obtain the 3D information of an object of interest. Achieving an increase in autonomy, by installing sensors and systems to automate crane operations, could be a significantly lower investment compared to the procurement of a new crane, and could lower the bar for future projects to invest in a higher degree of autonomy.

The crane on M/S Torra is today used in crane operations lifting various objects, of various shapes and sizes. However, many objects handled by the crane are of a cuboid shape, such as boxes, crates and an underwater Remote Operated Vehicle (ROV) (Sunde 2018). Consequently, SINTEF Ocean expressed that a monocular pose estimation pipeline using a model able to detect an object with a general cuboid shape is of interest. Also, since automation using monocular object detection and object pose estimation can be utilized in cases other than retrofitting non-autonomous cranes, the development of a CV model able to detect an object with a general shape is relevant. A shipping container was therefore selected as the object which would be detected and which pose would be estimated, whereby in addition to being a general cuboid shape, is an object heavily related to the maritime sector.

In order to improve safety, the pose estimation needs to be computed at a rate high enough for security systems or human personnel to intervene in case of potential hazards. Therefore, the system must compute pose estimation in real-time. In Zhou et al. (2021), image based onsite object tracking for automatic crane lifting tasks are conducted, concluding that a total image processing time of 0.6 seconds easily fulfils real-time requirement. In Price et al. (2021) a system is implemented for tracking of cargo lifting operations, and the project concludes that the safety criteria is met at processing time of 1 second. Unlike the construction cranes presented in these projects, the crane on M/S Torra will be affected by inertia and displacement provided by the motion of waves. As such, a larger margin of error is assumed in this thesis, and consequently in order to be classified as real-time, a processing speed of 0.25 seconds for each image is required.

1.2 Objective

This thesis aims to build on the problem previously described, hence this thesis will focus on using CV on object pose estimation of shipping containers. To reach object pose estimation, a strong object detection foundation is needed to secure the calculation of reliable poses. Therefore, this thesis will also focus on the vital object detection part of the process, as this is the cornerstone of all the succeeding methods. The research will be conducted using a monocular system, meaning data will be provided from a single camera for object detection and object pose estimation. Thus, the objective is defined as the following:

Research and implement a pipeline of CV models that utilize object detection to detect and classify shipping containers in order to estimate the pose of a shipping container in real-time.

In order to meet the objective and appropriately support the case provided by SINTEF Ocean, the following capabilities are regarded as vital characteristics for the proposed CV models, where real-time is defined as a processing speed of 0.25 seconds for one image. The capabilities were derived in collaboration with supervisor, and an assessment based on the importance of accuracy and precision introduced in the problem description.

- The model needs to be able to process data in real-time.
- The model needs to be able to detect shipping containers in images with an accuracy of 80% or more.
- The model needs to be able to estimate container coordinates in 3D space with an error of less than 10% of the distance between camera and container.

The source code for this thesis can be found on: Github, a code hosting platform for version control and collaboration.

1.3 Related Work

Materials presented in this section, as in general throughout this thesis, are mainly obtained using the databases ScienceDirect, Scopus and Google Scholar. Random search was used, with keywords within each topic to discover a basis of relevant materials, after which a strategy of following references and citations as describe in Snyder (2019) were implemented. While discovering new papers and materials, the criteria listed below were used as a basis to evaluate the trustworthiness of the material.

- *Where* is the paper published
- *Who* has written the material
- *When* was it published
- If available, the number of citations for the material

If an unknown source had published a paper, a search of the materials title were conducted on other relevant web pages, and if not found an assessment was conducted whether or not the paper and source could be trusted, based on the criteria listed above and an external search to obtain information of the source.

1.3.1 Monocular Object Pose Estimation

Monocular object pose estimation could be divided into two subtasks, instance-level object pose estimation and category-level pose estimation (Fan et al. 2021). Instance-level 6D object pose estimation tasks require the same statistical distribution both for source data, from which a classifier is learned, and for target data on which the classifiers will be tested. Hence, instance-based methods estimate 6D poses of familiar objects, mainly targeted to overcome challenges such as viewpoint variability, occlusion, clutter, and similar-looking distractors. (Caner Sahin 2019). The two categories are not uncorrelated, and category-level object pose detection is a more general extension of instance-level object pose estimation (Fan et al. 2021) which inherently involve the challenges such as distribution shift among source and target domains, high intra-class variations, and shape discrepancies between objects (Caner Sahin 2019). In the category-level monocular object pose detection tasks, a well-trained model should obtain the ability to recognize a line of object instances that belong to the same category (Fan et al. 2021). An example of category-level monocular object pose estimation is in X. Li et al. (2019). This project estimates poses of objects, such as sunglasses, consisting of a fixed number of adjustable rigid parts, using a similar approach as H. Wang et al. (2019). The assumption is made that no CAD model is available, and the object coordinates are then regressed within a Normalized Object Coordinate Space (NOCS) where objects are within a category are consistently oriented within a common normalized space. This method however, even though pose estimate is achieved, struggles with estimating pose using only RGB data (Fan et al. 2021, p. 22). Similar methods focusing on general solutions, such as in X. Chen et al. (2020), experience a significant drop in performance when depth data is not available.

Several instance based deep learning methods exist, where deep discriminative feature representations are learned (Caner Sahin 2019). According to different input data formats, instance-level monocular object pose detection methods are classified into RGB-based methods and (RGB)D-based methods (Fan et al. 2021, p. 9), where the more recent paradigm is to adopt end-to-end trainable CNN architectures, and solve the pose prediction problem in RGB-only (Sahin et al. 2020). Fan et al. (2021) divides the RBG-based deep learning methods into five major classes, direct methods, keypoint-based methods, dense coordinate-based methods, refinement-based methods, and self-supervised methods. In the following subsection the same classification will be used to introduce relevant background information of the other classifications in RGB-based deep learning pose estimation.

1.3.2 RBG-based deep learning methods

Refinement-based methods are methods requiring an additional step to improve their performance, and to some degree is a combination of other methods (Fan et al. 2021). Self-supervised methods, as defined in Fan et al. (2021), are methods that completely or partly generate data used for training itself or another model. As with refinement-based methods, self-supervised could also be a combination of the methods described below.

Direct methods

Direct methods estimate position and orientation directly by treating object pose estimation as a regression or classification task (Fan et al. 2021). PoseCNN Xiang et al. (2017) estimates 6D pose using a single CNN, where the network is trained to perform three separate tasks: semantic labeling, estimating 3D translation and estimating 3D rotation. The object center is estimated by predicting a unit vector from each pixel towards the object center. Robust to occlusions as pixels vote the object center even if it is occluded by other objects.

Dense coordinate-based methods

Dense coordinate-based methods are correspondence based methods, creating 2D to 3D correspondences by predicting the 3D object coordinate of each pixel (Fan et al. 2021). In Zhuo et al. (2018) 3D box proposals are generated by regressing 3D coordinates from 2D anchors. More specifically, a network predicts a depth map which is used in combination with a RGB image in generating 3D box proposals, which then are used to estimate the pose. However, the model struggles with prediction of depth when similar objects appear in the foreground and background. Park et al. (2019) constructed a network which generates normalized 3D coordinates of each pixel in the object coordinate space, which has similarities to H. Wang et al. (2019), but without the use of depth data. In summary, dense coordinate-based methods are robust to heavy occlusions and symmetries. However, regressing object coordinates remains more difficult than predicting sparse keypoints due to the larger continuously searching space. (Fan et al. 2021, p. 12).

Keypoint-based methods

Keypoint based methods is another example of 2D to 3D correspondence based methods, similar to dense coordinate based methods. Compared to directly predicting pose, keypoint based methods are more accurate (Fan et al. 2021). A widely used approach to the 6D pose estimation problem, the methods detect 2D keypoints in the image and then solves the perspective-n-point (PnP) problem (Fan et al. 2021), which is the problem of estimating translation and rotation of objects with respect to the camera using known camera parameters, known 2D- and 3D points (Fischler and Bolles 1981), further described in detail in Section 2.5.

Within keypoint-based methods there are various approaches to generating the 2D keypoints. Tekin et al. (2017) and Xu et al. (2020) infers 2D points directly from the network, training the network to predict the 2D correspondences to the 3D bounding box of the object. While achieving good results and real-time pose estimation, the networks struggles with severe occlusion. Oberweger et al. (2018) predict Gaussian heatmaps from multiple patches and accumulates the results, from which we compute the 3D object pose by solving the PnP problem.

Similar to Xiang et al. (2017), Peng et al. (2018) and Yu et al. (2020) approaches the keypoint estimation by generating unit vectors indicating keypoint directions from each pixel. The task of predicting pixel-wise directions enforces the network to focus more on local features of objects and alleviates the influence of cluttered background. Another advantage of this approach is the ability to represent keypoints that are occluded or outside the image. Even if a keypoint is invisible, it can be correctly located according to the directions estimated from other visible parts of the object (Peng et al. 2018). To increase the robustness to occlusions, voting strategies are employed to localize feature keypoints or coordinates (Yu et al. 2020).

Proposed approach

Any method engineered for 6D object pose estimation has to cope with the challenges of the problem in order to robustly work in a generalized fashion (Caner Sahin 2019, p. 3). Challenges such as handling objects with various textures, viewpoint variability in the dataset and occlusion. Occlusion occurs when an object is partly or completely hidden in the image, and as stated in Caner Sahin (2019) it is one of the most common challenges in 6D pose estimation. There is also other aspects of choosing a model. Discussed in full detail in Huang et al. (2016) and briefly discussed in combination as the results are presented in Tekin et al. (2017), there is a co-variation between an increase in speed and accuracy achieved. As the framework of this project is to develop a system able to estimate object pose in real-time, computational speed is of great importance while also needing to achieve an accuracy which complies with the characteristics define in Section 1.2.

Therefore, unit vector field generation was prepared according to the procedure used Peng et al. (2018) and Yu et al. (2020), as further describe in Section 3.3. The criteria for selection was based on a combination of results achieved, and especially as unit vector generation has shown efficient at handling occluded objects (Xiang et al. 2017), (Peng et al. 2018), (Yu et al. 2020), and its ability to achieve good results running in real time (Peng et al. 2018), (Yu et al. 2020).

While not being decisive for selection of method in this project, an advantage of the method presented in Peng et al. (2018) is that it can handle multiple instances using a strategy presented in Xiang et al. (2017) and Papandreou et al. (2018). For each object class, we generate the hypotheses of the object centers and their voting scores using our proposed voting scheme. Then, we find the modes among the hypotheses and mark these modes as centers of different instances. Finally, the instance masks are obtained by assigning pixels to the nearest instance center they vote for.

The unit vectors will only be generated in certain areas of the image, namely the pixels which represents a shipping container. For the network to be able to recognise and learn texture features, it is essential that it is presented the areas at which there is located a container. A high quality pixel-wise instance segmentation, resulting in the actual silhouette of the object in the image, is therefore critical. The task of instance segmentation can be formalized into a separate classification problem where for each pixel it is decided what class the pixel belongs to (Nikolenko 2021, p. 77). For that reason, the pipeline will conduct two separate tasks; first a instance segmentation and then a vector field prediction.

1.4 Thesis Overview

This thesis begins with covering the necessary preliminaries for CV and pose estimation in chapter 2. In chapter 3, the methodology is presented with all the methods used. chapter 4 introduces the proposed pipeline for solving object pose estimation in detail. The networks used are also covered in greater detail. In chapter 5, the results from training and testing the pipeline are presented. chapter 6 discusses the results achieved from the pipeline. Finally, in chapter 7 a conclusion is presented and potential further work is covered.

- **Chapter 2, Preliminaries**

  This chapter presents the necessary preliminaries the reader needs knowledge of to better understand the theories laid forth in this thesis, along with the proposed pipeline and neural networks.

- **Chapter 3, Methodology**

  In this chapter the quantitative methodology used in this research is described, and how and why the methods used address the research objective. The chapter describes the data collection process and technologies used to enable pose estimation.

- **Chapter 4, Pipeline**

  This chapter brings forth the proposed pipeline for monocular 6D pose estimation of shipping containers in offshore environment. The pipeline structure is presented, with all of its components. Further details of the two neural networks selected are then covered in detail.

- **Chapter 5, Results**

  This chapter systematically lays forth the results obtained as a results the implemented models. First, the results from the segmentation model are presented. Following this the pose estimation results for estimated vector fields are presented, in addition to altered versions of the ground truth vector field.

- **Chapter 6, Discussion**

  This chapter investigates the relationship between the findings presented in the previous chapter, and theory uncovered during the research project. The chapter intends to address the research objective.

- **Chapter 7, Conclusion**

  A summary of findings is presented as well as limitations to this study and recommendation for further work.

Chapter 2

Preliminaries

This chapter presents the necessary preliminaries the reader needs knowledge of to better understand the theories laid forth in this thesis, along with the proposed pipeline and neural networks. It is assumed that the reader has some underlying knowledge of concepts in the field of deep learning and computer vision. Section 2.1 introduces key aspects of computer vision and presents a compact introduction to deep learning and convolutional neural networks. Then, in Section 2.3 relevant mathematics are presented, mainly covering Euler angles and rotation. In Section 2.4 the pinhole camera model is covered in detail. Furthermore, in Section 2.5 the perspective-n-point algorithm is explained. Finally, in Section 2.6 the RANSAC algorithm is introduced.

2.1 Computer Vision

Computer vision is a field within artificial intelligence, that enable computers to interpret visual information such as images and videos. This section introduces subjects within the field of computer vision and deep learning in general, which is a method that has become the backbone of modern computer vision (Nikolenko 2021).

2.1.1 Deep Learning

Deep Learning is a subcategory of the broader field of machine learning, and subsequently artificial intelligence. The cornerstone of deep learning is artificial neural networks (ANNs); methods and algorithms designed to imitate, though simplified, the processing of information by neurons in the cortex . This enables an ANN to learn by example like humans (Abiodun et al. 2018). Sharma et al. (2020) quotes one of the first inventors of neurocomputers on the definition of neural networks to be:

"A computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs."

The word deep refers to the amount of hidden layers that structure the ANN, as shown in Figure 2.1, here with a depth of L_3. A classic ANN can be divided into three prime components: the input component, the hidden layers component and the output component. These components are depicted in Figure 2.1 where green is the input, purple the hidden layers and red the output. The input layer receives the input data. The data constitutes the intended functionality of the network, i.e image classification or natural language processing. Next, the hidden layers preform computations on the given input(s) to the network. Each hidden layer contains a predetermined number of neurons (purple). The output layer returns the output data where the format is co-dependant on the data type of the input.

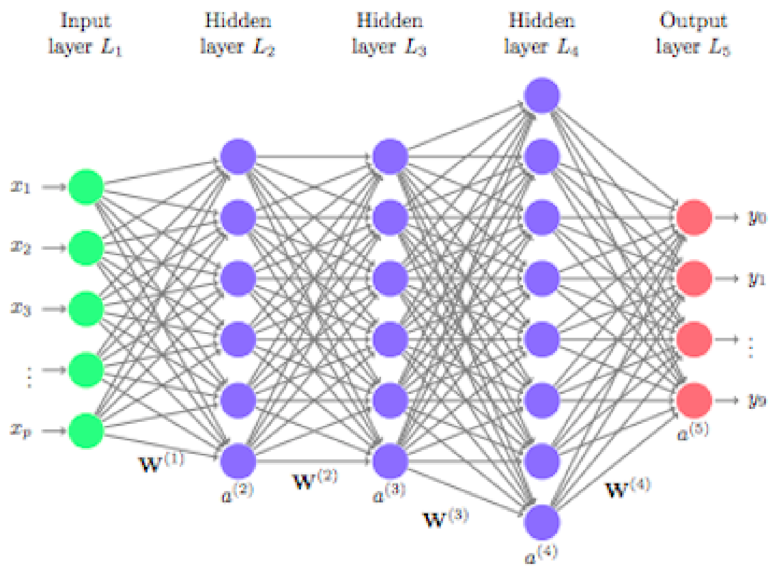


Figure 2.1: A illustration of a simple feed forward ANN. The weights and connectedness are represented by the arrows between each neuron (UCB 2018)

The network learns by decomposing the input down into features that it memorizes and uses to recognise these features in new instances of inputs. It does this by calculating weights and biases (Hinton 1992). The weights are the numeric association between neurons, depicted by the arrows in Figure 2.1. It effectively dictates the importance of the input values and assists in deciding what features are memorized.

Connectedness

The composition of neural networks allows for a diversity of connections between neurons in the network. These configurations in connections are often classified as either sparse connectivity or dense connectivity (Gooodfellow et al. 2016). Sparse connectivity is when a neuron is connected to only a portion of the neurons in the succeeding layer. Dense connectivity is when a neuron is connected to the entire succeeding layer of neurons. This can be seen in Figure 2.2 where the activated neuron is x_3 and its connections in layer s are highlighted in grey. Densely connected layers offer great accuracy but at a cost of computational complexity (Gooodfellow et al. 2016).

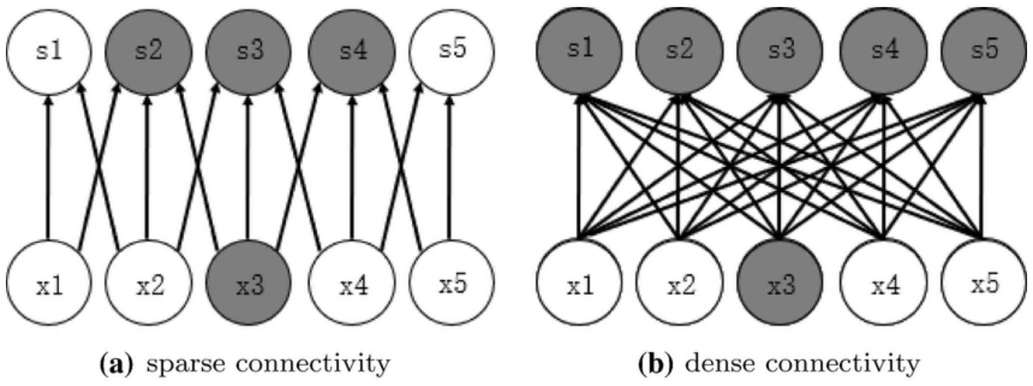


Figure 2.2: The network in a) is an example of sparse connectivity as the activated neuron x_3 is only connected to the three neurons s_2 ,s_3 and s_4. The network in b) displays a dense connectivity where x_3 is connected to all succeeding neurons. (J. Li and D. Liu 2021)

However, an ANN by it self is not capable of extracting feature information from an image as they struggle with the computational complexity it requires (O'Shea and Nash 2015). To utilize the exceptional computing power of ANNs on images the network needs to employ the mathematical operation of convolution. Integrating this operation into the network builds the foundational network of modern day computer vision, the convolutional neural network.

2.1.2 Convolutional Neural Networks

Convolutional neural networks are analogous to ANNs as they both are composed of nodes that self-optimise through learning (O'Shea and Nash 2015). However, Convolutional neural networks are greatly used in image processing problems where the operation of convolution is used to determine abstract features in a arbitrary image input (O'Shea and Nash 2015). This is done as the image propagates towards the deeper layers of the network. Intuitively, a network detects edges in the initial layers before detecting more complex patterns and high level features in the deeper layers (Albawi et al. 2017). Figure 2.3 visualizes this propagation during facial detection.

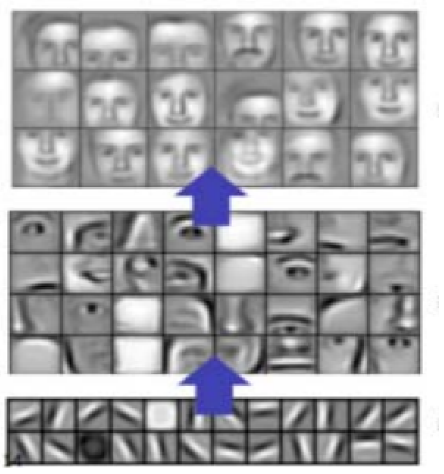


Figure 2.3: Illustration of learned features in a convolutional neural network (Albawi et al. 2017)

Architecture and Function

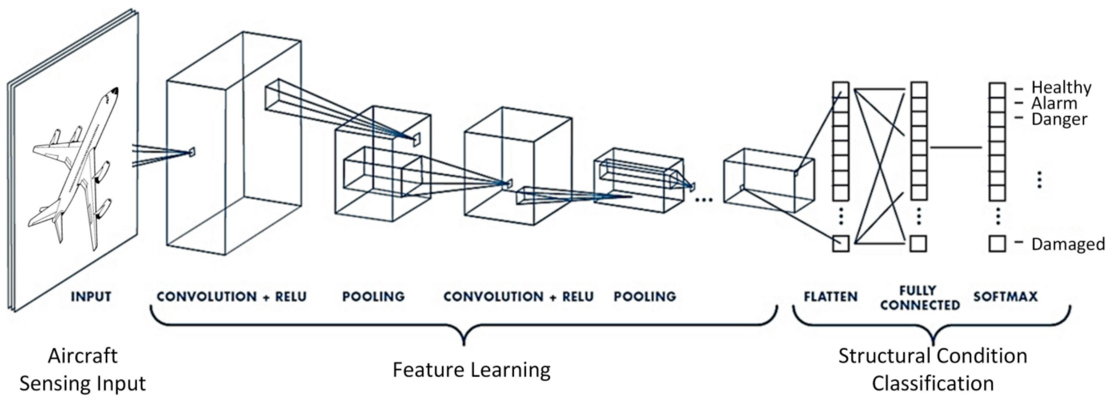


Figure 2.4: A classic CNN structure (Elhamraoui 2020)

Convolutional Neural Networks are composed of different layers to form two components; feature learning and classification, as shown in Figure 2.4. The feature learning component, along with the flatten and fully connected layer, contains the hidden layers of the network, while the softmax layer is the output layer. The inputs fed to the network in the input layer are images represented as matrices. These matrices share the dimensionality of the images and their properties. This means that an input consisting of a gray scale image with the dimensions of 64x64 is represented as a matrix of the same proportions. Consequently, each pixel of the image is represented by an integer in the matrix. This integer correlates to the color value intensity of the pixel. For a grey-scale image this value ranges from 0 to 255 where 0 is black and 255 is white. The network then performs the operations of convolution and pooling on the input before it is flattened and fed to to the fully connected layer and the output layer for classification, as shown in Figure 2.4.

Table 2.1 shows each hidden layer of the network and their contained functions.

Aircraft detecting CNN	
Layer	Functions
Input layer	Images
1. Hidden layer	Convolution + Relu Pooling
2. Hidden Layer	Convolution + Relu Pooling
3. Hidden Layer	Flatten Fully Connected
Output Layer	Softmax

Table 2.1: The architecture of the CNN in Figure 2.4

The convolution layers secures the vital information for classifying the input and discards the rest, thus reduces the amount of computation needed to classify the input. This is done by the use of learnable kernels (O'Shea and Nash 2015). A kernel, otherwise known as a filter, is a matrix of isomorphic dimensions, usually of small dimensionality and covers the entirety of the depth of the input (O'Shea and Nash 2015). The kernel is moved across the input with a given stride, generally two strides, until it parses the complete width of the input. It then jumps back to the beginning and shifts down in accordance with the stride, starting the process over. It repeats this path until the entire input is traversed.

Convolution

The mathematical operation of convolution on images is executed by the use of a kernel. The standard kernel size for convolutional operations are 5x5 and 3x3, but may vary on the basis of the effect the applier wants (Mittal 2019). The kernel extracts features using weighted functions represented by the equation below:

$$(i, j) = (K * I) = \sum_m \sum_n I(i - m, j - n)K(m, n) \tag{2.1}$$

where K is the kernel, or filter, I is the input (image or feature map) and $*$ is the convolution operation. Figure 2.5 illustrates the process of how the convolution kernel is applied to an input.

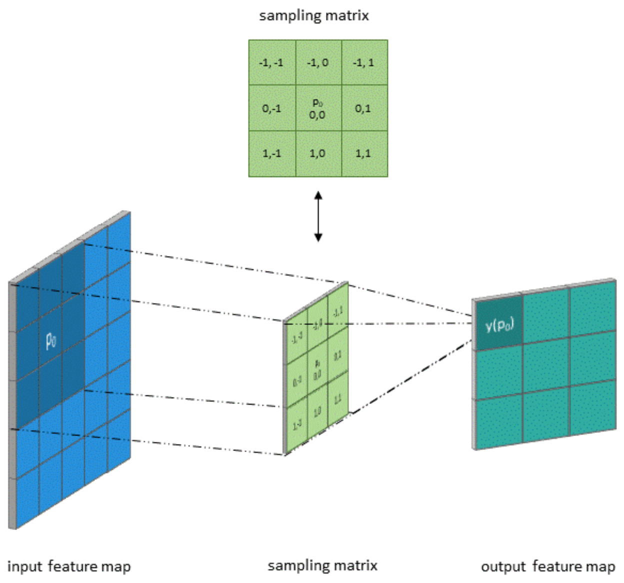


Figure 2.5: The visualisation of convolution (Raza 2019)

Pooling and downsampling

The pooling operation is a non-linear transformation that summarizes the values of a specific region of the input as a single value using a kernel. This summarized single value is derived from the statistics of the neighboring values covered by the kernel (Guissous 2019). The pooling operation reduces the dimensionality of the input, consequently reducing the memory usage. In short, its function is to reduce the spatial size of the convoluted feature map given from the convolution layer to lessen the number of parameters. The two most common pooling techniques are max and average pooling. Figure 2.6 shows their respective processes.

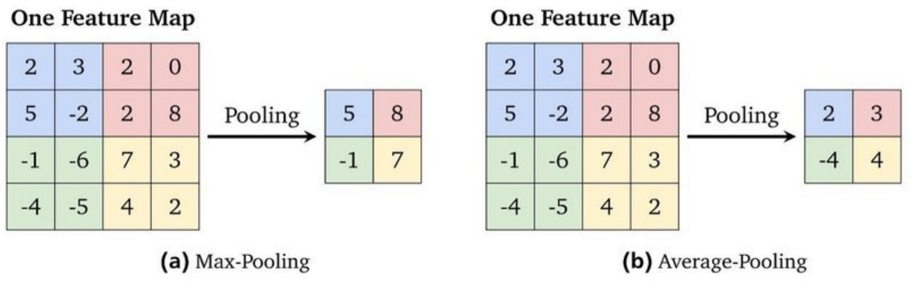


Figure 2.6: Pooling operation visualised (Guissous 2019)

An important aspect of pooling is that it contributes in making information in the input become approximately invariant to small translations (Nagi et al. 2011). Invariance to translation suggests that if one translates the input by said small amount, the values of the pooled outputs in the quadratic pooling region do not change.

Of the two named techniques, the max configuration is the most used as it often leads to faster convergence rate as it systematically selects the highest invariant features (Nagi et al. 2011). Similarly to convolution, it uses a kernel with a given stride over the input image. It collects the pixels with the highest intensity value and stores the output in a new feature map. The pooling operation is used after a convolution layer paired with an activation function.

Activation Functions

Activation functions, like ReLU, *Rectified Linear Unit*, used in the CNN described above, are what gives neural networks their ability to recognize complex mappings from the data. Without them a network would work like a sub optimal Linear Regression Model that has limited performance capabilities (Sharma et al. 2020). The purpose of the activation function is to decide whether a node should be activated or not by calculating the weighted sum, which is the sum of all preceding weights ($w_1, w_2, ..., w_n$), including any corresponding biases. A simplified illustration is shown in Figure 2.7.

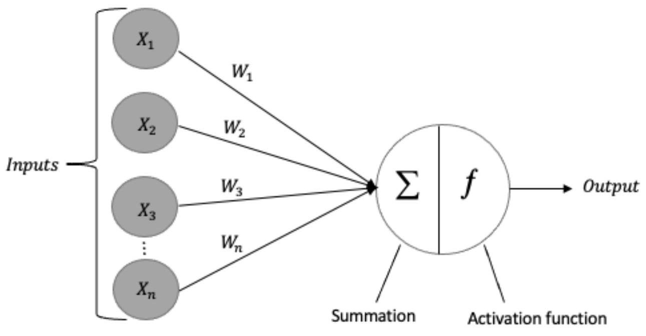


Figure 2.7: Illustration of a node (neuron) in a network

These calculations introduces non-linearity into the output of a node. If the result after the calculations are true, the node will pass on the weights to the succeeding layers. As mentioned there are several activation functions, but the most common is Rectified Linear Unit, or ReLU for short (Dishashree 2020).

ReLU activates or deactivates neurons based on the value of its output, and is defined as:

$$f(y) = \begin{cases} 0 & \mathit{if}\, y < 0 \\ y & \mathit{if}\, y \geq 0 \end{cases} \tag{2.2}$$

Here y is the output of the node, where the weights and biases have been summed. The output of ReLU is either 0 or y, depending on the sign of y. If $y \geq 0$ ReLU will return y, but if $y < 0$ it will return 0. If y is returned the corresponding node will be activated while 0 on the other hand will deactivate it.

Another popular activation function is LeakyReLU, a variant of ReLU which assigns none zero outputs for negative input: $f(y) = max(\alpha y, y)$ where α is a selected value in the range [0,1]. A common α is 0.001. The equation for LeakyReLU is given as:

$$f(y) = \begin{cases} 0.001y & \mathit{if}\, y < 0 \\ y & \mathit{if}\, y \geq 0 \end{cases} \tag{2.3}$$

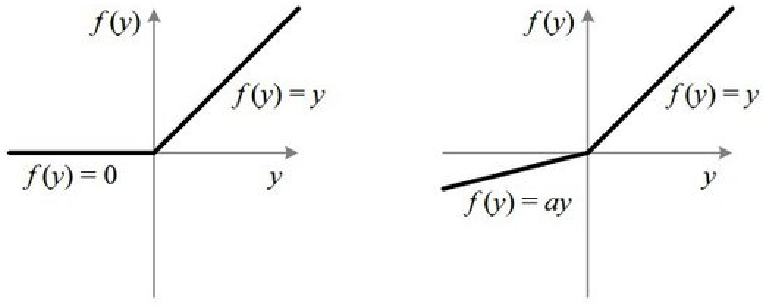


Figure 2.8: Plotting ReLU and LeakyReLU, source: (Z. Li et al. 2022)

LeakyReLU is popular due to that it avoids the dying ReLU problem, which happens when ReLU always outputs the same value for any input. This happens when a nodes learns large negative bias term for its weights. Effectively this means that the nodes will always return 0 and will never alter its weights.

Depending on the activation state of the nodes, the next step in the processes is the pooling operation.

Dropout

Dropout is a regularization technique that randomly drops a portion of the nodes that feed into the fully connected layer, temporarily removing them. This means that the nodes dropped have no affect on the corresponding activation function (i.e their contribution value is set to 0). Consequently, the gradients linked to these nodes are set to 0 as well and their weights are not updated (Srivastava et al. 2014).

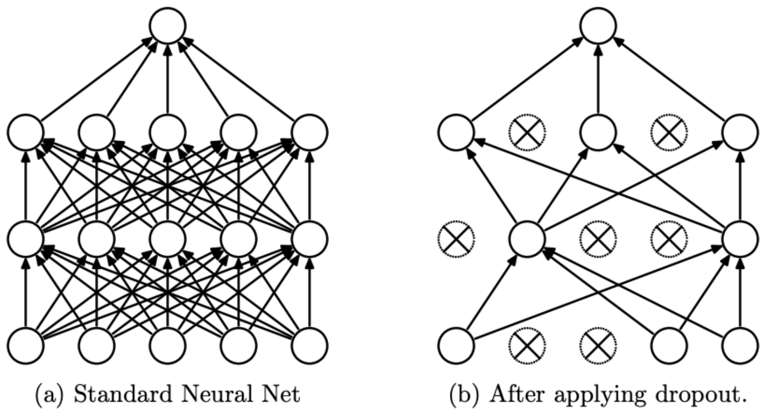


Figure 2.9: Network (a) is standard network with two hidden layers. Network (b) is a "thinned" network after applying dropout, here the nodes depicted with crosses have been dropped (Srivastava et al. 2014)

.

The use of dropout can lead to significantly lower generalization error, (Srivastava et al. 2014). This means that Dropout prevents the network from memorizing the training data and heightens prediction on new unseen data.

Batch Normalization

Batch Normalization (BN) has quickly established a indispensable position in modern neural networks. To put it generally, BN is a stabilization method applied on the distribution of inputs (of a single batch) to a given network layer during training. This stabilization of the distributions is achieved by augmenting the network with layers that subtract the mean μ and divide by the standard deviation σ. The normalized inputs are then scaled and shifted on the basis of the trainable parameters γ and β, (G. Zhang et al. 2018). The BN algorithm is as follows:

Algorithm 1 Batch Normalization - (G. Zhang et al. 2018)

```
Input: Values of $x$ in a batch: $\mathbf{B} = (x_1, ..., x_n)$; parameters: $\gamma$ and $\beta$
Output: $y_i = BN_{\gamma\ ,\beta}(x_i)$
  $\mu_{\mathbf{B}} \leftarrow \frac{1}{n}\sum_{i=1}^{n} x_i$    ▷ Calculating the mean value of the Batch
  $\sigma^2_{\mathbf{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{n}(x_i - \mu_{\mathbf{B}})^2$    ▷ Calculating the variance of the Batch
  $\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathbf{B}}}{\sqrt{\sigma^2_{\mathbf{B}} + \epsilon}}$    ▷ Normalizing the Batch
  $y_i \leftarrow \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$    ▷ Scaling and shifting
```

In summation we write the equation for BN as:

$$BN(x) = \frac{(x-\mu)\gamma}{\sigma} + \beta \tag{2.4}$$

Further, it is important to note that BN is applied before the activation function and not used in the output layer.

Softmax

The softmax function, also known as *multinomial logistic regression*, is a logistic regression which normalizes an input value into a vector of values that follows a probability function. The output values are in the range [0,1] and thus allows for multi class classification. The equation for softmax regression is:

$$Softmax \rightarrow f(x_i) = \frac{e^{x_i}}{\sum\limits_{j=1}^{k} e^{x_j}} (i = 1, 2, ..., k) \tag{2.5}$$

where $x_1, x_2, ..., x_k$ are the input values of the softmax layer and the output of $f(x_i)$ casts the probabilities that the input belongs to the i^{th} label. As mentioned above the softmax function allows for multi class classification, but for binary classification it approximates into the sigmoid function.

Sigmoid

The sigmoid function is one of the most common activation functions used in neural networks because of its non-linearity and computational simplicity of its derivative. The function has a range from [0,1], but it has an inherent odd symmetric property with respect to the point (0, 0.5). This means that the function obeys the expression $f(-x) = 1 - f(x)$, as shown below.

$$Sigmoid \rightarrow S(x) = \frac{1}{(1 + e^{-x})} = \frac{e^x}{e^x + 1} = 1 - S(-x) \tag{2.6}$$

where x is the input value of the sigmoid layer and the output of $S(x)$ casts the probability that the input is either a class or not.

2.1.3 Loss Function

Loss functions are in essence the objective function of a neural network. It is the function we want to minimize, according to the objective of the network. It calculates how far the model deviates from the target data. This means that the loss function is used to adjust the model's weights during training. For reference, a perfect model has a loss of 0.

2.1.4 Optimization Algorithm

Optimization algorithms attempts to optimize the loss function of a network with a desired objective, i.e minimize the loss function (Yousoff et al. 2016). The two optimization algorithms used in this thesis are Stochastic Gradient Decent and Adam.

Stochastic Gradient Decent (SGD)

One of the most common optimization functions is SDG, or *Stochastic Gradient Decent.* It is an iterative method that replaces the actual gradient from the entire dataset with an estimate, which is calculated from a randomly selected subset of data. This reduces the computational cost and speeds up iteration. The trade-off is a lower convergence rate (Loshchilov and Hutter 2017). SGD is defined in the following algorithm:

Algorithm 2 SGD-optimizer - (Loshchilov and Hutter 2017)

```
Require: Learning rate: $\alpha$, momentum factor: $\beta_1 \in \mathbb{R}$, Stochastic Objective Function with parameters $\theta \rightarrow f(\theta)$, Initial parameter vector: $\theta_0$, weight decay factor $\lambda \in \mathbb{R}$
Output: $\theta_t$ (Resulting parameters)
  Initialize:
  $m_0 \leftarrow 0$
  Schedule multiplier $\eta_{t=0} \in \mathbb{R}$
  $t_0 \leftarrow 0$                                          ▷ initialize the time-step
  while $\theta_t$ not converged: do
      $t \leftarrow t + 1$
      $\nabla f_t(\theta_{t-1}) \leftarrow$ getBatch($\theta_{t-1}$)   ▷ get the current batch and return the corresponding gradient
      $g_t \leftarrow \nabla f_t(\theta_{t-1})$
      $\eta_t \leftarrow$ SetScheduler($t$)   ▷ Set the scheduler, can be fixed, decay or used for warm restarts
      $m_t \leftarrow \beta_1 m_{t-1} + \eta_t \alpha g_t$          ▷ Correct first moment vector
      $\theta_t \leftarrow \theta_{t-1} - m_t$                      ▷ Correct parameters
  end while
```

Adam

Adam, or *Adaptive moment estimation*, computes individual adaptive learning rates for different parameters from estimates of first and second moments of the gradients. First introduced by (Kingma and Ba 2014), it has quickly become one of the most popular step size optimizer. The Adam algorithm is defined as the following:

Algorithm 3 Adam-optimizer - (Loshchilov and Hutter 2017)

```
Require: Learning rate: $\alpha$, Exponential Decay rates: $\beta_1, \beta_2 \in [0, 1)$, Stochastic Objective Function with parameters $\theta \rightarrow f(\theta)$, Initial parameter vector: $\theta_0$
Output: $\theta_t$ (Resulting parameters)
  $\rightarrow$ Initialize moment vectors:
  $m_0 \leftarrow 0$
  $v_0 \leftarrow 0$
  $t_0 \leftarrow 0$                                          ▷ initialize the time-step
  while $\theta_t$ not converged: do
      $t \leftarrow t + 1$
      $g_7 \leftarrow \nabla_\theta f_t(\theta_{t-1})$            ▷ Get gradients w.r.t stochastic objective at time t
      $m_t \leftarrow \beta_1 m_{t-1} + g_t(1 - \beta_1)$        ▷ Correct first moment
      $v_t \leftarrow \beta_2 v_{t-1} + g_t^2(1 - \beta_2)$      ▷ Correct second moment
      $\hat{m_t} \leftarrow \frac{m_t}{(1-\beta_1^t)}$            ▷ Compute bias-corrected first moment
      $\hat{v_t} \leftarrow \frac{v_t}{(1-\beta_2^t)}$            ▷ Compute bias-corrected second moment
      $\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m_t}}{(\sqrt{\hat{v_t}}+\epsilon)}$   ▷ Correct parameters
  end while
```

2.2 Performance

Performance of a network often comes from the fitting of the model, the measurement of how well the model generalizes to similar data of the training data. Model fitting is a vital part of any research that involve quantitative measurements and explore the relationship between two or more measures (Ying 2019). For machine learning, and specially deep learning, the two most notable performance issues are that of overfitting and underfitting.

2.2.1 Overfitting

Overfitting is a phenomenon that occurs when a model delivers excellent prediction results on data contained in the training set, but poor results on new unseen data in test sets.

2.2.2 Underfitting

Underfitting, on the other hand, is a phenomenon that occurs when a model delivers poor predicting results on both training data and test data, indicating its inability to both predict and generalize. In essence we say that if the model is underfitting it is not learning at all (H. Zhang et al. 2019).

2.2.3 Data

Data quality sets the upper bound of performance for any model (Jain et al. 2020). Assessing the quality of the data and how it is processed before inputted to a model is thus of vital importance. Failure to understand the data can result in inaccuracies and unsatisfactory predictions from the model. Data handling is apart of this assessment and securing that the data is logically split between training, validation and testing data is important.

Training Data

The training data defines the data directory solely used during the training process of the model. This directory contains a training data distribution X and a corresponding ground truth (GT), or target, distribution of Y. For image segmentation; X is the distribution of unique training images and Y is the distribution of the respective masks to the training images.

Validation Data

The validation data defines the data directory that is used for validating the model during training. This is very useful as it allows for live observation of how the model is progressing. The validation data should consist of images that are close to the real application of the model. At no point should data contained in the training data exist in the validation data as this will give a false narrative of how the model actually performs. The validation data is composed of a validation data distribution $\hat{X}$ and a corresponding ground truth distribution of $\hat{Y}$.

Testing Data

The testing data consists of real application data, meaning the actual data which the model is designed to be used on. The testing data can contain data that either includes the domain of target data or not. This to ensure the model is generalizing correctly and not producing false positive results.

Augmentation

Augmentation refers to the altering of data to the effect of generating more data. This means that images can be manipulated by utilizing transformations (i.e shear, crop, enlarge, flip, rotate, etc...) or image enhancements (i.e color saturation, blurring, sharpen, etc...) to create new images that differ from the original. Often these augmentations are performed at random to further increase the variety and amount of data. It is important to note that there is a risk of augmenting the data to a point where target object(s) in the images become too obscure. At that point the data becomes useless and could in fact be destructive for the model (Romero Aquino et al. 2017). Consequently, one must exercise caution when implementing data augmentations.

2.2.4 Hyperparameters

Hyperparameters are the controllable factors of a models learning process. This domain can consist of integer-valued(i.e Epochs, batch size), real-valued(i.e learning rate), categorical(i.e choice of optimizer) or binary(i.e if learning rate scheduler should be used or not) parameters. The tuning of these parameters are directly linked to the optimization of a model. Most of the hyperparameters used in this thesis are described below.

Epochs

Epochs, or *iterations*, represent the number of times the the model will be trained on the training data. During one epoch the entirety of the training data will have been given to the model exactly once. The number of epochs varies given a arbitrary model and should be chosen based on the size of the dataset and the used learning rate [source].

Batch Size

Batch size is defined as the amount of data the model works on before updating its internal parameters (i.e weights and biases). A batch of data is selected randomly from the training/validation/testing data and are given to the model until the dataset is exhausted. Note that even though the data is selected randomly an instance of data can only be selected once for each epoch. Batch sizes can be chosen at will, but common sizes are 16, 32, 64 or 128.

Learning rate

The learning rate is a tuning parameter linked to an optimizer algorithm. It determines the magnitude of movement towards a minimum of a loss function by setting the step size at each iteration. It is called learning rate because it influences the decision process of overriding old information with new information, thus the model metaphorically "learns". There is a trade-off here however, as a too high learning rate will potentially cause the "learning" to leap over a minimum and overshoot, but a too low learning rate may get stuck in a local minimum or use a monumental amount of time to converge.

Momentum

Momentum is a parameter of a activation function, most commonly used with gradient decent. Its purpose is to keep a consistent heading in the direction towards the minimum of a loss function. When the minimum is located in a narrow "valley", following the gradient direction can lead to oscillations of the search process. A simplified example of this is showed in figure Figure 2.10, comprised of a network with only two weights w_1 and w_2.

The idea of momentum builds on the theory that instead of following the negative gradient direction, computed from each new combination of weights, a *weighted average* of the current gradient and the previous correction direction is computed at each iteration. This means that, generally speaking ,if we set the momentum to be 0.5 we would have a new direction vector consisting of 50% of the old and 50% of the new, thereby safeguarding from drastically changing direction. When using a momentum during the backpropagation of a given network with k different weights $w_1, w_2, ..., w_k$, the i^{th} correction for weight w_k is given by:

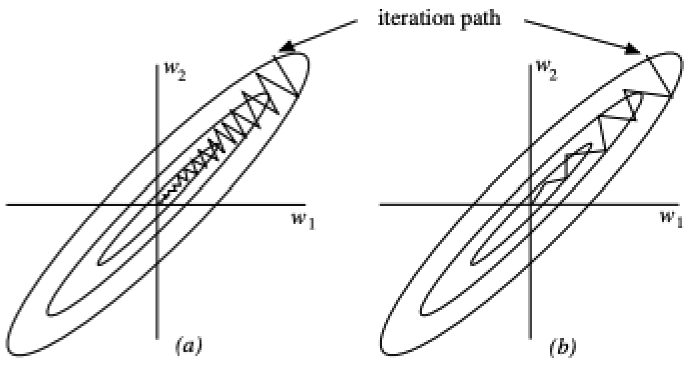


Figure 2.10: Backpropagation without momentum (a) and with momentum (b), note the difference in oscillation (Rojas 1996).

$$\Delta w_k(i) = -\gamma \frac{\delta \mathbf{L}}{\delta w_k} + \alpha \Delta w_k(i-1) \tag{2.7}$$

where γ and α are the learning rate and momentum rate, and $\mathbf{L}$ is the loss function.

Weight decay

Weight decay is a regularization technique applied on the optimization algorithm, and consequently on the loss function. It works by adding a penalizing term that shrinks the weights during back-propagation. Weight decay fights overfitting and the exploding gradient problem [source]. The most used regularization term is L2 penalty. Its formula is given below.

$$C = E + \frac{\lambda}{2n} \sum_{i=1}^{n} w_i^2 \tag{2.8}$$

L2 is applied to the loss function by adding the hyperparameter lambda (λ) multiplied by the squared sum of weights to the error term (E).

2.2.5 Measuring Accuracy

Dice Score

Dice score is a accuracy metric often used in classification problems, like image segmentation. It gauges the pixel-wise similarity between two discrete input images. The dice score equals twice the number of pixels common to both images divided by the combined total number of pixels in both images.

The formula for the dice score is given as:

$$Dice(y, \hat{y}) = \frac{2(y\hat{y})}{y + \hat{y}} \tag{2.9}$$

where, for image segmentation, y is the ground truth and $\hat{y}$ is the prediction.

2.2.6 Software Libraries

To realize the objective of this rapport, the following software libraries have been used. The usage of libraries greatly increases the speed of development by using functions, algorithms and methods already implemented. The results in this project has been garnered from the use of the high-level programming language of Python.

OpenCV

OpenCV stands for "*Open Source Computer Vision Library*" and is an open source software library for machine and computer vision. The library consists of more than 2500 classic and SOTA algorithms. It was built too provide a common infrastructure for computer vision applications. The library is focused on providing real-time CV functionality for vision based applications.

Pytorch

Pytorch is a open source machine learning library for computer vision and natural language processing developed by Facebook's AI research lab (FAIR). It was released in 2016 and has since been used as the foundation for many Deep Learning applications like Tesla's autopilot systems or Uber's Pyro. Pytorch stores the homogeneous multidimensional rectangular arrays of numbers in Tensors.

NumPy

NumPy has been regarded as the fundamental library for computing in Python. It makes use of a multidimensional array object, the *ndarray*, to perform fast operations. These operations include mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and more.

2.3 Relevant Mathematics

The following section briefly explains three possible representations of orientation in 3D space, **euler angles**, **rotational matrix** and **quaternions**.

Euler angles

Euler angles is a representation of 3D orientation using three angles of rotation, around three individual axes where the rotations are performed in sequence. The idea of euler angles is to decompose a single rotation, into these three successive rotational movements around fixed axes. Figure 2.11 show a single rotation decomposed into the rotation around z, x and y axes, with the angles ϕ, θ and ψ. The first rotation rotates around the z axis with the angle ϕ, the second rotation is around the rotated x-axis x' with angle θ, and the last rotation with angle ψ of the now rotated z-axis; z'. For an aircraft moving in positive x-direction the three euler angles ϕ, θ and ψ corresponds to yaw, roll and pitch.

Rotation matrix

Another representation of rotations is rotation matrices. In $\mathbb{R}^3$, coordinate system rotation around the x, y and z-axes in a counter clockwise direction looking in a negative direction towards the origin gives the following three matrices:

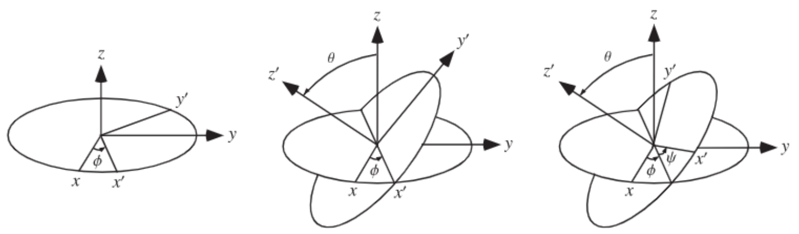


Figure 2.11: Euler angles (MathWorld 2022)

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\alpha & sin\alpha \\ 0 & -sin\alpha & cos\alpha \end{bmatrix} \quad \mathbf{R}_y(\beta) = \begin{bmatrix} cos\beta & 0 & -sin\beta \\ 0 & 1 & 0 \\ sin\beta & 0 & cos\beta \end{bmatrix} \quad \mathbf{R}_z(\gamma) = \begin{bmatrix} cos\gamma & sin\gamma & 0 \\ -sin\gamma & cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

All matrices are square, and as all columns form orthonormal systems, they are all orthonormal matrices meaning the transposed rotation matrices are equal to the inverse of the rotation matrices. Similar to euler angles, the rotation of an object using rotation matrices could be thought of as a sequential operation involving three rotations around each axis. As matrix multiplication does not commute, the order of which one multiplies the matrices will affect the result. The combined rotation R, if first rotated around the x-axis, then the y-axis and lastly the z-axis corresponds to the matrix product $\mathbf{R}_x(\alpha)\mathbf{R}_y(\beta)\mathbf{R}_z(\gamma)$. Each column in the resulting matrix R could be interpreted as to how the different x,y and z components of each vector is transformed during the rotation.

Quaternion

A third way of representing a rotation is using quaternions. Based on complex numbers, quaternions is a four-scalar set and is represented by a 4-dimensional vector. A quaternion Q is defined as a complex number

$$Q = a_0 + a_1\mathbf{i} + a_2\mathbf{j} + a_3\mathbf{k}$$

Formed from four different units (1, **i**, **j**, **k**) where $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = -1$ and the real numbers a_i (where i = 0,1,2,3). So essentially a quaternion consists of both imaginary and real numbers.

2.4 Pinhole Camera Model

The drop from a three-dimensional world to a two-dimensional image is a projection process in which we loose one dimension. A common way of modelling this process is by defining a ray from a point in 3D space through a fixed point, which is the center of projection. A plane is specified as an image plane, and the intersection of the ray and the image plane represents the 2D image projection of the 3D point (Hartley 2003, p. 7). This means that all points in a ray passing through the center of projection, can be considered equal for the purpose of image projection. Furthermore, we can consider all the image points as the set of rays through the center of projection. As a camera is a mapping between the 3D world and a 2D image, there are different models that represents the camera mapping . Hartley (2003, p. 153) lists several, including the one that will be used in this thesis, which is the pinhole camera model. This model describes a camera made as a chamber with an aperture in the front, defining an optical center of the camera. Rays of light passing through the aperture projects onto the back wall of the chamber which is called the image plane. A distance f from the image plane to the optical center is called focal length (Semeniuta 2016). In the further explanation of the mathematical relationship between coordinates of a point in 3D space and the projection onto the image plane, the pinhole camera model will be visualised as shown in Figure 2.12.

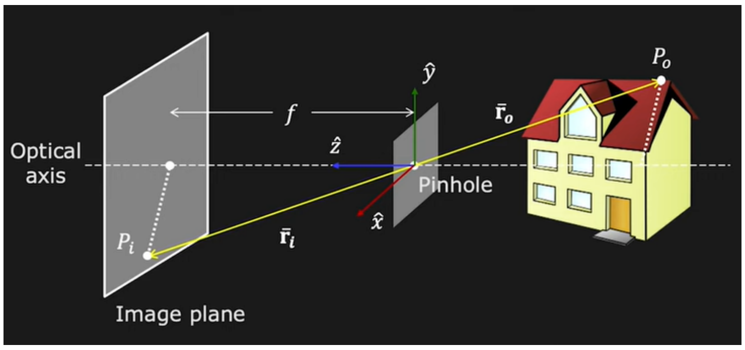


Figure 2.12: Perspective imaging with a pinhole (Nayar 2021b)

From Figure 2.12 we can see that there is a single ray travelling from point $\mathbf{P}_0$ located on the house, through the pinhole and projects onto the point $\mathbf{P}_i$ on the image plane. A coordinate system is defined with its origin at the pinhole, with the $\hat{\mathbf{z}}$ axis pointing along the optical axis, which is the axis that is perpendicular to the image plane shown in Figure 2.12 as a dotted line. With this defined, we can write the the points $\mathbf{P}_0$ and $\mathbf{P}_i$ using the vectors $\bar{\mathbf{r}}_0$ and $\bar{\mathbf{r}}_i$ defined as:

$$\bar{\mathbf{r}}_0 = (x_0, y_0, z_0) \qquad \bar{\mathbf{r}}_i = (x_i, y_i, f)$$

Two triangles are formed by the vectors $\bar{\mathbf{r}}_0$ and $\bar{\mathbf{r}}_i$. Using the rules of similar triangles, and since $\bar{\mathbf{r}}_0$ and $\bar{\mathbf{r}}_i$ are vectors we can break it down into its components and get the following:

$$\frac{x_i}{f} = \frac{x_0}{z_0} \qquad \frac{y_i}{f} = \frac{y_0}{z_0}$$

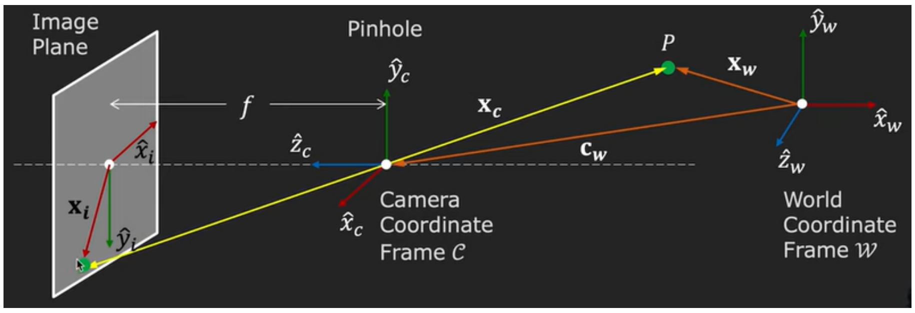


Figure 2.13: Mapping from world coordinate frame, to its equivalent projection on the image plane (Nayar 2021a)

Looking at Figure 2.13, and assuming we know the the relative position and orientation of the camera coordinate frame $\mathbf{C}$ with respect to the world coordinate frame $\mathbf{W}$, then the goal is to write an expression that transforms the known point $\mathbf{P}$ in world coordinates, to its projection $\mathbf{X}_i$ on the image plane. The next section describes this process, and will be explained in two sections. First the transformation of camera coordinates to image coordinates will be explained. This type of projection from 3D to 2D coordinates is called a perspective projection (Sturm and Deguchi 2021). Following this, the coordinate transformation from world coordinates to camera coordinates is explained. The subscripts c and w is used to indicate the use of camera coordinate system $\mathbf{C}$ and world coordinate system $\mathbf{W}$ respectively.

Perspective projection

Assuming we know the point $\mathbf{P}$ in camera coordinate frame and define the coordinates as $\mathbf{X}_c$. Using rules of similar triangles as described above, the coordinates of the projection of the point $\mathbf{P}$ onto the image plane is then:

$$x_i = \frac{x_c}{z_c} f \qquad\qquad y_i = \frac{y_c}{z_c} f$$

Coordinates x_i and y_i are in this case continuous, and in the same unit as point $\mathbf{X}_c$. However, in reality there is an image sensor that is divided into discrete pixels (Hata and Savarese 2021). This change of units is handled by introducing the parameters m_x and m_y, which has the units of for instance $\frac{pixels}{mm}$ and is called the pixel density in x and y axis, respectively. Two variables are introduced, as the number of sensor on the x and y axis of the image sensor might differ. If the aspect ratio is equal to one, then $m_x = m_y = m$ and the camera is said to have square pixels (Hata and Savarese 2021). The second set of parameters introduced describes how the origin of the image plane and pixel coordinate system might differ. Image plane coordinates have its origin in the middle of the plane, where the optical axis pierces the plane, as shown in Figure 2.13. The point where the optical axis pierces the image plane is called the principal point (Hartley 2003). The two variables o_x and o_y are introduced, and they are the pixel coordinates where the optical axis pierces the image sensor. In Figure 2.14, the image plane is shown with the origin of the image sensor located in the lower-left corner of the image.

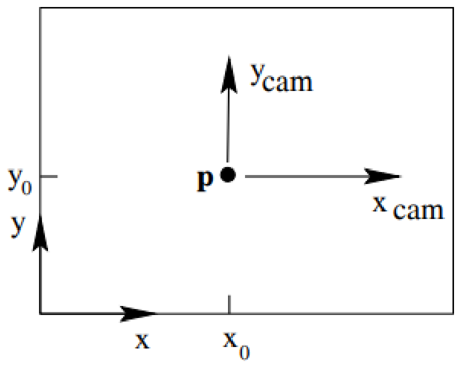


Figure 2.14: Image plane with the origin of the image sensor located in the lower-left corner of the image (Hartley 2003, p. 155)

Defining u and v as the pixel coordinates in x_i and y_i directions, respectively, then the pixel coordinates are:

$$u = m_x x_i + o_x = m_x f \frac{x_c}{z_c} + o_x = f_x \frac{x_c}{z_c} + o_x \qquad \text{where } f_x = m_x f$$

$$v = m_y y_i + o_y = m_y f \frac{y_c}{z_c} + o_y = f_y \frac{y_c}{z_c} + o_y \qquad \text{where } f_y = m_y f$$

In this section, four parameters have so far been defined, and they are intrinsic parameters of a camera. They are unique and represent the cameras internal geometry (Semeniuta 2016). Two parameters are currently missing from this formulation: skewness and distortion. As formulated in Hata and Savarese (2021), we often say that an image is skewed when the camera coordinate system is skewed, meaning that the angle between the two axes is slightly larger or smaller than 90 degrees. Distortion causes the image magnification to increase or decrease as a function of the distance to the optical axis (Hata and Savarese 2021). Both the distortion effect and skew of images are left outside the scope of this thesis. Therefore, f_x, f_y, o_x and o_y are the only intrinsic parameters considered.

However, our projection from camera coordinates to pixel coordinates is not a linear transformation (as z_0 is the denominator). One way of solving this problem is to change coordinate system (Hata and Savarese 2021). To represent the mapping from world coordinate frame to camera coordinate frame, the representation of homogeneous coordinates is used. A new coordinate system is introduced such that any point $\mathbf{A}' = [x', y']$ becomes $[x', y', 1]$ and $\mathbf{A} = [x, y, z]$ becomes $[x, y, z, 1]$. As stated in Hata and Savarese (2021) this augmented space is referred to as the homogeneous coordinate system. The equality between a vector and its homogeneous coordinates occurs when the final coordinate equals one. Converting from the homogeneous coordinates $(v_1, ..., v_n, w)$ to Euclidean coordinates results in the coordinates $(\frac{v_1}{w}, ..., \frac{v_n}{w})$. Introducing homogeneous coordinates simplifies calculations, such as representing the perspective projection as a linear transformation which allows it to be represented as a matrix-vector product (Hata and Savarese 2021). The calculations are simplified, as the point $\mathbf{A'}$ which in homogeneous coordinates is $\mathbf{A'}$ =[x',y',1], is in homogeneous coordinates equivalent to the point [2x',2y',2]. In homogeneous coordinates, the point $\mathbf{A'}$ is equivalent to any point [kx',ky',k] for any non-zero value of k (Hartley 2003). Using this, we can formulate the point [u, v] in homogeneous coordinates as:

$$\begin{bmatrix} u \\ v \end{bmatrix} \equiv \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \mathbf{M}_{int} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \mathbf{K}[\mathbf{I}0] \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \tag{2.11}$$

Where $\mathbf{K}$, as it is often denoted in literature (OpenCV 2022), is the camera intrinsic parameter matrix. This matrix completes the projection of a homogeneous representation of a point in camera coordinate frame in 3D to its homogeneous 2D pixel coordinate representation.

Coordinate transformation

In this section the 3D to 3D coordinate transformation of a point $\mathbf{X}_w = [x_w, y_w, z_w]^\top$ in world coordinates to a point $\mathbf{X}_c = [x_c, y_c, z_c]^\top$ in camera coordinates is given. This can be done using what is called the extrinsic parameters (Hata and Savarese 2021) of the camera, which is the position $\mathbf{c}_w$ and orientation $\mathbf{R}$ of the camera coordinate frame $\mathbf{C}$, in the world coordinate frame $\mathbf{W}$. $\mathbf{R}$ is the rotation matrix describing the rotation of the system, and is defined as the matrix product of the rotation of each axis in world coordinates, as described in **??**.

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.12}$$

Considering point $\mathbf{P}$ in Figure 2.13, the vector $\mathbf{X}_c$ is $\mathbf{X}_c = \mathbf{X}_w - \mathbf{c}_w$ in world coordinate frame, where multiplying with the rotation matrix $\mathbf{R}$ computes the transformation from world coordinates to camera coordinates (Hata and Savarese 2021). The vector $\mathbf{t}$ is defined as the translation vector and is used to describe the movement in each of the three axes (Hata and Savarese 2021). Meaning that the vector $\mathbf{X}_c$ in camera coordinate frame is:

$$\mathbf{X}_c = \mathbf{R}(\mathbf{X}_w - \mathbf{c}_w) = \mathbf{R}\mathbf{X}_w - \mathbf{R}\mathbf{c}_w = \mathbf{R}\mathbf{X}_w + \mathbf{t} \qquad \text{where } \mathbf{t} = -\mathbf{R}\mathbf{c}_w = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Writing vector $\mathbf{X}_c$ in homogeneous coordinates $\tilde{\mathbf{X}}_c$, with $\mathbf{t} = -\mathbf{R}\mathbf{c}_w = [t_x, t_y, t_z]^\top$ and $\tilde{\mathbf{X}}_c$ being the vector $\mathbf{X}_w$ in homogeneous coordinates, results in the following equation:

$$\tilde{\mathbf{X}}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \mathbf{M}_{ext}\tilde{\mathbf{X}}_w = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \tag{2.13}$$

The matrix $\mathbf{M}_{ext}$ is the extrinsic matrix, and contains the external parameters and does not depend on the camera itself (Hata and Savarese 2021). $\mathbf{M}_{ext}$ is a homogeneous transformation matrix (Dain Lin and Hwang 1994), and it completes the transformation of a homogeneous representation of a point $\mathbf{P}$ in world coordinate frame, to its equivalent representation in camera coordinate frame in homogeneous coordinates.

This completes the mapping of a 3D point $\mathbf{P}$ from world coordinate frame, to its equivalent projection on the image plane. Combining Equation 2.11 and Equation 2.13 results in the full projection:

$$\begin{aligned}\tilde{\mathbf{u}} = \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} &= \mathbf{M}_{int}\mathbf{M}_{ext}\tilde{\mathbf{X}}_w = \mathbf{K}[\mathbf{I}0]\mathbf{M}_{ext}\tilde{\mathbf{X}}_w \\ &= \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \end{aligned} \tag{2.14}$$

2.5 Perspective-n-Point

Estimating the 3D translation and rotation of objects with respect to the camera is an important task in various robotics applications, such as boosting a robots navigation performance in various settings. It can also be utilized in scenarios beyond robotics such as Virtual reality (VR) and Augmented reality (AR) (Ge and Loianno 2021). There are multiple approaches to pose estimation. The Perspective-n-Point problem (PnP), was introduced in Fischler and Bolles (1981), as a problem of estimating the position and orientation of a camera given its intrinsic parameters and a set of n known 3D points and their 2D projections on the image plane (Lepetit et al. 2008). Estimating the pose using the translation of 3D to 2D points requires a certain amount of 3D points. The minimal PnP problem, as refered to in Hesch and Roumeliotis (2011), is the problem of estimating the pose of the camera using $n = 3$, and is often referred to as the P3P problem. As demonstrated below this problem yields multiple solutions and therefore $n > 3$ is needed in order to obtain a unique solution.

2.5.1 P3P

An object without any constraints in 3D space has 6 degrees of freedom. Three degrees corresponds to a translational movement along the x,y and z axis, and three degrees corresponds to the rotational movement (yaw, pitch, roll) around each axis. Resulting from this is that an object with 0 fixed point has 6 degrees of freedom. Hence, the P0P problem has no solution as there is not enough data to estimate the position nor rotation.

An object with one fixed point has 4 degrees of freedom, meaning that one fixed point removes two degrees of freedom. One can visualize this by picturing a cube in an image, where one of the corners are in fixed position. This means that there are no limitations to the rotational directions of the cube, resulting in three degrees of freedom. In addition, the cube can move along the perspective projection line, as visualized in Figure 2.13.

Two fixed points results in 2 degrees of freedom, as there are rotational limitations where the cube with two fixed points can only rotate around one axis. As with one fixed point, the cube can still move along the perspective projection line and therefore the total degrees of freedom is 2. Three fixed points results in 0 degrees of freedom, as the cube can no longer do any rotational or translational movements. Yet the P3P problem, where three known 3D are present results in 8 different solution when estimating the pose.

Figure 2.15 illustrates pose estimation with three points, where $\mathbf{X}_0$ is the camera position, and $\mathbf{X}_i$ where $i = 1, 2, 3$ are positions known both in 3D and 2D coordinates. Using the known position of

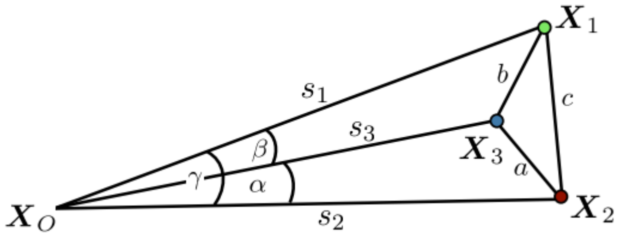


Figure 2.15: Pose estimation with three known points (Stachniss 2021).

the 3D points, the angles θ, γ, α and the distance a, b and c can be calculated. This information can then be used in a set of equations resulting in a fourth degree polynomial with 8 solutions. It can be shown that four solution are on either side of the image plane, resulting in only four feasible solutions. A visualization of these four solutions can be viewed in Figure 2.16, where $a = b = c$ and $\theta = \gamma = \alpha$. Where instead of moving $\mathbf{X}_0$, resulting in a tilting of the triangle ($\mathbf{X}_1$, $\mathbf{X}_2$, $\mathbf{X}_3$), the position of the known points are changed for visualization purposes.

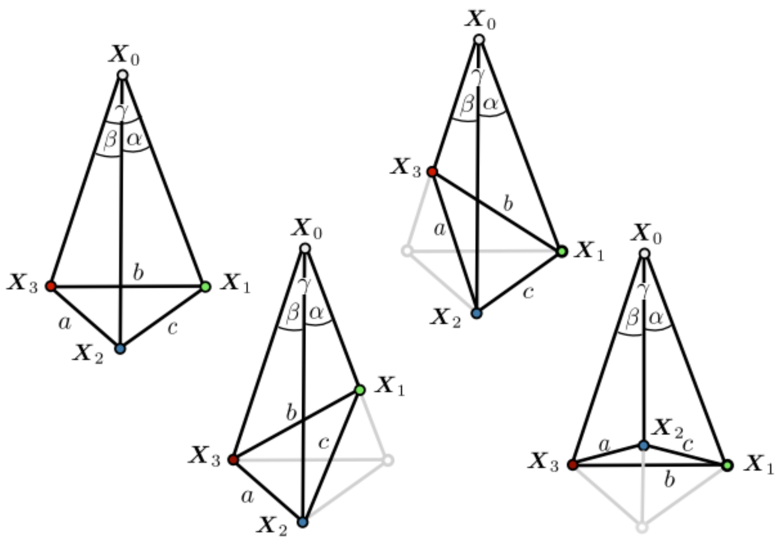


Figure 2.16: Possible solutions of P3P problem with $a = b = c$ and $\theta = \gamma = \alpha$ (Stachniss 2021)

Therefore, deriving a unique solution to the PnP problem requires more than three points. In general there are different methods of solving the PnP problem, and the solutions can be classified as iterative and non-iterative methods (S. Li et al. 2012). As stated in Lepetit et al. (2008) and S. Li et al. (2012), the non-iterative methods are efficient and computationally faster. The iterative methods rely on minimizing a criterion such as a cost function. The iterative methods in general have a higher computational cost but can achieve great results when converging properly.

2.5.2 EPnP

EPnP is a non-iterative solution to the PnP problem, and was presented in Lepetit et al. (2008), as a method for quickly solving the PnP problem while performing well. The central idea of the method is to represent the known 3D points as a weighted sum of four virtual control points, thereby reducing the problem to estimating the coordinates of these control points in the camera referential (Lepetit et al. 2008). The following subsection is based on EPnP method presented in Lepetit et al. (2008).

Assume a set of n points, where the 3D coordinates in world coordinate frame and their 2D image projections are know, be represented as reference points $\mathbf{p}_i$ where i=1,...,n. These known points and their projections are then represented as a set of four control points $\mathbf{c}_i$ where i=1,...,4. The reference points are represented as a weighted sum of the control points, where α_{ij} are homogeneous barycentric coordinates:

$$\mathbf{p}_i^w = \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^w \qquad \text{where} \quad \sum_{j=1}^{4} \alpha_{ij} = 1 \tag{2.15}$$

Next, the projection from 3D reference points in camera coordinate system to 2D projections is formulated. As the same relation in Equation 2.15 holds for both world and camera coordinate system we can formulate the projection as:

$$\forall i \quad , \quad w_i \begin{bmatrix} \mathbf{u}_i \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{p}_i^c = \mathbf{K} \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^c \tag{2.16}$$

Where $\mathbf{K}$ is the camera intrinsic matrix, same as Equation 2.11, w_i are scalar projective parameters, and $\mathbf{u}_i$ is the known 2D image projection of the $\mathbf{p}_i$ reference points. Inserting the camera intrinsic matrix, with the parameters f_x, f_y and the coordinates o_x, o_y of the principal point, and expanding the expression by considering the specific 3D coordinates of each control point $\mathbf{c}_j^c$ results in the following equation:

$$\forall i \quad , \quad w_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \sum_{j=1}^{4} \alpha_{ij} \begin{bmatrix} {x_j}^c \\ {y_j}^c \\ {z_j}^c \end{bmatrix} \tag{2.17}$$

With 4 control points, Equation 2.17 is a linear system with a total of 12 + n unknowns. 12 control point coordinates and n projective parameters w_i where $i = 1,...,n$. From the last row in Equation 2.17 we can write $w_i = \sum_{j=1}^{4} \alpha z_j^c$, and substitute this expression in the first two rows resulting in two linear equations for each reference point:

$$\sum_{j=1}^{4} \alpha_{ij} f_x {x_j}^c + \alpha_{ij}(o_x - u_i) z_j^c = 0 \tag{2.18}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_y {y_j}^c + \alpha_{ij}(o_y - u_i) z_j^c = 0 \tag{2.19}$$

By arranging the coefficients of these equations, for each reference point, a matrix $\mathbf{M}$ and a linear system on the form $\mathbf{Mx} = \mathbf{0}$ is generated. Where $\mathbf{x} = [\mathbf{c}_1^{c\top}, \mathbf{c}_2^{c\top}, \mathbf{c}_3^{c\top}, \mathbf{c}_4^{c\top}]^\top$ and $\mathbf{M}$ is a $2n{\times}12$ matrix. The solution of the PnP problem is in the null space of $\mathbf{M}$, and can be expressed as:

$$\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i$$

where $\mathbf{v}_i$ are the columns of the right-singular vectors of $\mathbf{M}$ corresponding to the N null singular values of $\mathbf{M}$. They can be found as the null eigenvectors of matrix $\mathbf{M}^\top\mathbf{M}$.

2.6 RANSAC

In this section, a general introduction of the fundamentals of RANSAC is presented, while a more thorough presentation of how the method was adjusted to and utilized in this thesis specifically is presented in Section 3.3.2. Introduced in Fischler and Bolles (1981), Random Sample Consensus (RANSAC) is an iterative method to detect outliers in measurements/data points (Saini and Maity 2022). The method is widely used in computer vision, such as in pose estimation, where Peng et al. (2018) and Yu et al. (2020) uses the method to create 2D to 3D correspondences of keypoints using a similar method as presented in this thesis. RANSAC is also used in Lv et al. (2015) for camera calibration, which in essence is the process of estimating the parameters in the intrinsic camera matrix in Equation 2.14. RANSAC is not only used in computer vision, for instance it is used in Saini and Maity (2022) in detection of faulty sensors by detecting outliers in sensor data.

The methods use a hypothesize-and-test framework by randomly sampling a subset of the measurement set to detect the model that maximizes the number of valid data points (Heinrich 2013). A data point is randomly sampled from the measurement set, estimating a hypothesis using the randomly sampled data. The estimated hypothesis is evaluated on the remaining data points. The number of remaining valid data points, which satisfies the estimated hypothesis is called inliers, while the invialid data points is calles outliers (Heinrich 2013). This process is repeated by drawing new samples from the measurement set, calculating the consensus of different models with the complete measurement set. The model with the highest consensus is returned as the final model. In algorithm 4, the algorithm for the general RANSAC process is described.

Algorithm 4 Ransom Sample Consensus RANSAC, adapted version obtained from Hartley (2003, p. 118)

```
Require: Dataset $S$ which contains outliers and Threshold values $t$, $T$ and $N$
    1. Randomly select a sample of $s$ data points from $S$ and instantiate the model from this subset.
    2. Determine the set of data points $S_i$ which are within a distance threshold $t$ of the model. The set $S_i$ is the consensus set of the sample and defines the inliers of $S$.
    3. If the size $S_i$ (the number of inliers) is greater than some threshold $T$, re-estiamte the model using all the points in $S_i$ and terminate-
    4. If the size of $S_i$ is less than $T$, select a new subset and repeat the above.
    5. After $N$ trials the largest consensus set $S_i$ is selected, and the model is re-estimated using all the points in the subset $S_i$.
```

Chapter 3

Methodology

A research methodology could be defined as a collection of research methods that is being used by the researcher, to systematically approach the research objective. A research methodology is typically quantitative or qualitative (Kothari 2004). In this chapter the quantitative methodology used in this research is described, and how and why the methods used address the research objective. The data collection method are presented in Section 3.1, where various alternatives were evaluated, and a synthetic dataset was generated using Unity. The subsequent sections describes the technologies used to enable pose estimation. In Section 3.3 the method of vector field generation combined with the usage of RANSAC and PnP in order to estimate pose is described.

3.1 The 6D Pose Estimation Dataset

Deep learning requires well developed and efficient algorithms to train the network and large amounts of data and computational power to run the software. Therefore, deep learning has greatly benefited from the converging trends in development and availability of algorithms to train the network, and advancements in for instance graphical processing units (GPUs). However, as the networks requires large amounts of data to optimize its parameters, this has lead to the availability of well labeled and structured data being the bottleneck in the development of deep learning models. (de Melo et al. 2022).

One way of obtaining data is to use publicly available datasets. These datasets could be used for many different problems such as image segmentation and classification, but as illustrated in Nikolenko (2021, p. 3), challenges might occur if the requirements of the researchers are something beyond the classes, settings or conditions presented in this data. Using an example of ImageNet Data (Lab 2021), which is a database containing images for classification and localization of 1000 object classes. Naturally, if one requires images of a class that is not in the dataset then you will not find relevant data, but there are also other less obvious situations where difficulties with these types of datasets might occur. For instance if a specific research requires images with a specific lighting or camera angle.

There is also existing 6D pose estimation datasets such as the LINEMOD (BOP 2022) used in for instance Bukschat and Vetter (2020). LINEDMOD is a widely-used benchmark RGB+D dataset, where RGB+D indicates that the dataset contains not only RGB images but also data describing three dimensional information about the images, such as depth visualizations. LINEMOD contains training, validation and test images, as well as depth and masks images describing the object silhouettes. The set also contains 3D object models, camera parameters (intrinsic camera matrix) and ground truth annotations (object id, 3x3 rotation matrix and 3x1 translation vector). 2D bounding box of the object silhouette given in image coordinates and described with height and width, 2D bounding box of the visible part of the object silhouette and the number of pixels in object silhouette are also found in the dataset. Similar data is available in T-LESS, which is a benchmark dataset analogous to LINEMOD (BOP 2022). Both LINEMOD and T-LESS are relevant in 6D pose estimation problems presented in this report, but the similar issues with predefined datasets arise as previously explained. For instance does LINEMOD contain annotations, images and data relevant for object post estimation, but consists of 15 object models, where the range of camera to object distance is between 601 - 1102 mm (BOP 2022), and neither LINEMOD nor T-LESS contains models of shipping containers.

In contrast to using publicly available data, a project specific dataset could be fabricated. A notable example of this would be to calculate and annotate the data fields in LINEMOD, listed in the previous section. It is worth mentioning that datasets are co-dependant on the method chosen for the pose estimation, meaning that all data listed are not necessarily used in the process of training a model. In the following subsections, the paper moves on to describe in greater detail how a dataset could be designed and fabricated. In Section 3.1.1, examples of methods and tools used collect and annotate images are given. In Section 3.1.2, the definition of synthetic data is given in detail, as well as the synthetic data generation conducted for this thesis.

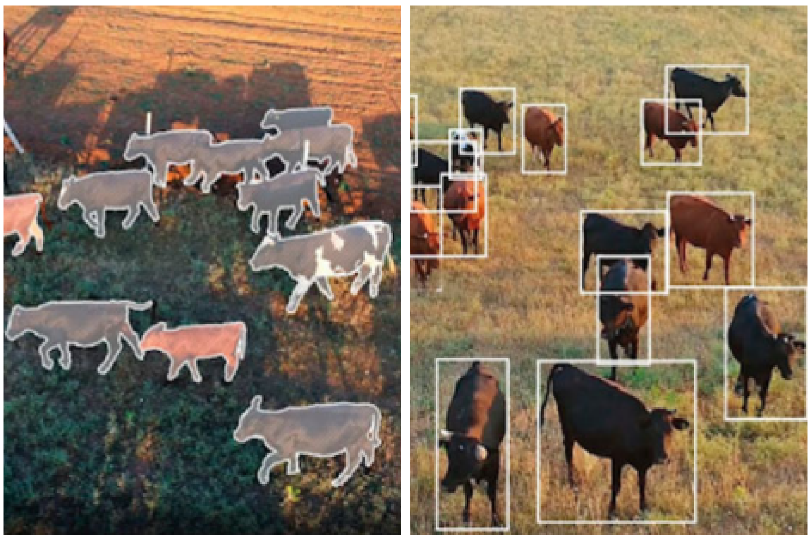

(a) Instance segmentation (b) Object detection

Figure 3.1: Ground truth image annotation (Nikolenko 2021, p. 3)

3.1.1 Data collection

Manually annotating a large amount of images could be an extensive amount of work. Defining how many images to include in a dataset during training, to achieve a well performing model, could be challenging as it does not solely depend on the number of images (Shahinfar et al. 2020). Further stated in Shahinfar et al. (2020), model metrics and parameters during training, and the number of classes to classify highly affect the model performance. Shahinfar et al. (2020) still concludes that model performance metrics such as accuracy, precision and recall all have a logarithmic relationship with the number of images, meaning that the performance of the model increase as the number of images increase. In order to annotate a dataset of sufficient size, one can imagine the amount of work needed to label images with for instance object detection as shown in Figure 3.1b, or instance segmentation (Figure 3.1a). Manually labeling images with accurate depth data or optical flow estimations is in many cases close to impossible.

Naturally, there has been developed tools to simplify the process of annotating images. Software tools such as MakeSense.AI[1], which was used in the preceding specialization project on object detection, allows for annotation such as bounding box labeling, classification labels and instance

[1]https://www.makesense.ai/

labels. Similar manual annotation tools could be found in V7 Labs[2] and Supervisely[3]. Despite the development of manual software tools to streamline the process of annotating images, a lot of time is required to develop a high quality dataset. Lin et al. (2014) presents the development of the Microsoft Common Objects in COntext (MS COCO) dataset, and exemplifies the user interface of the workers who annotated the images. Even though using a manual software tool the total time required to annotate the enormous 328 000 image, 2.5 million labels data set, was around 70 000 working hours. It is worth mentioning that this dataset consists of instance segmentation and category labels only. Some software tools further aids the user in the annotation process. V7 Labs presents an auto-annotation software using machine learning and computer vision to recognise parts of images that is being annotated by the user. Supervisely's software will suggest a segmentation, and the human annotator is only supposed to handle the errors by click individual pixels that are segmented incorrectly. But as pointed out in Nikolenko (2021, p. 3), it could take minutes per photo to handle the mistakes, and with a large dataset consisting of potentially thousands of images this adds up to a lot of time spent on labelling only.

There are other ways of overcoming the challenges of annotating 6D pose data however. Ge and Loianno (2021) introduces an automatic labeling technique, by using a Vicon[4] motion capture system to label images with 6D object pose data, where both the position and orientation of each object was obtained at 100 Hz. The tracking and data obtaining of the objects were made possible by attaching grey markers on the objects that were used by the motion capture system for localization. Another alternative is to outsource the manual process of collection and annotation of data. Services such as V7 labs Data Annotation[5] and crowdsourcing marketplaces such as Amazon Mechanical Turk[6], which was used in the development of MS COCO (Lin et al. 2014), enables a collection and annotation of potentially massive amounts of data. Yet these services are not free, and even though there are several companies that offers them, most likely with a wide range of competence and knowledge, as stated in Movshovitz-Attias et al. (2016) labour markets often lack expert knowledge potentially making some challenges and calculations difficult to complete.

Domain Randomization

An important aspect when collecting data, is how to transfer the model from source domain, which is the domain the model is trained in, to target domain (i.e. physical world). Essentially, a model transferred from source domain to target domain is bridging the gap between training and testing, by having a model performing in a generalized manner to unseen real world data (Weng 2019). Domain randomization aims to produce data samples from a large variation of the possible image space. The samples must not follow any real-world observed distribution, hence possibly producing just as many extreme real-world outliers (Valtchev and Wu 2021). By doing this, aiming to train models to detect precisely what features identify each class, regardless of how feasible each sample is.

Synthetic Data

An alternative to collecting and labeling real data, is to create artificial, or synthetic data, that mimics real-world observations. Nikolenko (2021) gives the following definition of synthetic data: *"Any data produced artificially with the purpose of training machine learning models"*. Within the field of computer vision there are many use cases and ways of utilizing synthetic data. With the steady increase in computational power, the lowered cost of rendering software, and the availability of 3D CAD models, for the first time it is now becoming possible to fully model not just the object of interest, but also its surrounding environment (Movshovitz-Attias et al. 2016). Movshovitz-Attias et al. (2016) presents a fully rendered datasets as a way of providing nearly limitless data, and states that rendering engines offer the promise of perfect labels in addition to the data. Data fields describing what the precise camera pose is, the precise lighting location, temperature and distribution, and what the geometry of the object is could be accurately and easily extracted.

[2]https://www.v7labs.com/annotation
[3]https://docs.supervise.ly/
[4]https://www.vicon.com/
[5]https://www.v7labs.com/labeling-service
[6]https://www.mturk.com/

Other projects use a combination of synthetic and real data. An example is Neumann-Cosel et al. (2009), which combines separate real and rendered images to train a lane tracking algorithm. Combining the artificial data with real-world images is done in Dwibedi et al. (2017) where cropped image object instances are places within the real image to train an image segmentation model, while Josifovski et al. (2018) uses a similar tactic but places 3D models in real-world images to estimate object pose. In both these cases the annotations and necessary data are extracted automatically without any need for human assistance. This is proving to be one of the great advantages with synthetic data, since the pipeline has information about the scene details, it can automatically generate error-free ground-truths (de Melo et al. 2022). In addition, manual making pixel perfect annotations is so labour intensive that it might not be worth the effort. This could result in annotations where information is left out. As exemplified in Nikolenko (2021, p. 11) and visualized in Figure 3.2, samples from the aforementioned MS COCO dataset shows that the segmentation mask consists of rougher lines and that many finer features are missed.

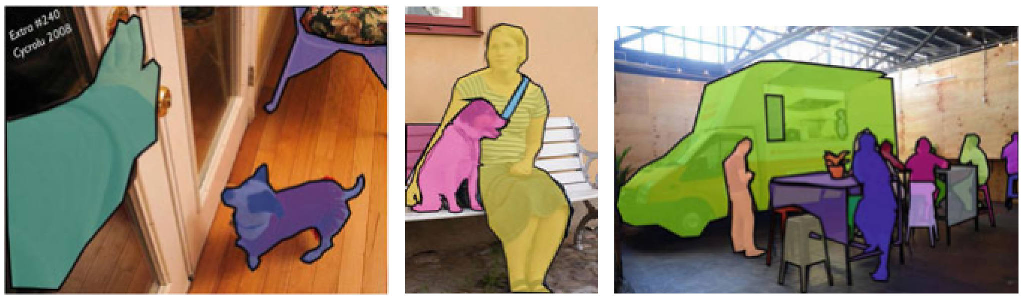


Figure 3.2: MS COCO instance segmentation examples (Nikolenko 2021, p. 13).

3.1.2 Sythetic Data Generation

The study required a varied and large amount of data. In addition to images used to train the network; image masks, keypoint coordinates and translation and rotation data of containers and camera was necessary. Gathering a large number of images, calculating the necessary pose data and segmenting all of them would include a large amount of work. To the best of our knowledge, there exist no publicly available datasets which contains images relevant for this project, while also containig the necessary annotations for the 6D pose estimation problem presented.

As outsourcing annotation work was not an alternative due to financial limitations, and computational capacity during training was not an issue, a synthetic dataset was fabricated. With the objective of thesis being a to implement a pipeline able to detect and estimate pose of shipping containers, as specified in Section 1.2, the approach of synthetic data is particularly useful. A fully rendered image allows for varied scenes, lighting, weather and other external factors without increasing the workloads drastically, these changes can be made in the modelling software and then used to generate a dataset. The synthetic dataset presented in this thesis is named SINTEF 6DPE ISO Container Dataset.

As stated previously *domain randomization* is an important aspect in creating a model able to generalize, and thereby limiting the gap between the source domain, which is the simulator, and the target domain which is real world images. As later described the SINTEF 6DPE ISO Container Dataset consist of images in an oceanic setting, as can be seen in Figure 3.4. An advantage of generating synthetic data is that if needed, the scene environment could be changed, for instance to a more arid environment, and new data can then be quickly generated. Also, other 3D models can be added to the scene and generated at random meaning that for instance an ROV object model could replace the ISO container in a new dataset, with minor changes in the code and modelling software. Therefore, constructing a fully rendered dataset was decided, based on the fact that a fully generated scene is flexible for future changes without large changes and workloads.

There are of course challenges with synthetic data, challenges related to how well the model performs on real-world images, compared to the data the model is presented during training. This issues are discussed in most sources including Dwibedi et al. (2017), Neumann-Cosel et al. (2009), Nikolenko (2021) and Movshovitz-Attias et al. (2016). While discussing challenges, the results presented in these sources signifies that there is potential in using this technology in novel area of research.

Widely used 3D modelling softwares are for instance Unity, Blender, Maya and Unreal engine (Aranjuelo et al. 2021). As our team had previous experience with Unity we decided to generate the data set in Unity. This is a multi platform games creation and 2D and 3D drawing tool (Unity 2021e), famous for being used to create a large variety of popular games (Unity 2021d). The software also has other uses cases such as film animation and AR/VR tools used for education, architecture and construction (Unity 2021d).

In Unity, a scene is constructed where various objects are placed and manipulated either by physical interaction in the Unity Editor or by scripting of the objects. The programming language used is C# and Unity operates with object-oriented scripting languages. Unity runs in a loop reading all of the data that is in the scene, and calling a function named Update() in each frame, meaning all code written and called in the Update() function run once in each frame, unless otherwise specified (Unity 2022). Each project in Unity has a render pipeline which performs a series of operations that take the content of a scene and displays them on a screen. At a high level these operations are: culling, rendering and post-processing. Unity has three prebuilt render pipelines with different characteristics (Unity 2021a). In this project the Buildt-in Render Pipeline was used, as the extra customization provided in the other prebuilt pipelines, Universal Render Pipeline (URP) and Heigh Definition Render Pipeline (HDRP), was deemed as unnecessary for this project. This assessment was made as only single images would be generated and the images generated were evaluated to be sufficiently realistic to train a model. Each image was rendered in perspective view to increase the realism of the images.

At each frame a specified number of container objects, with a randomly generated colour could be generated in front of the camera. The dataset presented in this thesis consists of images and data where one container is generated at each frame. Relevant data fields are stored in a *.txt* file, and two images are saved. Unity physics engine was used to prevent object generated from colliding with static objects placed in the scene and other randomly generated objects. In the case of generating multiple objects at each frame, the Unity physics engine was also used to prevent generated objects from being generated in front of each other, as this would lead to the data fields of an object being written to the *.txt* file, while the container is invisible in the image.

The container used in this project is a 3D model on the *.obj* format. The 3D model is of the same dimensions as a twenty foot equivalent unit (TEU), which according to Netherlands (2021) is "a unit based on an ISO container of 20 foot length (6.10 m) to provide a standardised measure of containers of various capacities and for describing the capacity of container ships or terminals." The model was downloaded from the Unity Asset Store which is a library of models, templates, tools etc that is used for development in Unity (Store 2021). The container is defined in Unity with a length of 6.10 meters, and a width and height of 2.439 meters. An image of the container, with the exact dimensions can be viewed in Figure 3.3.

The images from Unity were obtained using U3DC (2017) to capture image variants that is relevant in training the network. Minor changes were made to the code in the retrieval of the images, such as disabling the depth information and optical flow data generation to increase code efficiency. The images obtained are RGB images, saved on the *.png* format. All images have the dimensions 600x600, meaning 600 pixels in both height and width direction. The two image variants stored in the dataset are the original image and the corresponding image mask. The initial code used to generate the synthetic data, was obtained using Stratospark (2019).

Original image

Each image contains one container located in the scene using screen coordinates of the camera. The container is positioned in the interval x = [0.4 , 0.6] and y = [0.4 , 0.6] where the value (x,y) corresponds to screen coordinates where (0,0) is bottom-left corner of the screen and (1,1)

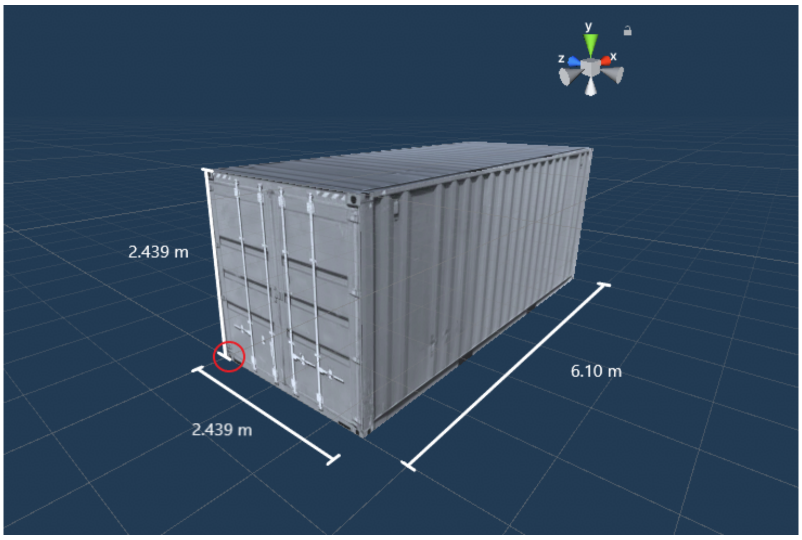


Figure 3.3: Container model used in Unity project

corresponds to top-right corner of the screen (Unity 2021b). The containers are located in the interval z = [25 , 60], where z describes the distance between container and camera and is given meters. The origin of the container, which is the pivot point that the container is rotating around, is located in the front lower left corner of the container, as shown with a red marker on Figure 3.3. Every fifth image the camera location and rotation is updated. The images are generated in a marine environment, in various weather and light conditions. An example of the resulting images can be seen in sub figures a-d in Figure 3.4.

Image mask

Each object in Unity has its own instance ID that is unique for all objects located in the scene. The image mask is generated by coloring each container object in a scene to a color based on its instance ID, while all other objects are colored black. This results in an instance segmentation where only container objects are visible. An example of the resulting image masks can be seen in sub figures e-h in Figure 3.4.

Text file

Each set of images has a corresponding *.txt* file with data of the objects in the Unity scene. The file contains positional and rotational information of the camera capturing the image and the container in the image. The position of the container are provided in two coordinate systems, world coordinates and screen coordinates. The world coordinate system is defined by a fixed origin in the Unity scene, while screen coordinates is defined as described in Section 3.1.2. All units in world coordinates are in meters. The rotational data of the container and camera is in world coordinates and is provided as quaternions. Positional data of keypoints on the container is provided as a JSON object where the key annotation consists of three letters describing which keypoint belongs to the corresponding data.

Cropped dataset

As mentioned in Section 1.3.2, the pipeline conducts two separate tasks, instance segmentation and vector field prediction. After the first task, the images are cropped and provided for vector field estimations. As the keypoint coordinates are screen coordinates on the format previously described, a different set of keypoints are needed for the full sized version of the image and the cropped version. Cropping the images and recalculating the screen coordinates of the keypoints for the ground truth images and vector fields were redundant computations that was repeated at each epoch if not handled. Therefore, a cropped version of the dataset was fabricated, estimated to significantly reduce training time. The cropped dataset version contains two *.npy* files containing the updated keypoints and vector field for a specific image instead of the original *.txt* file.

For in depth description of the data fields provided, and how to extract further data fields using the scripts provided, see appendix.

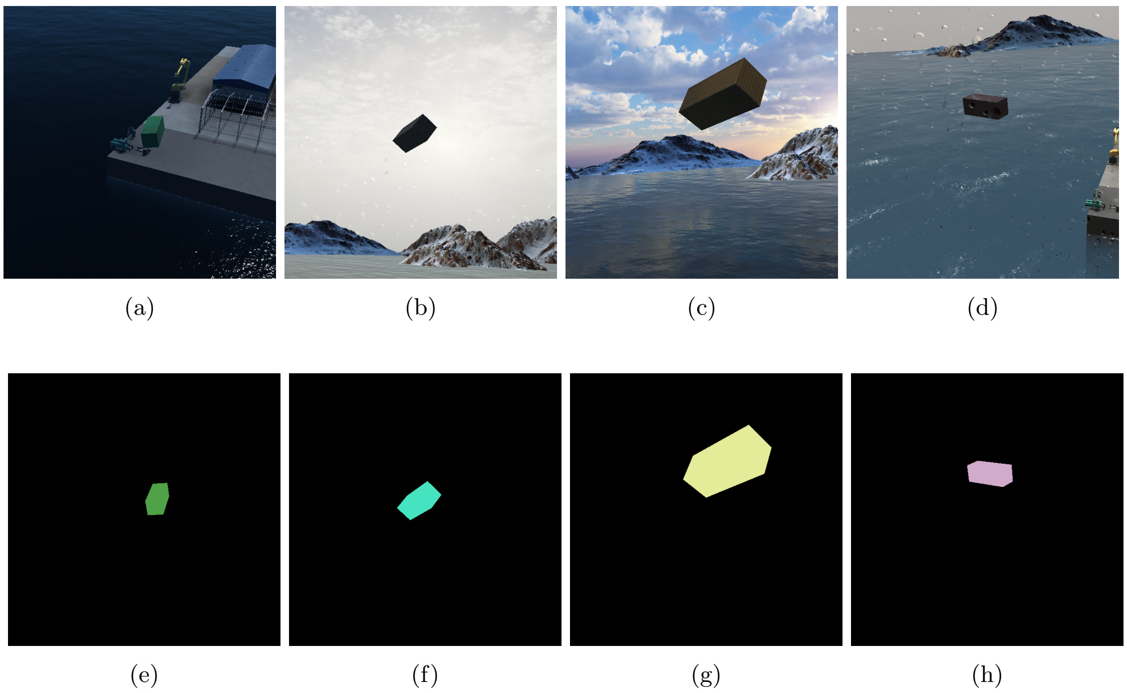


Figure 3.4: Samples of images and the corresponding image mask from the SINTEF 6DPE ISO Container Dataset

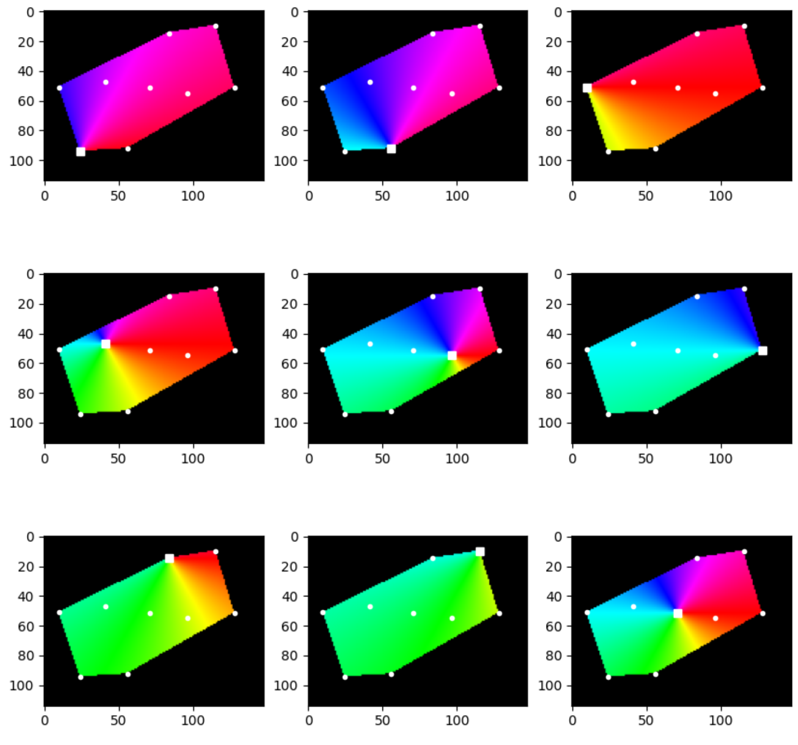
3.2 Semantic Image Segmentation

Semantic image segmentation, or *pixel-wise classification*, focuses on allocating a categorical label to every pixel enclosed by the target object(s) in an image (Long et al. 2015). It has quickly become one of the most used techniques for detection due to its ability to provide the right visual perception to machines. Semantic image segmentation has been used in various applications, ranging from medical applications (L. Liu et al. 2020), Driver Assistance systems (ADAS) ((Menze and Geiger 2015);(Cordts et al. 2016)), to handwriting recognition (Jo et al. 2020). Networks employed for semantic segmentation are usually structured around encoder-decoder architecture (Khan et al. 2020). One of the most popular architectures, as stated by Gómez-Flores and Coelho de Albuquerque Pereira (2020), which completed a comparative study of the accuracy and speed of different segmentation networks, is U-Net. U-Net is a classic, fast, efficient and precise segmentation network (Zhao et al. 2019). It was first introduced by (Ronneberger et al. 2015) for use on biomedical image segmentation, but has since been used for segmentation on all sorts of data. This is because of the fact that it does not demand a massive dataset to perform well. Based on these characteristics, U-Net was selected as the segmentation network for the proposed pipeline.

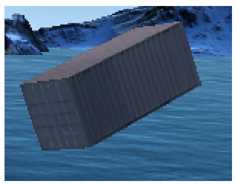
3.3 Keypoint Vector Field

This section describes the main method of utilizing the synthetic data presented in Section 5.3, into a method estimating the 6D pose of container objects. To handle challenges occurring when estimating object 6D pose, and the time requirements presented in Section 1.2, a keypoint-based method was chosen. Central in these method is creating 2D to 3D point correspondence, using a generated unit vector field to predict 2D projection location of known object 3D points. The method is largely based on the research conducted in Peng et al. (2018) and Yu et al. (2020), while also having similarities as Xiang et al. (2017) as introduced in Section 1.3. To estimate 2D projection of known 3D points, a voting based method named Random Sample Consensus (RANSAC) is utilized. The pose is estimated by solving the Perspective-n-Point problem using OpenCVs solvePnP() method using the EPnP method described in Section 2.5.2.

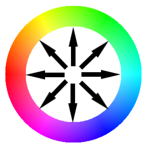
First presented is the unit vector field, followed by a presentation of the RANSAC variant used in this project and the perspective-n-point implementation.



(a) Ground truth vector field



(b) Ground truth image



(c) Vector direction color spectrum

Figure 3.5: Vector field visualization and ground truth image.

3.3.1 Unit Vector Field

Given a RGB-image, a unit vector field representation consist of pixel-wise unit vectors pointing in the direction of the 2D projection of the known 3D point. This results in a vector field, where each pixel has the components of a unit vector pointing towards each keypoint.

For each pixel $\mathbf{p} \in M$ where M is the image mask defined as: $M \subset I$ and I is the image. In Figure 3.4e, M corresponds to all pixels colored green. Each pixel $\mathbf{p}$ stores a x and y component of a unit vector, for each keypoint k, where the predicted unit vector $\hat{\mathbf{v}}_{\mathbf{k}}(\mathbf{p})$ is defined as:

$$\hat{\mathbf{v}}_{\mathbf{k}}(\mathbf{p}) = \frac{\mathbf{p}_k - \mathbf{p}}{|\mathbf{p}_k - \mathbf{p}|} = \begin{bmatrix} \hat{\mathbf{v}}_{k,x} \\ \hat{\mathbf{v}}_{k,y} \end{bmatrix}$$

Where $\mathbf{p}_k$ is the pixel coordinate value of keypoint k. Where k = [0,1,...,8] corresponds to the eight object corners and the mass center of the container. In Figure 3.5 all vector fields calculated for an image is displayed. The figure consists nine images, one for each keypoint, where the keypoint whose vector field is displayed is marked as a white square. As expected when the keypoints located on the outer parts of the figure most vectors will point in more similar directions resulting in a figure consisting of more like, and fewer colors. The opposite can be seen when the keypoints are located inside the figure. The ground truth image used to calculate the vector field is seen in Figure 3.5b, and the spectrum of colors for each unit vector is displayed in Figure 3.5c.

3.3.2 RANSAC

In Section 2.6, the general theory for Random Sample Conscensus (RANSAC) was presented and the algorithm presented is the step wise process of the basic version of RANSAC. In this project, RANSAC is applied as a method to eliminate outliers in the vector field generated by DVFnet.

Unit vectors are generated, for each pixel in image mask M. Two randomly sampled pixels $\mathbf{p}^1$ and $\mathbf{p}^2$ are obtained from the image mask M. For each keypoint k, and randomly sampled pixel $\mathbf{p}^i$ where i = 1,2. The slope m^i is calculated as:

$$m^i = \frac{\Delta Y^i}{\Delta X^i} = \frac{\hat{\mathbf{v}}^i_{k,y}}{\hat{\mathbf{v}}^i_{k,x}}$$

Where $\hat{\mathbf{v}}^i_{k,y}$ is the y-component of the estimated unity vector from pixel $\mathbf{p}^i$ pointing towards keypoint k. Using x_i and y_i which is the x and y coordinates of pixel $\mathbf{p}^i$, the x_k and y_k coordinates of the intersection of the two lines directed by the unit vectors $\hat{\mathbf{v}}^1_k$ and $\hat{\mathbf{v}}^2_k$. The pixel $\mathbf{p}^k$ with the pixel coordinates (x_k, y_k) is used as a hypothesis, and is initialized with a weight $w_k = 0$. For a set number n of the remaining pixels in the image mask M, randomly select pixel $\mathbf{p}^j$ from M and repeat the following process n times:

The unit vector from pixel $\mathbf{p}^j$ towards pixel $\mathbf{p}^k$ with pixel coordinates (x_k, y_k) is calculated using:

$$\mathbf{v}^j_k = \begin{bmatrix} \frac{x_k - x_j}{m_j} \\ \frac{y_k - y_j}{m_j} \end{bmatrix} = \begin{bmatrix} \mathbf{v}^j_{k,x} \\ \mathbf{v}^j_{k,y} \end{bmatrix} \qquad \text{where } m_j = \sqrt{(x_k - x_j)^2 + (y_k - y_j)^2}$$

The calculated unit vector $\mathbf{v}^j_k$ is now compared to the estimated unit vector $\hat{\mathbf{v}}^j_k$. This is done using the dot product of the two vectors, where $\mathbf{v}^j_k \cdot \hat{\mathbf{v}}^j_k = |\mathbf{v}^j_k||\hat{\mathbf{v}}^j_k| \cos\alpha = \cos\alpha$ since $\mathbf{v}^j_k$ and $\hat{\mathbf{v}}^j_k$ are unit vectors, to compare the angle α between the two vectors. A threshold value is chosen, where for each time a pixel $\mathbf{p}^j$ has an estimated unit vector, where the dot product of the estimated and calculated unit vector is above the set threshold, the weight w_k increments by 1. This process is repeated for all keypoints k. A set number of hypotheses are generated, and the number of hypotheses generated is equivalent to the variable N presented in Section 2.6. The process is visualized in Figure 3.6.

Resulting from these steps is a number of weighted hypotheses to where each keypoint k is located. The weighted average value of all coordinates obtained in the previous steps are calculated and returned as the final estimate of the pixel coordinates $\mathbf{p}_k = (x_k, y_k)$.

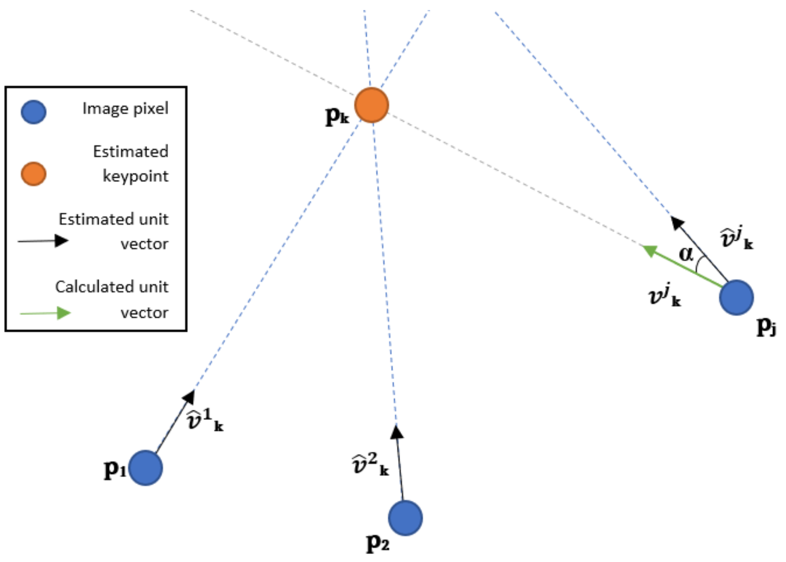


Figure 3.6: RANSAC visualization

3.3.3 Perspective-n-point

OpenCVs solvePnP() function estimates the object pose given a set of local object points in 3D and their corresponding image projections and Camera intrinsic matrix (OpenCV 2022). Eight local object coordinates are provided into the function, one for each container corner.

The camera intrinsic matrix includes the following parameters f_x, f_y, which are expressed in pixel units, and o_x, o_y, which is the pixel coordinates of the principal point. A focal length of $f = 50$ mm; and a sensor size 36 mm: x 24 mm was used. With an image size of 600x600 pixels;. This result in the parameters $m_x = \frac{600}{36}\frac{pixels}{mm} \approx 16.67\frac{pixels}{mm}$ and $m_y = \frac{600}{24}\frac{pixels}{mm} = 25\frac{pixels}{mm}$. Combining this information gives us the following definition of f_x and f_y. $f_x = m_x f$= $16.67\frac{pixels}{mm}$·50 mm;= 833.5 pixels. $f_y = m_y f$= $25\frac{pixels}{mm}$·50 mm;= 1250.0 pixels.

The parameters o_x and o_y are located in the middle of the image plane, similarly as visualized in Figure 2.14 (Unity 2021c). Therefore, o_x=o_x=300 pixels; The resulting camera intrinsic matrix used in this project is:

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 833.5 & 0 & 300 \\ 0 & 1250 & 300 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

OpenCVs solvePnP() can use different methods to solve the PnP problem. In this project the EPnP method based the work conducted in Lepetit et al. (2008), and described in Section 2.5.2. This methods was used as achieving a computationally fast network was deemed important to implement real-time pose estimation as described in Section 1.2.

The estimated pose returned is the rotation matrix and translation vector that allows for a transformation of 3D points expressed in object coordinate frame to the camera coordinate frame, which is equivalent to estimating the extrinsic matrix $\mathbf{M}_{ext}$ in Equation 2.13.

3.4 Hardware

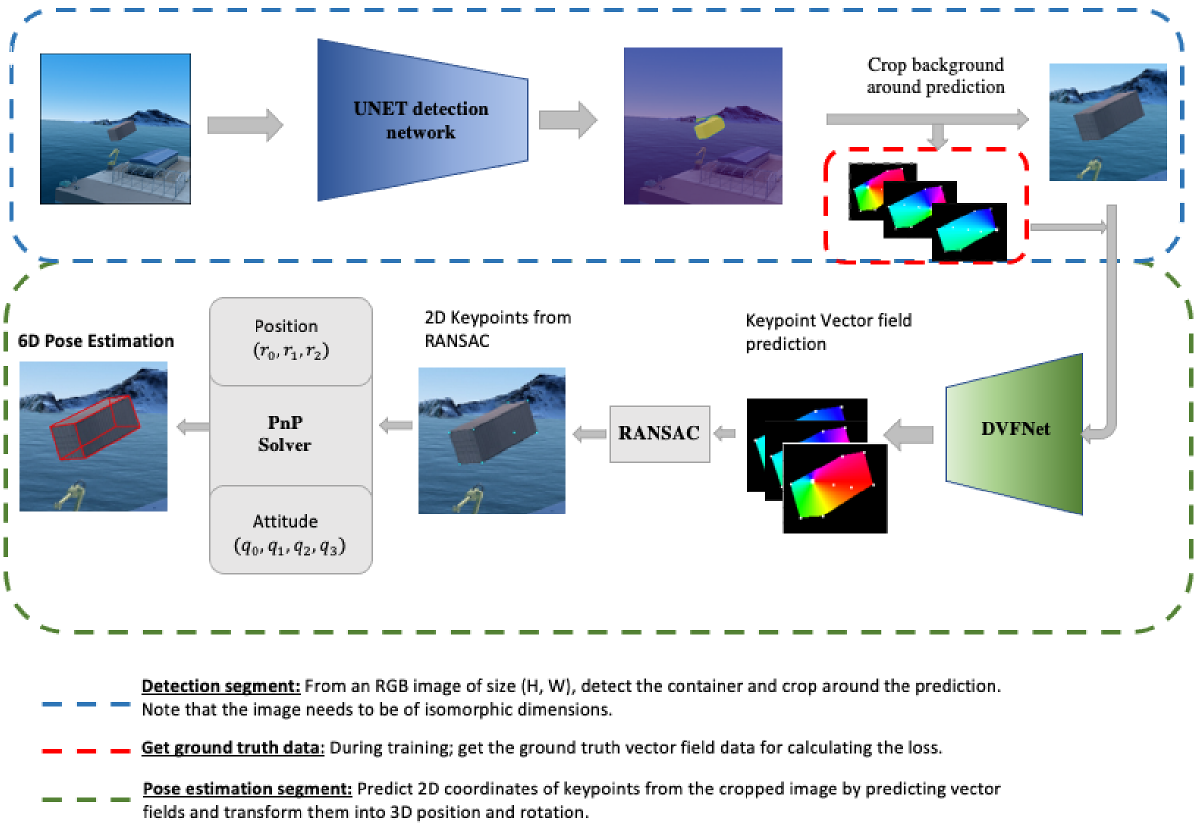
The models in this thesis were trained and tested with the computational environment provided by Google Colab and NTNU Idun. Development and testing of the models and algorithms implemented in this thesis was done using Google Colab, because of its availability and easy access to GPUs. NTNU Idun was used as the main environment for training the models as they offer access to multiple GPUs, of type V100 and P100, and distributed data parallelization (Själander et al. 2019).

Chapter 4

Pipeline

This chapter brings forth, based on the methods chosen in chapter 3, the proposed pipeline for monocular 6D pose estimation of shipping containers in offshore environment. First the entire structure of the pipeline is presented, with all of its components. Then, the two networks chosen for the pipeline are covered in detail.

4.1 Structure


UNET detection network
Crop background around prediction
6D Pose Estimation
Position (r_0, r_1, r_2)
PnP Solver
Attitude (q_0, q_1, q_2, q_3)
2D Keypoints from RANSAC
RANSAC
Keypoint Vector field prediction
DVFNet

Detection segment: From an RGB image of size (H, W), detect the container and crop around the prediction. Note that the image needs to be of isomorphic dimensions.

Get ground truth data: During training; get the ground truth vector field data for calculating the loss.

Pose estimation segment: Predict 2D coordinates of keypoints from the cropped image by predicting vector fields and transform them into 3D position and rotation.

Figure 4.1: The proposed pipeline

Many computer vision applications require solving multiple tasks in real-time. A neural network can be trained to solve multiple tasks simultaneously using multi-task learning. This can save computation at inference time as only a single network needs to be evaluated. Unfortunately, this often leads to inferior overall performance as task objectives can compete (Standley et al. 2020). Further described in Y. Wang et al. (2021) multitask learning is not always beneficial as the performance is likely to be harmed by the negative transfer (negative knowledge transfer across tasks). As such, for this pipeline an approach was chosen consisting of two separate neural networks. Hence, the following section presents the pipeline utilized in this thesis.

The proposed pipeline is composed of two segments: the detection segment and the pose estimation segment, as depicted in Figure 4.1. The detection segment can be further broken down into two main parts: the detection of the object of interest and the cropping of the image around the object. First, an established number, the batch size, of images are randomly selected with their corresponding masks from the training set. Then the images are fed to the U-Net network for semantic segmentation of the shipping container. After predicting the location of the container, the original image is cropped around the prediction. This is to ensure that the noise of the background is reduced and any unnecessary information for the pose estimation model are shaved away. Subsequently, by cropping the image we are also decreasing the time complexity of the pose estimation network by reducing the dimensionality of the input. This in turn is favorable as speed is vital to reach the end goal of real-time pose estimation. The first segment is summarized in **Algorithm 5** below.

Algorithm 5 Monocular Container segmentation

```
Require: RGB-image, $I_{2D}$, of isomorphic dimensions and corresponding gt mask $M_{2D}$.
Ensure: Cropped image $I_{C2D}$
  1. $dim(I_{2D}) == dim(M_{2D})$        ▷ Check dimensions of image and mask are the same.
  2. Normalize $I_{2D}$ and assert unique values $M_{2D} \leftarrow (0., 1.)$
  3. Input $I_{2D}$ into U-Net and output the predicted mask $\hat{y}_{2D}$;
  4. Quadratic crop of $I_{2D}$ where $\hat{y}_{2D} \geq 0.6$
```

The next segment, the pose estimation segment, is built up by three parts. The first part, marked in red in figure 4.1, takes in the cropped image from the detection segment and calculates a ground truth unit vector field for each keypoint. In the second part the cropped image is sent to the DVFnet network for unit vector field prediction. After the cropped images has been processed by DVFnet and a unit vector field prediction for each keypoint has been output, the predictions are sent to the RANSAC algorithm for keypoint localization. The RANSAC algorithm, as described in Section 3.3.2, takes in the predicted unit vector fields and estimates the location of a keypoint. Through this process, 9 two dimensional keypoints hypothesis for the container are output. Finally, the keypoints are fed into the PnP solver to obtain the estimated attitude and position of the shipping container. **Algorithm 6** below describes the process of the pose estimation segment.

Algorithm 6 Pose Estimation of container

```
Require: Cropped RGB-image; $I_{C2D}$, corresponding keypoints; $k \rightarrow [\mathrm{k}_1, k_2, ...k_9]$
Ensure: $(q_0, q_1, q_2, q_3)$ and $(r_0, r_1, r_2)$
  1. $k \leftarrow$ updatedKeypoints($I_{C2D}$, $k_0$)        ▷ scale keypoints to cropped dimensions
  2. $y \leftarrow$ getvector field($I_{C2D}$)        ▷ get gt-data for vector field
  3. Input $I_{C2D}$ into DVFNet and output vector field predictions $\hat{y}_{vf}$
  4. Apply the RANSAC algorithm on $\hat{y}_{vf}$ for 2D keypoint regression; $\mathbf{p}_k \leftarrow [\mathbf{p}_{k_1}, \mathbf{p}_{k_2}, ..., \mathbf{p}_{k_9}]$
  6. Send $\mathbf{p}_k$ to PnP solver to get the estimated attitude and position.
```

4.2 The networks

As seen in figure 4.1, the pipeline relies on two deep neural networks to estimate the 6 degrees of freedom of the shipping container. The first network is the selected U-Net and the second is a customized network called DVFnet.

4.2.1 U-Net

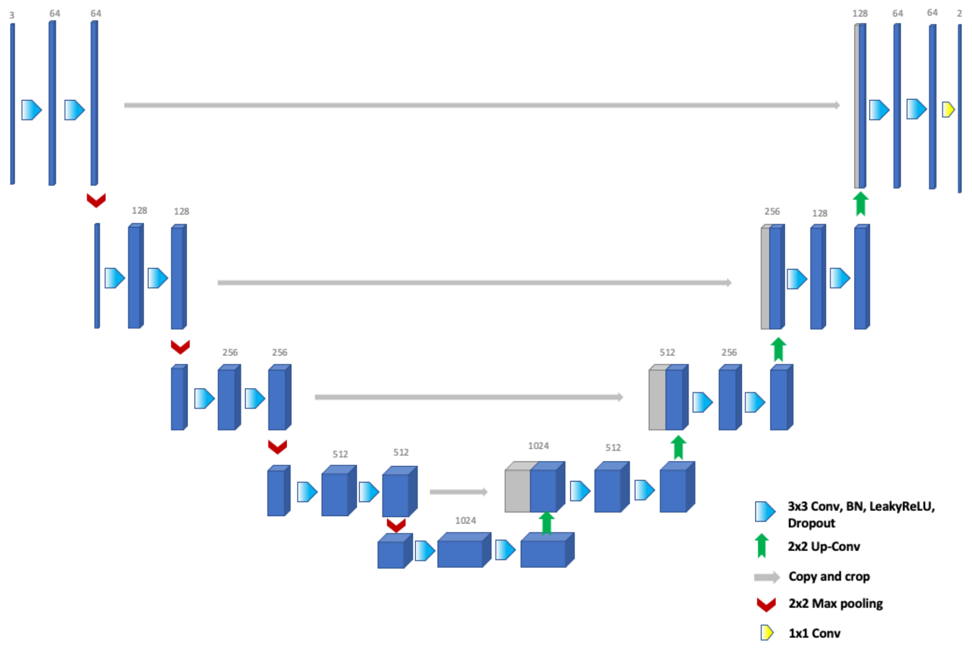


Figure 4.2: Illustration of the architecture of U-Net, with number of channels for each layer.

The architecture of the U-Net, as shown in Figure 4.2 above, is structured around three main parts; the contracting part, the bottleneck and the expansive part. The contracting part mimics the classical convolutional network in its structure, as it uses repeated application of two convolutions of size 3x3, padding of 1 and a stride of 1. Each convolution is then followed by Batch Normalization, the activation function LeakyRELU and a droput layer, with a activation probability of 10%. After dropout, a 2x2 max pooling operation with a stride of two is applied for downsampling the generated feature map. After every pooling operation the number of feature channels are doubled.

The bottleneck, located at the bottom of the contracting part and the beginning of the expansive part, consists of two 3x3 convolutions with no downsampling. Its purpose is to secure vital feature information at the highest number of feature channels without further obscuring the data with downsampling.

The expansive part is built around the operation of transposed convolution ("Up-convolution"), which is an operation that upsamples the input to generate an output feature map with a spatial dimension greater than the input feature map, while maintaining the connectivity pattern of convolution.

Every step of this part upsamples the feature map originating from the bottleneck by a 2x2 transposed convolution that also halves the number of feature channels. After the transposed convolution the feature map is concatenated with the corresponding feature map from the contracting part, showed in **??** as the grey arrows between the contracting and expansive part. Following the concatenation are two 3x3 convolutions, each succeeded by Batch Normalization, LeakyRELU and dropout layer. The final layer of the expansive part is a 1x1 convolution that maps the 64 channels to the predetermined number of classes, which in our case is one: the shipping container. Lastly the output is sent to a sigmoid activation function to normalize the logits given by the network into probabilities with the value for the shipping container.

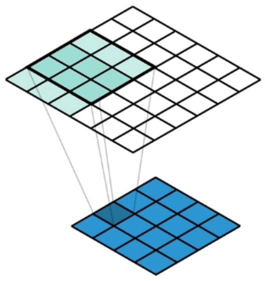


Figure 4.3: Illustration of transposed convolution; from a 4x4 feature map (blue) to a 6x6 feature map (green) (Lane 2018)

The two loss functions used with the U-Net network during training were binary cross entropy loss and Dice Loss.

Binary Cross Entropy

Binary Cross Entropy (BCE) compares the predicted probabilities to the ground truth mask with values $\in [0,1]$. The score is calculated in each pixel by calculating the distance between the ground truth value and the predicted value. By definition, BCE is the negative average of the log of corrected predicted possibilities and is defined as:

$$BCE \rightarrow L_{BCE}(y,\hat{y}) = -(ylog(\hat{y}) + (1-y)log(1-\hat{y})) \tag{4.1}$$

where y is the ground truth and $\hat{y}$ is the prediction.

The implementation of the BCE loss function used in this thesis is taken from the Pytorch library, there known as BCEWithLogitLoss (PyTorch 2022a).

Dice Loss

Dice Loss is a loss function based on the *Dice Coefficient*, which is a popular accuracy metric in computer vision used in various detection and segmentation tasks. It calculates the similarity between two inputs, commonly images. The Dice Loss is calculated using the following equation:

$$DiceLoss \rightarrow L_{DC}(y,\hat{y}) = 1 - \frac{2(y\hat{y}) + \epsilon}{y + \hat{y} + \epsilon} \tag{4.2}$$

where y is the ground truth, $\hat{y}$ is the prediction and $\epsilon \in [0 > \epsilon \leq 1]$ is added in both the numerator and denominator to guarantee the loss is not undefined in edge case scenarios, mainly when $y = \hat{y} = 0$. A code snippet of the Dice Loss class programmed for this thesis is displayed below.

```python
import torch

class DiceLoss(torch.nn.Module):
    def __init__(self):
        super(DiceLoss, self).__init__()

    def forward(self, inputs, targets, epsilon=0.0001):

        #sigmoid activation layer --> Remove if Sigmoid is applied in the network
        inputs = torch.sigmoid(inputs)

        # flatten the target and prediction tensors
        inputs, targets = inputs.view(-1), targets.view(-1)

        #Compute intersection between target and prediction
        intersection = (inputs * targets).sum()
        #Calculate the dice score between target and prediction
        dice = (2.*intersection + epsilon) / \
            (inputs.sum() + targets.sum() + epsilon) #Adding a smoothing variable
    to safeguard from zero division
        #subtract the dice score from 1 to find the loss and return
        return 1 - dice
```
Listing 4.1: Python Dice Loss implementation

The source code of the U-Net network can be found in the appendix.

4.2.2 DVFnet

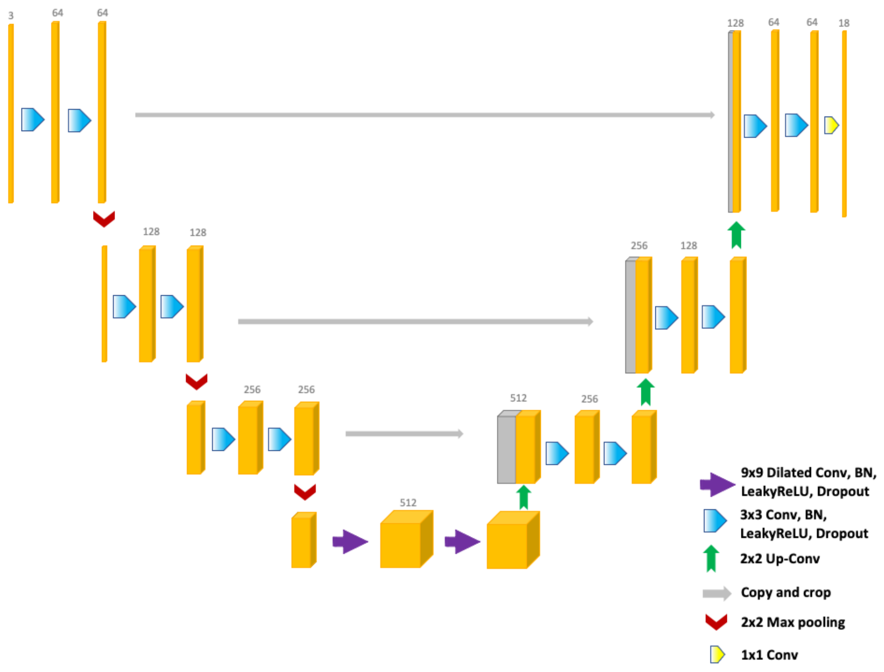
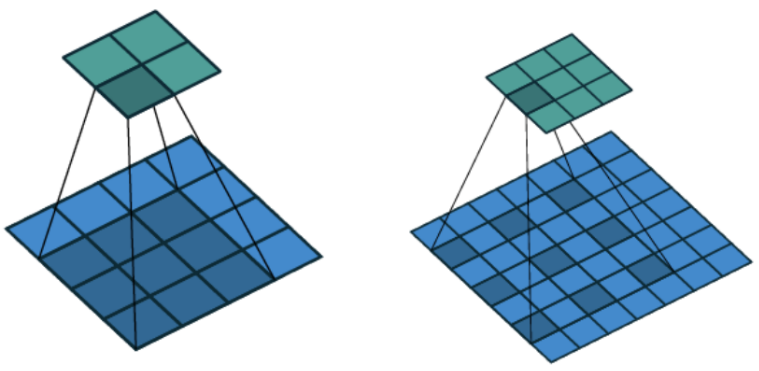


Figure 4.4: Illustration of the architecture of DVFnet with number of channels for each layer.

DVFnet, or *Deep Vector Forecasting Network*, is a network based on the architecture of U-Net, due to its capabilities describe in the previous section. DVFnet is a more shallow network than U-Net, meaning fewer channels/connection between layers, and has half as many convolution operations. The network also utilizes dilated convolution in the bottleneck part of the architecture to compensate for fewer channels. Dilated convolution is a operation that expands the kernel by applying gaps between the consecutive elements. This implies that the kernel skips over pixels in the gap and as such cover a larger area of the input. The expanded kernel is defined by a dilation factor, l, which determine to what extent the kernel is expanded. Based on the value of l, a total of $(l-1)$ pixels are skipped in the kernel. The objective of dilated convolution is to cover more information from the output obtained with every convolution operation. The advantages given by dilated convolutions are a wider kernel field of view at the same computational cost, and less need for pooling, do to the kernel expansion over the area of the input. An illustration of dilated convolution is given in figure 4.5 next to conventional convolution.



(a) $l = 1$ (b) $l = 2$

Figure 4.5: Illustration of convolution (a) and dilated convolution (b) (Z. Zhang et al. 2019)

These alterations described above were made to increase the speed of the network while simultaneously keeping as much of its detection capabilities, minimizing the compromise between speed and accuracy.

The architecture of DVFnet is structured around three parts, like U-Net, and is illustrated in Figure 4.4. The first part is almost identical to the classical U-Net above with repeated applications of 3x3 convolutions of stride and padding of 1. This is followed by Batch Normalization, the activation function LeakyReLU and a dropout layer with a activation probability of 20%. DVFNet has a higher dropout probability than U-Net due to fewer convolutional layers, implicating fewer dropout possibilities. After dropout, a 2x2 max pooling operation with a stride of two is applied for downsampling the generated feature map. Similar to U-Net, after every pooling operation the number features channels are doubled. The bottleneck of DVFnet is composed of two sequential dilated convolutions of size 9x9, with a padding and stride of 1, and no downsampling. After the bottleneck, the feature map output from the dilated convolutions are up-sampled by a 2x2 transposed convolution that halves the number of feature channels. After the application of transposed convolution the feature map is concatenated with the corresponding feature map from the contracting part. Following the concatenation is a 3x3 convolution, succeeded by Batch Normalization, LeakyRELU and dropout layer, with a activation probability of 20%. The final layer of the expansive part is a 1x1 convolution that maps the 64 channels to the predetermined number of channels, which for the vector field prediction is 18; a vector field for each of the 9 keypoints, represented by the keypoints coordinates.

The source code of the DVFnet network can be found in the appendix.

The two loss functions used with the DVFnet during training were Huber loss and a custom Huber Loss

Huber Loss

The Huber Loss function creates a criterion that uses a squared term if the absolute element-wise error falls below a values δ and a scaled absolute element-wise error otherwise. The absolute element wise error is calculated as $|x_n - y_n|$ where x_n is the predicted value and y_n is the target value. (PyTorch 2022b). In this thesis the value of $\delta = 0.5$ is selected. As Huber Loss is loss function with similarities to SmoothL1Loss (PyTorch 2022c), a loss function not used in this project, but that was utilized in similar projects such as Peng et al. (2018). A δ of 0.5 makes the Huber Loss function similar to SmoothL1Loss, but as Huber Loss is a loss function that is less sensitive to outliers PyTorch (2022b) this function was selected.

The output of the Huber Loss function is reduced to a mean value output.

For a batch of size N, the unreduced loss function can be described as:

$$l_n = L = [l_1, ..., l_N]^\top$$

with

$$l_n = \begin{cases} 0.5(x_n - y_n)^2, & \text{if } |x_n - y_n| < \delta. \\ \delta(|x_n - y_n| - 0.5\delta), & \text{otherwise.} \end{cases} \tag{4.3}$$

Custom Huber Loss

A custom Huber Loss function was implemented, to have full flexibility and control of computational calculations conducted, and investigate if a custom implementation could increase performance. The custom Huber Loss function was implemented identically to the function described above.

Chapter 5

Results

This chapter lays forth the results from the pipeline described in Chapter 4. All models were trained and tested on the SINTEF 6DPE ISO Container Dataset as introduced in Section 3.1.2. First, the results from the segmentation model are presented with a detailed table containing the hyperparameters, following this the pose estimation results for estimated vector fields are presented, in addition to altered versions of the ground truth vector field. Section 5.3 presents an overview and detailed description of the SINTEF 6DPE ISO Container Dataset.

5.1 Segmentation results

In this section, the results obtained during training and testing of U-Net is presented. The section is divided in to four subsections. First, the a detailed list covering the training process with hyper-parameters used and accuracy are introduced. Then, the results from the best model predicting on synthetic data are covered along with their respective dice score. Next, the results from the best model predicting on real data are shown. Finally, a real time segmentation test is conducted and its results are presented. Based on the accuracy shown in Table 5.1, UNETv.8 was selected as the best model and was utilized in the tests onward.

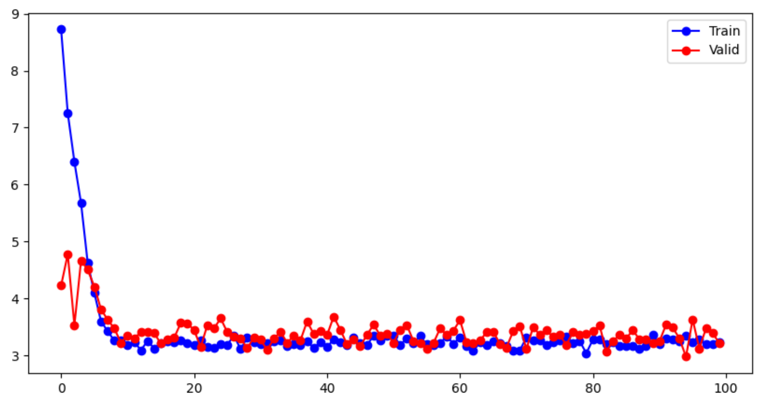


Figure 5.1: UNETv.8

Segmentation results					
Model name	Learning Rate	Optimizer	Batch Size	Epochs	Accuracy (Dice score)
UNETv.1	0.0002	Adam	25	100	0.371
UNETv.2	0.01	Adam	50	100	0.173
UNETv.3	0.0005	Adam	32	100	0.616
UNETv.4	0.0001	Adam	50	100	0.325
UNETv.5	0.03	Adam	20	70	0.178
UNETv.6	0.1	Adam	1	50	0.0257
UNETv.7	0.005	Adam	25	100	0.734
UNETv.8	0.003	Adam	32	100	0.825
UNET-SGD.1	0.0002	SGD	32	100	0.479
UNET-SGD.2	0.01	SGD	10	100	0.539
UNET-SGD.3	0.0005	SGD	32	100	0.636
UNET-SGD.4	0.0001	SGD	50	100	0.361
UNET-SGD.5	0.03	SGD	10	70	0.415
UNET-SGD.6	0.1	SGD	50	50	0.0174
UNET-SGD.7	0.005	SGD	32	100	0.674
UNET-SGD.8	0.003	SGD	32	100	0.592

Table 5.1: U-Net segmentation models

5.1.1 Prediction on synthetic data

To properly test the model it was given four images from the test data, which are not included in the training data, of different difficulties, ranging from easy to very hard. The predictions provided by UNETv.8 on synthetic data are introduced below. Along with the predictions are the cropped images given by the predictions. A good quality crop is a cropped image that contains the entire container. The predictions corresponding dice scores are also presented. The images selected for testing are from the testing dataset, in line with theory presented in the Section 2.2.3, and thus are unknown to the network.

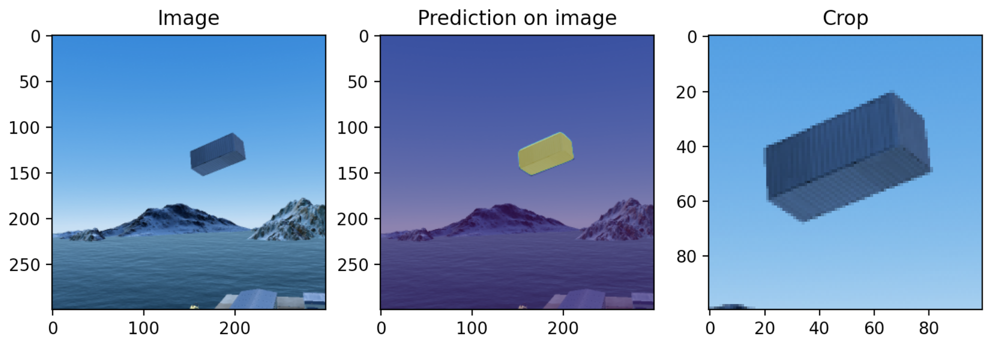


Figure 5.2: Easy prediction - Dice score: 96%

The first detection was done, Figure 5.2, on a test image deemed as an easy image. This is because the container is in plain sight with limited noise; the mountains in the background and a small portion of the dock is visible in the right bottom corner, and no occlusions. On this image the model outputted a prediction with a dice score of 96% and a corresponding crop of good quality.

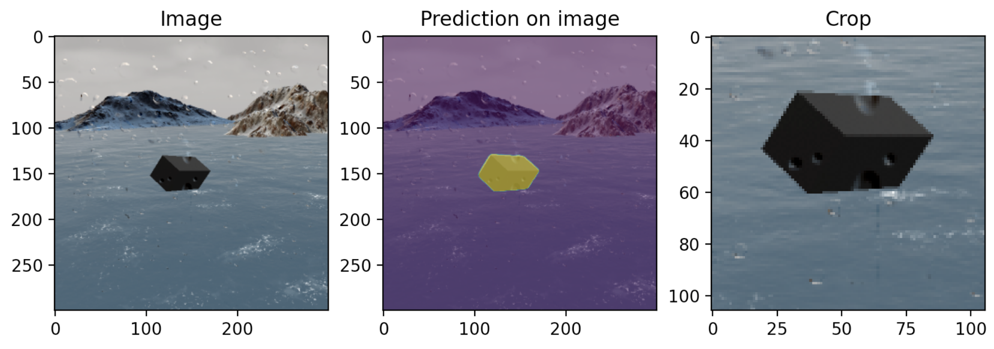


Figure 5.3: Medium prediction - Dice score: 97%

The second detection was done, Figure 5.3, on a test image of medium difficulty. The image has added noise of simulated rain drops on the camera lens to imitate difficult weather conditions, along with the mountains in the background and rough seas. Despite this the model outputted a prediction with a dice score of 97% and a corresponding crop of good quality. This demonstrates the models ability to handle disturbances related to offshore environments.

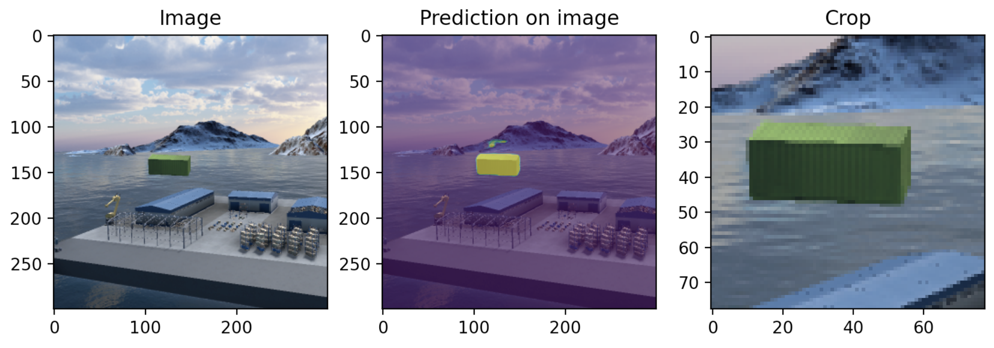


Figure 5.4: Hard prediction - Dice score: 93%

The third prediction was done, Figure 5.4, on a test image of hard difficulty. The container is located in a cluttered space with buildings of seemingly the same size because of their distance from the camera is greater than the container to the camera. The buildings also share similar patterns on the roof to further increase of difficulty. To lessen the challenge the container is colored in another color than the buildings. The noise in the image is substantially more than the previous images, with multiple boxes, buildings, sharp lines and mountains. Surprisingly, the model outputted a prediction with a dice score of 93% and a corresponding crop of good quality.

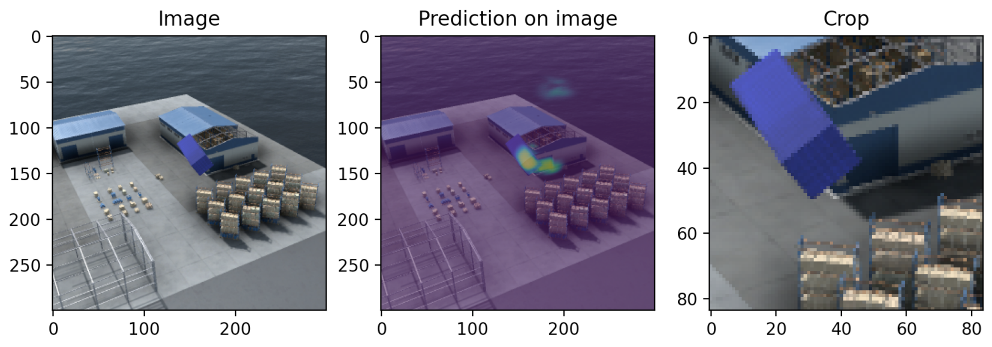


Figure 5.5: Very Hard prediction - Dice score: 47%

The forth and last prediction was done, Figure 5.5, on a test image of very hard difficulty. Although there is no occlusion, the image is heavily cluttered with objects, boxes, buildings and other noise. The container also shares its color closely with its surroundings. This hard environment manifests it self in the prediction as the dice score dropped from the 90s to 47%. Yet, the cropped image is of good quality despite the bad prediction.

5.1.2 Prediction on real data

The predictions provided by UNETv.8 on real data are introduced below. The predictions are ranked from easy to very hard, analogous with the synthetic data test. The final crop given by the prediction is also shown. Note that there were not calculated any dice score as the real images used do not have any ground truth masks.

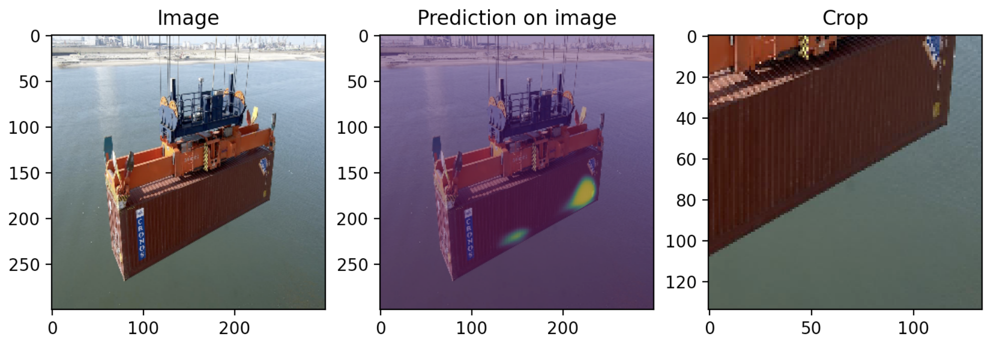


Figure 5.6: Easy prediction

The first detection was completed on, Figure 5.6 ,an image of easy difficulty. The container is clearly visible with a partial occlusion by the container spreader attached to the top of the container. The container spreader contributes to some noise as well as its shape and parts are of cuboid shapes. As shown in Figure 5.6, the model outputs a very bad prediction and consequently a bad cropped image. It gives a strong prediction in the lower right corner of the container.

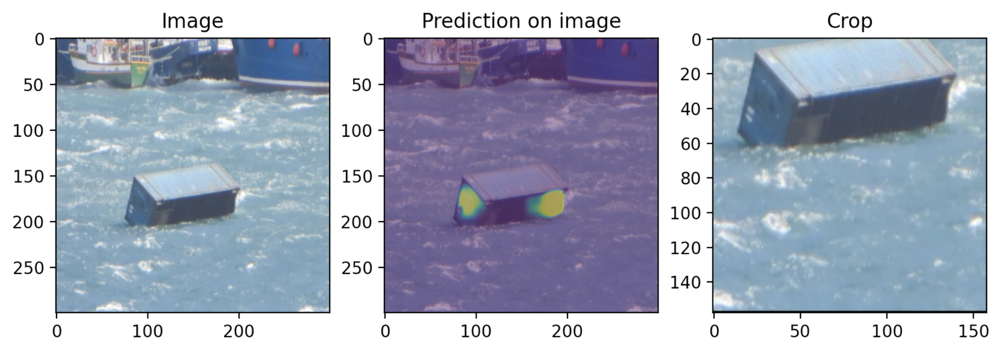


Figure 5.7: Medium prediction

The second detection was completed on, Figure 5.7, an image of medium difficulty. The container is partially occluded by water and shares a shade of blue similar to the water surrounding it. The rough water with its white crests of breaking waves and the boats in the background provide clutter and generates noise. The prediction given by UNETv.8 is of mediocre quality, which leads to a mediocre crop as the top left corner is shaved off. The prediction marks two separate areas on the container, with the strongest prediction being made on the right side of the container.

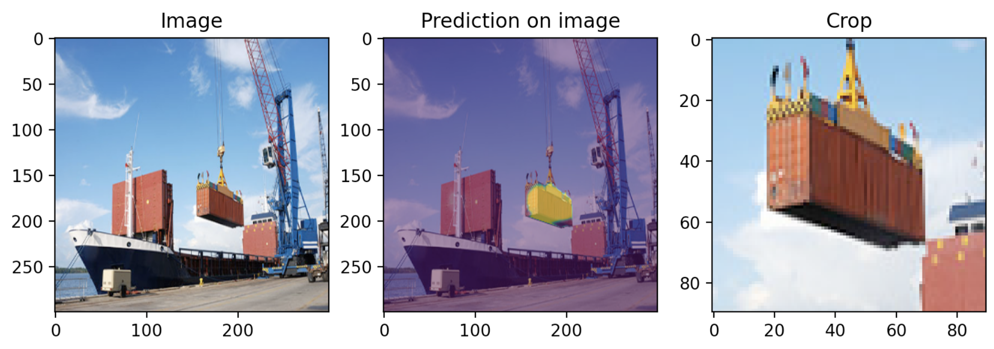


Figure 5.8: Hard prediction

The third detection was completed on, Figure 5.8, an image of hard difficulty. The container hangs from a crane above the cargo hold of a large vessel, between two large cuboid shaped hold doors. The container and the hold doors shares identical colors and as such the noise the hold doors generate poses as a though distraction for the model. The container is also partially occluded by a container spreader attached to it's roof. The prediction however is surprisingly good and as such delivers a crop of good quality.

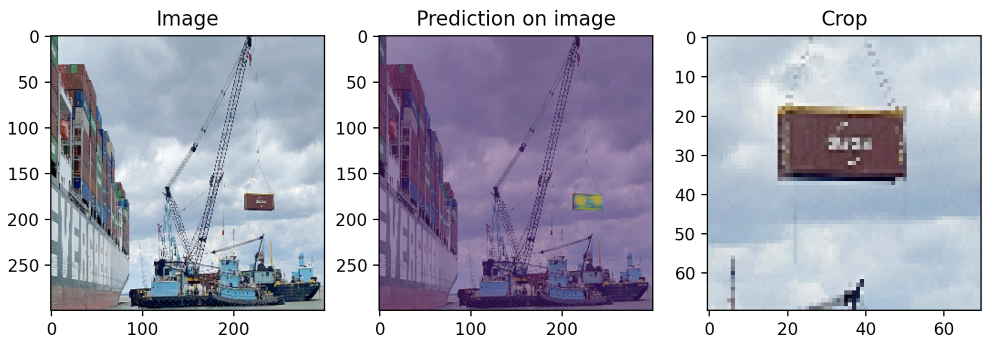


Figure 5.9: Very Hard prediction

The last detection was completed on, Figure 5.9, an image of very hard difficulty. Similar to the last prediction above, the container is hanging in the air by a crane during a crane operation. Though the container is not occluded, the image contains a lot of noise. The ships in the background, the container freight on the left side and the abundance of containers on the freight all generate much noise. Remarkably, the model is able to make a good confident prediction on the container, which leads to a crop of good quality.

5.1.3 Live segmentation test

The purposes of the live test is to test the speed of the first part of the pipeline, segmentation and cropping, to see if it has real time capabilities. The tests were conducted on 10 images in series to mimic video feed. The time are average time taken to segment the object and the average cropping time.

Live Test results - UNETv.8					
Hardware	type	Amount	Avg. speed	Avg. crop speed	Avg. total speed
i7	CPU	1	1.716 seconds	0.0417 seconds	1.7577 seconds.
V100	GPU	1	0.683 seconds	0.0248 seconds	0.7078 seconds
P100	GPU	1	0.741 seconds	0.0283 seconds	0.7693 seconds

5.2 Pose estimation results

In this section, the results obtained during training, validating and testing of DVFnet is presented. The section of divided in two subsection. First the results obtained during the vector field generation is presented, followed by the presentation of keypoint and pose estimation results yielded during the PnP and RANSAC part of the pipeline.

For the pose estimation results, different DVFnetwork models with the structure described in Section 4.2.2 was initiated and trained. Various hyperparameters were used during trainig, and validation loss, training loss and training images were stored during training and used for evaluation of the models. Table 5.2 compares the different vector field estimation models trained and evaluated during this project. As a larger number of models were trained and evaluated, some were selected for further testing with testing images, which is images previously not seen by the model. The models selected for further testing were evaluated as best performing.

Pose Estimation results

Model name	Learning Rate	Optimizer	Batch Size	Loss function	Epochs
DVFnet_v.1	0.003	Adam	6	Huber*	50
DVFnet_v.2	0.001	SDG	8	Huber*	30
DVFnet_v.3	0.0005	Adam	8	Huber	100
DVFnet.v.4	0.01	SDG	6	Huber	20
DVFnet_v.5	0.003	Adam	4	L1	80
DVFnet_v.6	0.02	Adam	4	Huber	50
DVFnet_v.7	0.001	Adam	3	Huber	80
DVFnet_v.8	0.01	SDG	6	L1	100
DVFnet_v.9	0.001	Adam	6	L1	50

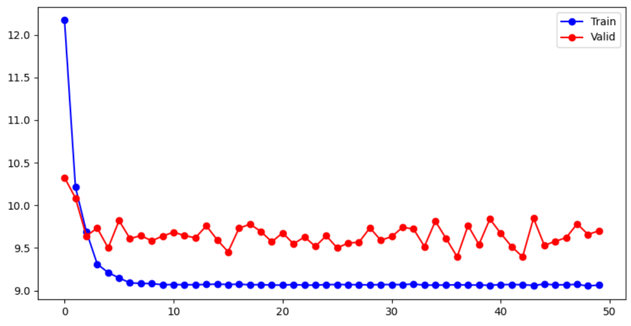
Table 5.2: DVFnet models, where * indicates custom made loss function

5.2.1 Vector field prediction results

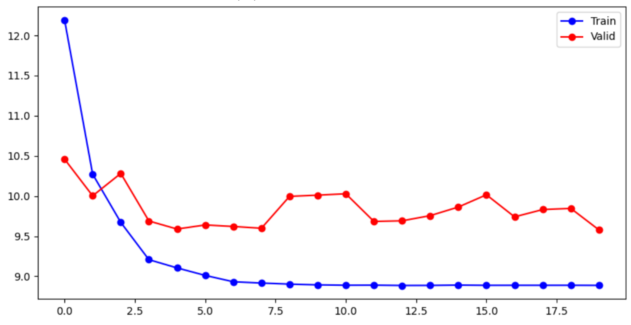
More detailed hyperparameter information of the selected models are shown in Table 5.2. All models, including the ones that were selected for further testing was trained with equal validation and training batch size. Therefore, in Table 5.2, the column *Batch size* refers to both validation batch size and training batch size. All models were trained with a learning step = 3. Three models were selected for further testing based on the average loss value, and loss value plot.

The loss function plots of the three selected model are visualized in Figure 5.10. Based on these plots, and validation images predictions during training, the model DVFnet_v.1 was evaluated as the best performing model. From the loss plots we can see that DVFnet_v.1 (Figure 5.10a), while having a training loss almost equal to DVFnet_v.2 (Figure 5.10b), the validation loss of DVFnet_v.1 is lower. Compared to DVFnet_v.3 (Figure 5.10c), DVFnet_v.1 achieves considerably lower validation loss.

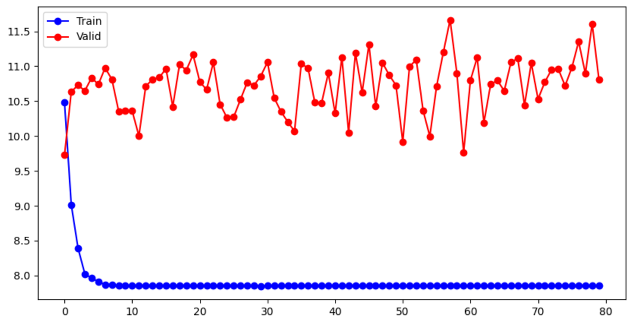
In Figure 5.12, a validation field generated from an image in validation set at epoch 69 is visualized. What is interesting in this data is the fact that the containers appears to be identified. Also worth noting is that the vector field bears resemblance to the circular spectrum of color visualized in the ground truth vector field in Figure 3.5a. In Figure 5.13a vector fields generated during testing is displayed. From these images, what can be seen is that the colorization, and also vector direction, of the area outside of the image mask does not only varies form image to image, but also within each image consists of multiple colors. In Figure 5.11, vector field generated during training is visualized. The average computational time for model DVFnet_v.1 to process one image, and return a vector field estimate using a V100 16 GB GPU, was 0.87 s.



(a) DVFnet_v.1



(b) DVFnet_v.2



(c) DVFnet_v.3

Figure 5.10: Training av Validation loss for the three models selected for further testing.

5.2.2 PnP and RANSAC results

The first phase of model testing resulted unfortunately in vector fields that were not of a level high enough to estimate pose. Therefore, this section will first presented the keypoint estimations resulting from the estimated vector fields. Following this, altered versions of the ground truth vector fields were used to estimate the use of PnP and RANSAC to obtain keypoints and pose, as originally planned in the pipeline.

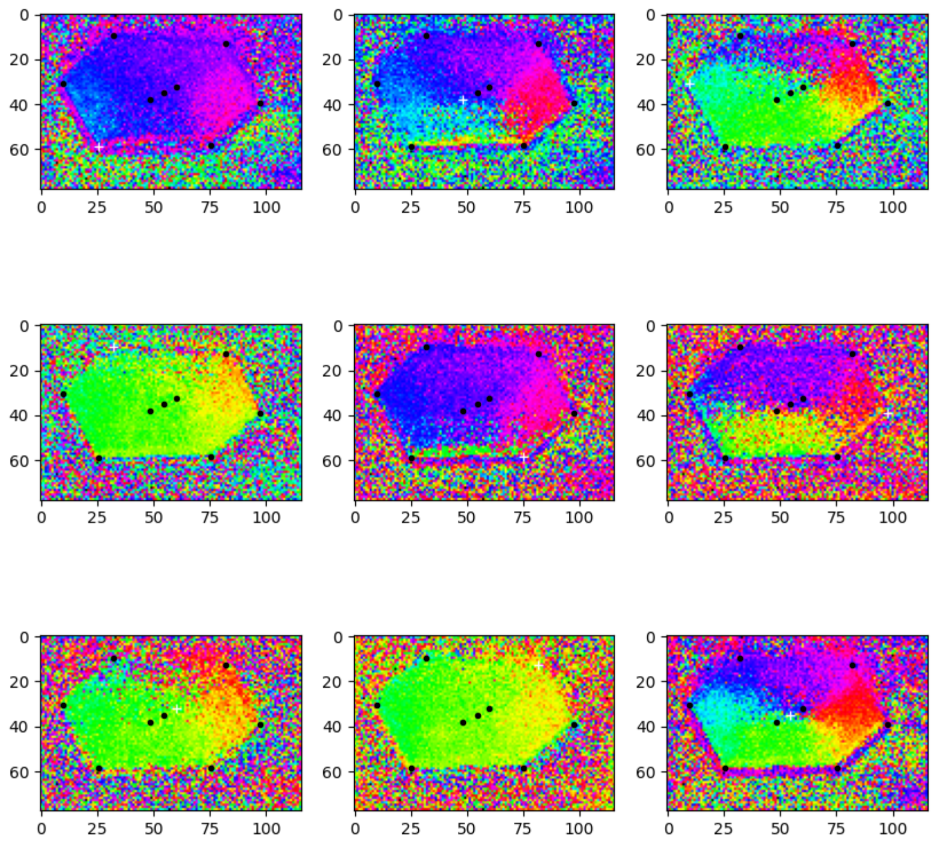


Figure 5.11: Visualization of vector field generated during training

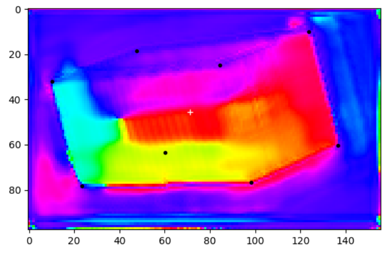
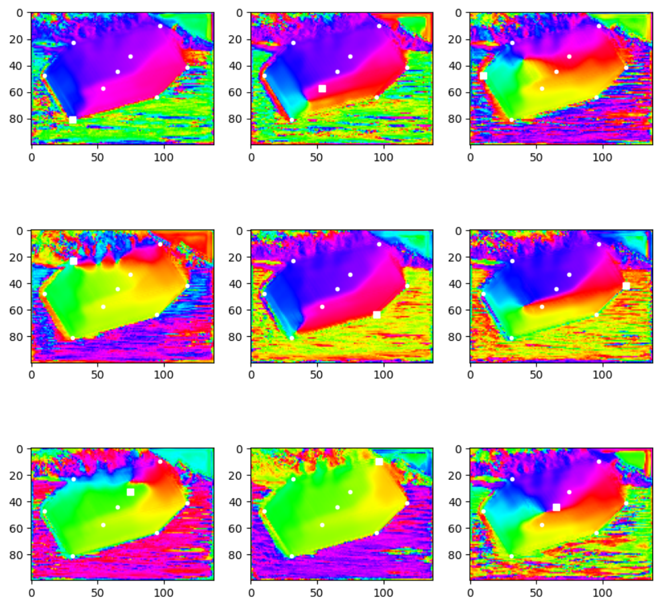


Figure 5.12: Visualization of vector field generated at during validation


(a) Prediction on validation set

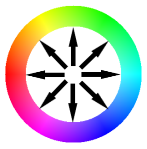

(b) Vector direction color spectrum

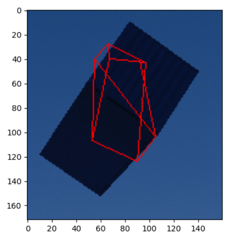
Figure 5.13: Vector field prediction generated during validation, and vector direction color spectrum.

Results using predicted vector field

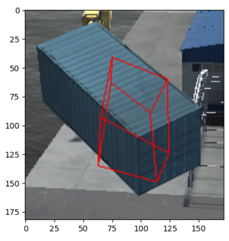
Overall the resulting pose and keypoint estimation obtained from the predicted vector field did not yield any pose. As apparent from Figure 5.14, which displays representative examples of visualized poses obtained during testing, pose can not be estimated.

Figure 5.15 displays the generated hypotheses, for a sample of each of the eight container corners, with the keypoint in focus marked with a cyan square and the hypotheses marked with green circles. Each hypotheses is displayed, independent of the keypoints weight w_k. These images are generated with a total number of hypotheses of 100, and a ransac threshold of 0.99. The number of hypotheses was chosen in order to investigate in such a high number of hypotheses would increase the pose estimation. Increasing the number of hypotheses, results in task more computationally demanding. At early stages of testing of the model, considerably lower levels of hypotheses were used, but as the vector fields did not return any reasonable pose, the number of hypotheses were increased to increase performance. At 100 hypotheses and a ransac threshold at 0.99, the total runtime of the RANSAC voting and keypoint estimation averaged at 57 seconds for a single image, running on one V100 16 GB GPU. Therefore, further testing with increasing number of hypotheses were ended.

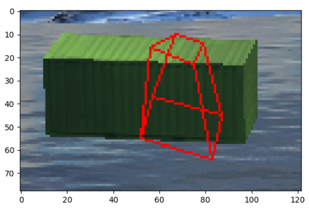
To clarify, the number of green circles on each sub figure in Figure 5.15 is not equal to 100, as hypotheses with values outside the screen dimensions were not included.



(a)

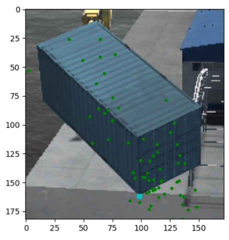


(b)

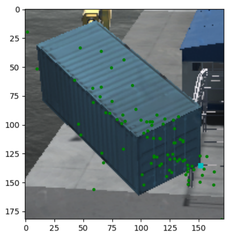


(c)

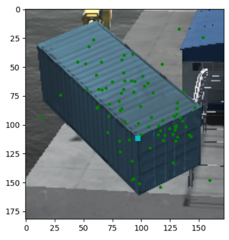
Figure 5.14: Estimated pose using predicted vector fields



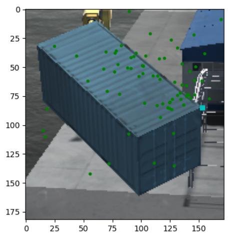
(a)



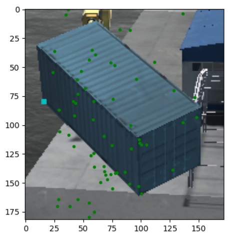
(b)



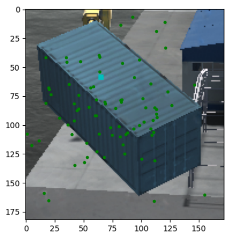
(c)



(d)

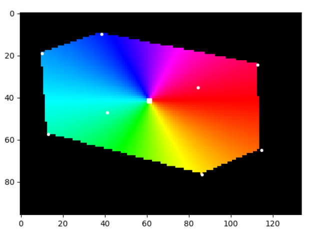


(e)

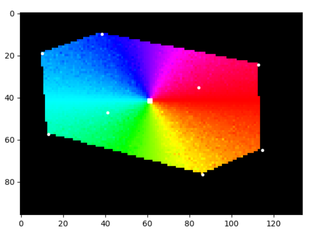


(f)

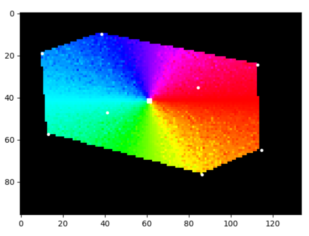
Figure 5.15: Keypoint estimate using predicted vector field



(a) α = 0



(b) α = 30%



(c) α = 50%

Figure 5.16: Ground truth and altered vector fields

Results using ground truth vector field

As pose estimation using the generated vector fields did not provide any feasible solutions, further testing with keypoint detection and post estimation using Perspective-n-Point (PnP) and RANSAC, was done using altered vector field versions, generated by adding noise to the ground truth vector field. The algorithm in 2 was used to generate noise to the vector field, and the ground truth vector field and custom vector fields are visualized in Figure 5.16.

```
Algorithm 7 Add noise to ground truth vector field
Require: vector field, $V_{2D}$, interval value $\alpha$
  1. Iterate through all coordinates in $V_{2D}$
  2. With a probability of 33.3%, change the coordinate value randomly in the interval $\pm$ $\alpha$%.
  return Altered vector field $V'_{2D}$
```

Three tests were conducted, for the following values for α: 0, 30,% and 50%. The test conducted with $\alpha = 0$ is equivalent with the test being conducted with the ground truth vector field. The test were systematically conducted with different values for the two parameters; number of hypotheses and ransac threshold values, defined as described in Section 3.3.2. The results of these tests can be seen in Table 5.3. In evaluation of the parameters, the following metrics are used:

Average time is the average time in seconds used to estimate the pose for each image, using the RANSAC voting process and solvePnP() function.

Mean 2D keypoint distance is the mean distance between the estimated keypoint and the ground truth keypoint, in number of pixels.

Mean 3D distance. Returning from the solvePnP() function is the rotation and translation vector that transform a local 3D point expressed in the object coordinate frame to the camera coordinate frame (OpenCV 2022). Therefore, to calculate this metric the 3D object coordinate point was first transformed from object coordinate space to camera coordinate frame, using the estimated rotation and translation vector. After this the 3D camera coordinate estimates were transformed to world coordinate frame using the known camera orientation and position. The mean distance between estimated 3D world position and ground truth was then calculated.

From the data we can see that the keypoint estimation using the ground truth vector field (α = 0), the only parameter which varies is the average time. What stands out in the table, is the fact that while mean 2D distance for $\alpha = 0$ is significantly lower compared to the other test, the mean 3D distance is higher compared to the other tests.

Also worth noting is that there appears to be a linear relationship between number of hypotheses, and the average time. For the mean 2D distance, while improving with an increased number of hypotheses, it is not a linear relationship. Using the data presented in Table 5.3, the average decrease in mean 2D distance when increasing the number of hypotheses from 5 to 10, is 30.2%. The same increase from 10 to 20 hypotheses results in an average decrease of 18.9%. Worth noting out here is that there are exceptions, and relatively small amount of data.

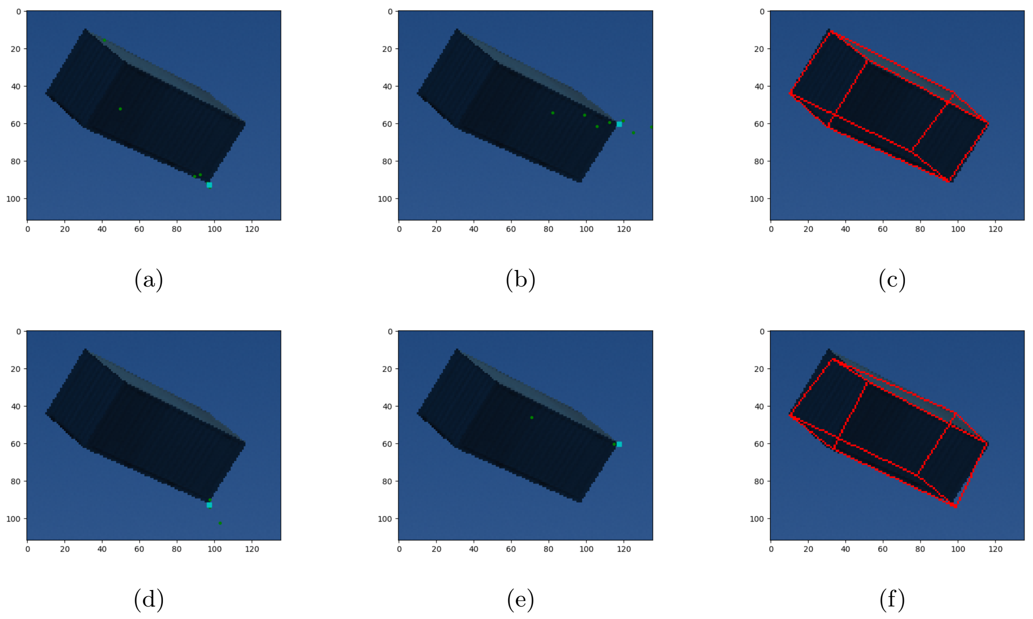


Figure 5.17: Keypoint hypotheses for α = 50, where the number of hypotheses for subfigures a-c is 10 and 5 in the case visualized in subfigures d-f.

The data also describes how the PnP RANSAC voting process handles outliers, as the increase of disturbance by increasing the value of α from 30% to 50%, results in a only a small increase of mean 2D distance and a negligible effect on mean 3D distance. This is further emphasised in Figure 5.17, which displays keypoint hypotheses, similarly as Figure 5.15.

To investigate if there was any correlation between the distance between the camera and container, and the resulting error of pose estimation, each distance and corresponding error from each image is presented in Figure 5.18. In this figure the distance between the camera and container, and the mean 3D error, of all the points for each container, is plotted. A regression line relating the two variables is also plotted. The data in this table presents the fact there is an increased mean error in the 3D point transformation with the estimated pose, as the distance between the camera and container increases. Interestingly, the data presented in the figure again states that the that the mean 3D transformation error for α = 0, as it increases at a higher rate and is continuously above the other regression lines, is higher compared to the two other cases when α = 30 and α = 50. Resulting from the data presented in Figure 5.18 is the mean error of the 3D coordinate estimation, in relationship to the distance between camera and container. The mean error for each value of α is presented in Table 5.4.

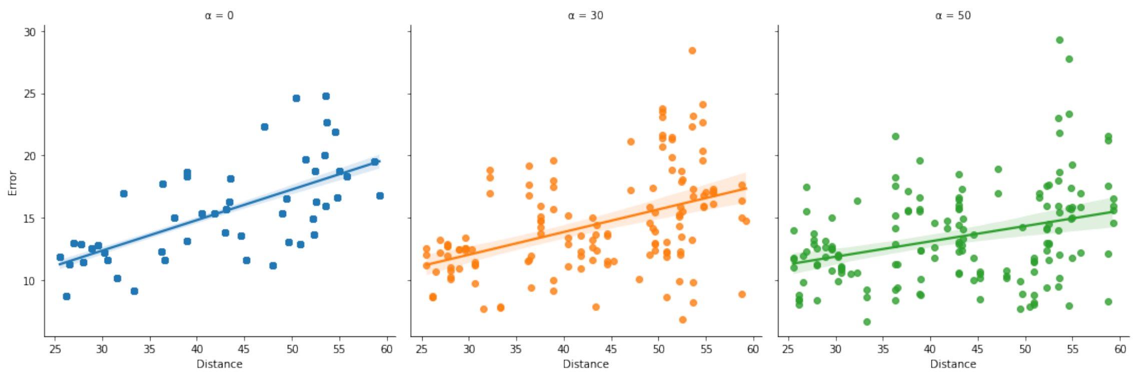


Figure 5.18: Scatterplot with a regression line relating the variables distance between container and camera, and mean 3D distance error.

PnP and RANSAC results					
Ransac threshold	Number of hypotheses	Mean 2D distance	Average Time	Mean 3D distance	α
0.9999	5	0.77	0.50	15.88	0
0.9999	10	0.77	0.95	15.88	0
0.9999	20	0.77	1.88	15.88	0
0.99	5	0.77	0.48	15.88	0
0.99	10	0.77	0.95	15.88	0
0.99	20	0.77	1.89	15.88	0
0.9	5	0.77	0.52	15.88	0
0.9	10	0.77	0.94	15.88	0
0.9	20	0.77	1.90	15.88	0
0.9999	5	2.74	0.39	14.83	30%
0.9999	10	1.58	0.81	15.61	30%
0.9999	20	1.51	1.51	15.33	30%
0.99	5	5.74	0.40	14.42	30%
0.99	10	3.37	0.77	14.03	30%
0.99	20	2.52	1.48	14.11	30%
0.9	5	6.53	0.43	15.17	30%
0.9	10	5.09	0.77	12.68	30%
0.9	20	4.50	1.54	13.72	30%
0.9999	5	3.85	0.36	14.73	50%
0.9999	10	2.50	0.68	15.13	50%
0.9999	20	1.45	1.37	14.61	50%
0.99	5	6.61	0.37	15.18	50%
0.99	10	4.64	0.68	15.11	50%
0.99	20	4.14	1.40	14.19	50%
0.9	5	7.98	0.38	12.56	50%
0.9	10	7.09	0.71	12.54	50%
0.9	20	5.68	1.38	12.27	50%

Table 5.3: PnP and RANSAC results on altered vector fields

SINTEF 6DPE ISO Container Dataset	
α	Mean error
0	36.94%
30	34.08%
50	33.46%

Table 5.4: Mean 3D error for different values of α

SINTEF 6DPE ISO Container Dataset		
Area of use	Filename	Number of images
Training U-Net	dataset	1000
Validation U-Net	val_dataset	250
Test U-Net	test_dataset	100
Training DVFnet	dataset_cropped	1000
Validation DVFnet	val_dataset_cropped	250

Table 5.5: SINTEF 6DPE ISO Container Dataset

5.3 SINTEF 6DPE ISO Container Dataset

The final synthetic dataset created for this thesis, named "SINTEF 6DPE ISO Container Dataset", contains 1250 instances of computer generated graphical images of containers in diverse environments and conditions of both weather and lighting. Along with the images comes the corresponding mask image, and the metadata files, which holds all the information of the given image, i.e the camera configuration and the localization and orientation of each container in the image used to calculate the final pose. The dataset is provided in two versions, one original version consisting of image of size 600x600 pixels and a cropped version used in the training of DVFnet. More details is displayed in Table 5.5.

The SINTEF 6DPE ISO Container Dataset contains images of a single container, positioned central in the image. The position and rotation of both the container and camera, is changed at a fixed rate. The positioning of the container was chosen, as a single container captured from a camera, at a high number of different camera angles, would result in a model able to generalize, by learning the features specific to shipping containers.

Chapter 6

Discussion

This chapter investigates the relationship between the findings presented in chapter 5, and theory uncovered during the research project. The chapter intends to address the objective and characteristics presented in Section 1.2.

6.1 Segmentation

This section discusses the results from the testing of UNETv.8 presented in Section 5.1. The second characteristic stated for the pipeline in the objective in Section 1.2 is to correctly detect shipping containers with an accuracy of 80% or more. UNETv.8 had an average dice score of 82,5% on its last epoch, meaning it upholds the capability of detection above 80%.

From the plot of UNETv.8's loss we can observe no significant signs of overfitting. This is clearly indicated when the validation loss begins to increase while the training loss remains stable or decrease (IBM 2021). In other words there is a high error occurring in the validation of the model and a low error during training. Overfitting will always occur when training a deep neural network (Ying 2019). Looking at the loss of UNETv.8 we can see that the validation loss follows the training loss quite closely, indicating that the model is generalizing well. The oscillations seen in the validation loss is most likely due to the fact that the model uses dropout during training and not when validating (Smith and C. Chen 2018). There appears to be no signs of underfitting, which is good as this indicates that the model is learning the features for segmentation (IBM 2021).

6.1.1 Prediction on synthetic data

The results from the predictions done by UNETv.8 on synthetic test data in Section 5.1.1 shows that the model overall predicts accurately on new instances of unseen synthetic data. This implies that the model has generalized to a satisfactory level. All but one test shared a dice score above 90%, displaying excellent segmentation. The outlier was the very hard prediction, Figure 5.5 with a dice score of 47%. By looking closer at the prediction we can observe that the model is predicting where the garage door on the building is located. This is not surprising as the container is located in the foreground of the door and shares a shade of blue not far off from the color of the garage door. The garage door also has a cuboid shape which could further throw the model off. A solution to this issue is to provide further domain randomization, described in Section 3.1.1, to the data, making sure such cases are covered in the set (Loquercio et al. 2020). The model also predicted, though not as confidently, a container in the sea in the top center right of the image, but because of the confident threshold of 60% implemented in the crop function, see appendix, the prediction was purged from the crop, subsequently the crop managed to be of good quality and be usable for the DVFnet.

6.1.2 Prediction on real data

The results from the predictions done by UNETv.8 on real test data in Section 5.1.2 shows a variation in performances. As illustrated in Figure 5.6, the model outputs a poor prediction on the easy image and consequently a poorly cropped image. It gives a strong prediction in the lower right corner of the container. Why it does this is hard to deduce directly from the prediction alone, but it could be an indication of the model trying to detect a smaller container. The synthetic training set generated, as described in Section 3.1, contains images with containers located 25m to 60m away from the camera, while the distance of the container in the real image provided in Figure 5.6, though exact distance is unknown, is far closer to the camera. The real image is thus not representative of the training data distribution and this domain deviation could be why the model has an unexpected performance drop (J. Chen et al. 2021).

Domain deviation could be the case in Figure 5.7 as well, since there are no instances of a container partially submerged in water in the training set. This issue can be solved however, by generating instances of the domain deviations and expanding the training data to include such instances. This phenomenon can also explain the surprising results of the real data test; that the performance was relatively good on the hard and very hard images. These images, while cluttered and labeled difficult, are more analogous with the synthetic training set. For both images we can observe that the container is in mid air and is at a greater distance from the camera, similar to the data in the synthetic training set, than the easy and medium difficulty predictions. Even so, UNETv.8 displays good generalization in these predictions, ignoring the cargo holding doors, and predicting correctly on the container even though it is attached to a container spreader, which is a domain configuration not included in the training set. Similarly, in the very hard image UNETv.8, surprisingly, completely ignores the other containers on the container freight to the left in the image. This could be because of the fact that a clustered group of containers are treated as background since it has no references to such a formation in the training set. Nevertheless, the model trained on synthetic data displays a promising detection capability on real data.

6.2 Pose estimation

A vital characteristic in meeting the objective defined in Section 1.2, and support the case provided by SINTEF Ocean, is to accurately estimate object pose in real-time. This section discusses the relationship between the findings obtained as a result of the development and implementation of DVFnet, Section 4.2.2, and the theory and previous research presented in this thesis. In this thesis, a keypoint-based method was chosen, more specifically a model generating a unit vector field was developed. The criteria for selection was theoretical indications that this type of model will handle occlusions and other visibility challenges, while achieving pose estimation in real time, defined as computation time of 0.25 seconds for one image.

6.2.1 DVFnet

From the plots of training and validation loss of DVFnetv_1, DVFnetv_2 and DVFnetv_3 we can observe quite evidently overfitting occurring in all models. The training loss drops to a low value and stabilizes quickly while the validation loss maintains a high value. Of the three models, DVFnetv_1 shows the least overfitting, although still excessive. By definition the models are becoming very good at predicting the training data, but fails on the validation data. This is a clear sign that the models cant generalize well (C. Zhang et al. 2018). The question to why the networks are overfitting is a broad question with numerous possible answers. The problem of overfitting was taken into account during the design of DVFnet. As described in Section 4.2.2, the architecture of DVFnet incorporates different tactics to reduce overfitting such as dropout, batch normalization and reduction of the number of features (Alhichri et al. 2019), (Elleuch et al. 2018).

Another proposed solution to reduce overfitting is adding more augmentations to the dataset. During training using DVFnet, augmentations such as color jitter and Gaussian blur are applied with a given probability and intensity to the training data. However, this might not be enough and Ma et al. (2019) proposes that global augmentations, meaning altering all values in an input, of rotations, translations and compression are more effective against overfitting. Adding such augmentations in combination with altering pixel values could be successful in reducing overfitting (Tian et al. 2020).

Although DVFnetv_1 overfits the data, further investigation of DVFnetv_1 is need to complete the performance analysis. From the vector field predictions provided in Figure 5.13a, we can observe the results on an image from the validation set. Focusing on the vector field calculated within the container one can deduce that the estimations are, while poor, not entirely off. Going forward, we denominate the predictions as $\hat{v}_1, \hat{v}_2, ..., \hat{v}_9$, where $\hat{v}_1$ is the prediction in the top left and so on. Looking at $\hat{v}_1$ we can see that, by using the color wheel for vector direction from Section 3.3, the unit vectors estimated do in fact point roughly towards the current keypoint. This is the case as well for $\hat{v}_2, \hat{v}_4$ and $\hat{v}_8$. However, looking at $\hat{v}_3, \hat{v}_5, \hat{v}_6, \hat{v}_7$ and $\hat{v}_9$ we can observe that the unit vectors estimated are pointing in different directions, essentially meaning that the model is guessing. These observations are in line with the loss plot of the model, indicating overfitting but an ability to generalize to some extent.

Another possible reason for the poor performance of DVFnet could be a batch size problem. Although the batches created included the appropriate amount of images according to its size, the effective batch size became one. This is because of the cropping of the images resulted in different image dimensions and as such could not be stacked into a tensor. Training a network with a very low batch size is sub-optimal in most cases(Masters and Luschi 2018). Masters and Luschi (2018) states that for very small batches, the estimation of the batch mean and variance can become very noisy, which may limit the effectiveness of Batch Normalization, which DVFnet uses, in reducing the covariate shift. A failure in reducing the covariate shift can lead to generalization problems, as observed in the loss.

While the RANSAC PnP manages to, in most cases, estimate keypoint location on the container surface, the methods yields no pose usable for further calculations. In addition to this, plotting the hypotheses generated during pose estimation, as done in Figure 5.15, further indicates that the model cannot locate the container keypoints in a sufficient manner. A plausible cause contributing to these results are symmetries in the container objects. In the estimated keypoint- and object pose visualizations, a pattern was observed where a large number of keypoints tend to be located towards the center of mass of the container object, as illustrated in Figure 5.14.

Symmetric objects, whether they a continuous, such as a bottle, or discrete (e.g box) as they are in this case, have multiple visually, almost, indistinguishable points. In this thesis, when a continuous function (CNN) maps unit vectors towards the keypoints, it can exist multiple sets of coordinates that describe different but equally valid poses (Richter-Klug and Frese 2020). These symmetries therefore create ambiguities when estimating the object pose. As a network attempts to learn these features, it would converge to a model predicting the average of possible poses (Pitteri et al. 2019). It can thus be suggested that the poses predicted in Figure 5.14 are an average estimation of possible poses, as the network is not able to properly distinguish object features. This could also be the reason why a large portion of the keypoint hypotheses in Figure 5.15, are on the container surface and towards the mass center of the container.

6.2.2 PnP RANSAC

As pose estimation using the predicted vector fields from DVFnet did not yield any usable results, an altered vector field, derived from ground truth data, was used to investigate if the RANSAC PnP keypoint-voting process is a viable option to use in a pipeline, as described in Section 1.2.

The parameter α was defined as a parameter describing to what degree the ground truth vector field is altered. Three values were chosen for α: 0, 30% and 50%. This means that the RANSAC PnP keypoint-voting process was tested with three different vector fields, where one was the ground truth data. The various values of α to be tested, and the probability of which a coordinate is altered, was decided empirically. The argumentation for this, was that it is essential to determine the viability of the RANSAC PnP keypoint-voting process in order to meet the objective presented in Section 1.2 and appropriately support the case provided.

The most obvious finding to emerge from these tests was that there is a linear relationship between the computational time, and the number of hypotheses. This applies to all vector fields tested, regardless of the value of α. Another important finding is that while the 2D distance between ground truth and estimated keypoint location decrease with an increased number of hypothesis, the two parameters do not have a linear relationship. This finding suggest that for the data used in this thesis, RANSAC is an efficient method of eliminating outliers in the data, and obtaining results which on average differs from ground truth estimations with just a few pixels. This is especially true in cases in which a higher RANSAC threshold was used.

Combining this with the aforementioned thesis objective could suggest that in order to implement a model able to estimate pose in real time, a high RANSAC threshold combined with a relatively low number of hypotheses generated could be favorable. However, with an increased number of hypotheses the performance improves, as visualized in, Figure 5.17. Previously described in Section 2.6, RANSAC is a method where samples are randomly selected. This means that the number of outliers, relative to the total dataset size will affect how efficient the method is. Therefore, in order to achieve a high performing pipeline, the parameters chosen for RANSAC could be adapted to the quality of the vector field estimation, where a low quality vector field would need a higher number of hypotheses as there are more outliers. Similarly, as stated in Hossein-Nejad and Nasri (2018) the RANSAC threshold value is considered empirically. Selection of the appropriate threshold value in RANSAC is a critical problem. If a small value is chosen, the rate of true matches diminishes; and in the case of big value selection, we face the increased rate of mismatches.

The results of this study shows that there is a correlation in the distance between the camera and container, and the mean 3D distance error. Intuitively this is expected, as the object becomes smaller with an increased distance, and relative distance between the keypoints becomes smaller, thereby the objects pose is visually more difficult to interpret. Surprisingly, it was found that the mean 3D distance increased as the noise of the data decreased. This is counter-intuitive as the pose obtained from ground truth data should be accurate, and the mean 3D distance error should decrease, in cohesion with the mean 2D distance error. A previous study utilizing the OpenCV solvePnP() function, observed a consistent error in the x-axis direction during pose estimation (Kriegler and Wöber 2020). Apart from that, to the best of our knowledge there is little found in the literature on the question of OpenCVs solvePnP() with consistent inaccurate results from reliable and accurate data. There is also no similar systematical axis errors in our data, as reported in Kriegler and Wöber (2020). The function solvePnP() is stated to estimate the object pose (OpenCV 2022), and as the data provided from ground truth data involves no stochastic variations, the presented mean 3D distance was theorized to be an varying error occurring during pose estimation. However, this theory is questionable, as visualized in Figure 5.18, for $\alpha = 0$ the data shown a lower degree of variations especially for an increasing distance.

As UNETv.8 is able to generalize and segment real data, it can thus be suggested that the use of synthetic data is a viable alternative for data generation in object segmentation. However, the same can not be said for DVFnet, as no pose could be obtained using the generated vector field predictions. The PnP RANSAC voting results presented in this thesis is still highly relevant to the objective presented, as the PnP RANSAC voting is conducted using the camera intrinsic matrix, a set of 3D points and corresponding 2D image projections of the 3D points. Hence, the process of estimating pose using 3D to 2D correspondences, is conducted without the use of the image data.

6.3 The pipeline

The purpose of this thesis is to investigate the possibilities of implementing a monocular object detection and object estimation pipeline. To reach object pose estimation, a strong object detection foundation is needed to secure the calculation of reliable poses. The objective of this thesis is defined as:

Research and implement a pipeline of CV models that utilize object detection to detect and classify shipping containers in order to estimate the pose of a shipping container in real-time.

This section concerns the objective of this thesis, and directly addresses each of the following characteristics vital for the proposed CV models defined in Section 1.2.

- The model needs to be able to process data in real-time.
- The model needs to be able to detect shipping containers in images with an accuracy of 80% or more.
- The model needs to be able to estimate container coordinates in 3D space with an error of less than 10% of the distance between camera and container.

As discussed in Zhou et al. (2021) and Price et al. (2021), a CV object tracking system which supports crane operations, is required to update at a frequency high enough to detect abnormalities, and for human personnel or security systems to intervene. The average total time for the total pipeline resulted in 2.23 seconds, conducted on one V100 16GB GPU. In this thesis real-time processing was defined as a computation time of 0.25 s for each image, which is significantly lower than the average processing time achieved.

While code optimization could reduce the computational time, and the entire pipeline is implemented in Python, which is a programming language generally expected to run slower compared to several other programming languages (Python 2022). It also is worth mentioning that this computational time is on a specific GPU, and that the computational time of the pipeline, run on other GPUs could be different. However, a computational of such a high value does not pass the characteristics defined the thesis objective. A central aspect of the problem presented in Section 1.1 is the safety aspect of an autonomous crane, the speed characteristic defined is essential in giving the security systems or human personnel time to intervene in case of potential hazards.

As the average error resulting from the PnP pose estimation being well over 30%, for each of the generated altered vector fields, the EPnP achieves an accuracy significantly higher than the characteristics defined in the objective. As precision and accuracy is essential, especially as the crane operations will be conducted at sea, the error presented is very high. EPnP is a non-iterative method chosen for its computational speed, and in this project the algorithm was used with eight local 3D object coordinates for the containers. The number of local coordinates may be relatively low and increasing the number of local coordinates, as stated in Lepetit et al. (2008) increasing the number of keypoint generally increases pose estimations.

Chapter 7

Conclusion

This research aims to identify opportunities within the field of automation of crane operations, by retrofitting a hydraulic crane with sensors and systems to increase the degree of autonomy, in order to improve personnel safety and reduce time- and energy usage. The sensor and system in question is a monocular object pose estimation system processing a single RGB image with real-time capabilities. Based on extensive research, a synthetic 6D pose estimation dataset was constructed, as well as a proposed CV pipeline utilizing semantic image segmentation, and estimating 6D object pose of shipping containers.

The segmentation model presented in this thesis, UNETv.8, performed above the 80% accuracy threshold stated in the objective. The model also does not overfit and shows good generalization ability. However, the model struggles when presented with data that stray too much from the domain of the training set, or when the container becomes to obscure to distinguish from its background. Subsequently, implementing the cropping algorithm allowed for decent cropping in spite of average and below average segmentation. In all, this indicates that the first part of the pipeline, the detection segment, works as intended.

The success of UNETv.8 also indicates that the synthetic dataset worked as a replacement of real data, following good predictions on real data closely related to the synthetic image domain. However, the same conclusion can not be drawn for the proposed vector field estimation network, DVFnet, as it struggled with performance and therefore provided inconclusive results. Overfitting the training data, indicates that the models were memorizing the training data rather than learning. Of the models trained, DVFnetv_1 was the model showing the least degree of overfitting, though substantial enough to be regarded as unusable. The model's failure in generalizing new instances of data, although some predictions were not far off, the majority were of such poor quality it can be regarded as pure guessing. The addition of rotational, translational and compression augmentations to the training data could reduce the prominent overfitting. This could be a good starting point for enhancing the networks performance. To further increase the generalization abilities, and subsequently the accuracy of both UNETv.8 and DVFnet, an expansion of the dataset could be implemented.

Moving forward from vector field estimation to keypoint localization it is concluded that the RANSAC voting process is efficient at handling outliers, but the choice of input parameters are project specific and needs to be adjusted in order to achieve an efficient and robust algorithm. Another finding is that there is a positive correlation between the distance between camera and container, and the pose return from the solvePnP() function. The estimated pose is also prone to a consistent error, which is significantly higher than the characteristics defined in the objective. Both these findings supports the idea that the tool, as implemented in this thesis, could not be used as a component in a pipeline estimating 6D pose.

To summarize, based on the findings in this thesis the proposed monocular object pose estimation pipeline did not work as intended. The hydraulic crane needs to be both precise and accurate.

Therefore, in order to be a viable replacement of manually operated tasks, the CV pipeline needs to provide accurate positional and rotational data of an object, which at this point is not obtained. The pipeline is also currently processing images at a computational time which is not defined as real-time, as defined in this thesis. In spite of this, the RANSAC voting section and the detection segment in the pipeline could be further utilized in similar pose estimation pipelines, where the other components' performance are increased. In order to achieve this, the following further work, based on the results presented in this thesis is recommended.

7.1 Limitations and Further Work

As discussed in Section 6.1, a possible solution to increase generalization, and consequently accuracy, is to expand the dataset further with images of the container in new configurations. As of now the container is always in the air and often centralized in the image. Adding more situations where the container is on the ground, partially occluded by water, completely submerged, closer to camera and connected to a crane are some configurations that could further increase generalization. Since the source code for the dataset created for this thesis is highly customizable one can even replace the container 3D model with another object to test the pipeline on different objects in cohesion to SINTEF's and M/S Torra's operations.

The performance of DVFnet displays, as concluded in chapter 7, a need for a overhaul of the network and its training process. Some of the solutions discussed were to increase the amount of augmentations applied during the training process, as using color jitter and Gaussian blur were proven not to be enough for breaking up the similarity in the data. Introducing rotational, translational and compression augmentations could provide better generalization and thus reduce the problem of overfitting. Additionally, expanding the cropping algorithm to scale all crops to the same dimensions can fix the covariate shift problem.

To handle the ambiguities resulting from the symmetrical properties of the container, the study should be repeated using a dataset where all containers are rotated at a canonical position as done in Rad and Lepetit (2017). Another alternative that could be investigated is an approach conducted in Hodan et al. (2020) where the object surface is represented as fragments allowing for a prediction of possibly multiple correspondences per pixel.

The main limitation of the EPnP implementation in this thesis, was that the method was only tested with eight local coordinates. This was done to reduce the computational time, but future projects should investigate if increasing the number of local object coordinates increase the performance of the pose estimation, and what impact this has on the computational time. Also, as the method of solving PnP used in this thesis was EPnP, in future projects, other PnP solving methods should be tested in order to detect if this could increase the pose estimation.

After the pipeline has gone through the changes presented above it could be beneficial to test its accuracy and speed on benchmark datasets, such as LINEMOD or T-Less. Doing so will put the pipeline up against SOTA (State-of-the-art) models and pipelines to see how it compares. This will signify if the pipeline is translatable to pose estimation task on other objects and situations other than the offshore environment.

Bibliography

1. Abiodun, Oludare Isaac et al. (2018). 'State-of-the-art in artificial neural network applications: A survey'. In: *Heliyon* 4.11, e00938. ISSN: 2405-8440. DOI: https://doi.org/10.1016/j.heliyon.2018.e00938. URL: https://www.sciencedirect.com/science/article/pii/S2405844018332067.

2. Albawi, Saad, Tareq Abed Mohammed and Saad Al-Zawi (2017). 'Understanding of a convolutional neural network'. In: *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

3. Alhichri, Haikel et al. (2019). 'Helping the Visually Impaired See via Image Multi-labeling Based on SqueezeNet CNN'. In: *Applied Sciences* 9.21. ISSN: 2076-3417. DOI: 10.3390/app9214656. URL: https://www.mdpi.com/2076-3417/9/21/4656.

4. Aranjuelo, Nerea et al. (2021). 'Key strategies for synthetic data generation for training intelligent systems based on people detection from omnidirectional cameras'. In: *Computers Electrical Engineering* 92, p. 107105. ISSN: 0045-7906. DOI: https://doi.org/10.1016/j.compeleceng.2021.107105.

5. BOP (2022). *BOP: Benchmark for 6D Object Pose Estimation.* Last accessed 15 May 2022. URL: https://bop.felk.cvut.cz/datasets/.

6. Bremvåg, Terje (2015). Last accessed 07 May 2022. URL: https://www.sintef.no/globalassets/sintef-fiskeri-og-havbruk/aquanor2015/brosjyre-torra.pdf.

7. Bukschat, Yannick and Marcus Vetter (2020). 'EfficientPose: An efficient, accurate and scalable end-to-end 6D multi object pose estimation approach'. In: abs/2011.04307. DOI: 10.48550/ARXIV.2011.04307. URL: https://arxiv.org/abs/2011.04307.

8. Buntine, Wray (2020). *Machine learning after the deep learning revolution.* URL: https://link.springer.com/article/10.1007/s11704-020-0800-8#citeas.

9. Caner Sahin Guillermo Garcia-Hernando, Juil Sock Tae-Kyun Kim (2019). 'Instance- and Category-Level 6D Object Pose Estimation'. In: *RGB-D Image Analysis and Processing.* Cham: Springer International Publishing, pp. 243–265. ISBN: 978-3-030-28603-3. DOI: 10.1007/978-3-030-28603-3_11. URL: https://doi.org/10.1007/978-3-030-28603-3_11.

10. Chen, Jiefeng et al. (2021). *Detecting Errors and Estimating Accuracy on Unlabeled Data with Self-training Ensembles.* DOI: 10.48550/ARXIV.2106.15728. URL: https://arxiv.org/abs/2106.15728.

11. Chen, Xu et al. (2020). *Category Level Object Pose Estimation via Neural Analysis-by-Synthesis.* DOI: 10.48550/ARXIV.2008.08145. URL: https://arxiv.org/abs/2008.08145.

12. Code, Papers With (2017). Last accessed 07 May 2022. URL: https://paperswithcode.com/task/6d-pose-estimation.

13. Cordts, Marius et al. (2016). *The Cityscapes Dataset for Semantic Urban Scene Understanding*. DOI: 10.48550/ARXIV.1604.01685. URL: https://arxiv.org/abs/1604.01685.

14. Dain Lin, Psang and Shyh Guang Hwang (1994). 'Modeling for error analysis of cam mechanisms by using 4 × 4 homogeneous transformation matrix'. In: *Mathematical and Computer Modelling* 19.5, pp. 33–42. ISSN: 0895-7177. DOI: https://doi.org/10.1016/0895-7177(94)90088-4. URL: https://www.sciencedirect.com/science/article/pii/0895717794900884.

15. de Melo, Celso M. et al. (2022). 'Next-generation deep learning based on simulators and synthetic data'. In: *Trends in Cognitive Sciences* 26.2, pp. 174–187. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2021.11.008. URL: https://www.sciencedirect.com/science/article/pii/S136466132100293X.

16. Dishashree (2020). URL: https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/.

17. Dwibedi, Debidatta, Ishan Misra and Martial Hebert (2017). 'Cut, Paste and Learn: Surprisingly Easy Synthesis for Instance Detection'. In: DOI: 10.48550/ARXIV.1708.01642. URL: https://arxiv.org/abs/1708.01642.

18. Eade, E. and T. Drummond (2006). 'Scalable Monocular SLAM'. In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 1, pp. 469–476. DOI: 10.1109/CVPR.2006.263.

19. Elhamraoui, Zahra (2020). *Introduction to convolutional neural network*. Last accessed 8. May 2022. URL: https://medium.com/analytics-vidhya/introduction-to-convolutional-neural-network-6942c189a723.

20. Elleuch, Mohamed, Adel M. Alimi and Monji Kherallah (2018). 'Enhancement of Deep Architecture using Dropout/ DropConnect Techniques Applied for AHR System'. In: *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–6. DOI: 10.1109/IJCNN.2018.8489245.

21. Fan, Zhaoxin et al. (2021). *Deep Learning on Monocular Object Pose Detection and Tracking: A Comprehensive Overview*. DOI: 10.48550/ARXIV.2105.14291. URL: https://arxiv.org/abs/2105.14291.

22. Fang, Yihai et al. (2020). 'Cyber-Physical Systems (CPS) in Intelligent Crane Operations'. In: *Cyber-Physical Systems in the Built Environment*. Ed. by Chimay J. Anumba and Nazila Roofigari-Esfahan. Cham: Springer International Publishing, pp. 175–192. ISBN: 978-3-030-41560-0. DOI: 10.1007/978-3-030-41560-0_10. URL: https://doi.org/10.1007/978-3-030-41560-0_10.

23. Fenstad, Arne (2021). Last accessed 07 May 2022. URL: https://www.tu.no/artikler/flere-alvorlige-ulykker-med-skipskraner-i-ar-losningen-er-under-utvikling/511840.

24. Fischler, Martin A. and Robert C. Bolles (June 1981). 'Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography'. In: *Commun. ACM* 24.6, pp. 381–395. ISSN: 0001-0782. URL: https://doi.org/10.1145/358669.358692.

25. Ge, Rundong and Giuseppe Loianno (2021). 'VIPose: Real-time Visual-Inertial 6D Object Pose Tracking'. In: URL: https://arxiv.org/abs/2107.12617.

26. Gómez-Flores, Wilfrido and Wagner Coelho de Albuquerque Pereira (2020). 'A comparative study of pre-trained convolutional neural networks for semantic segmentation of breast tumors in ultrasound'. In: *Computers in Biology and Medicine* 126, p. 104036. ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2020.104036. URL: https://www.sciencedirect.com/science/article/pii/S001048252030367X.

27. Gooodfellow, Ian, Yoshua Bengio and Aaron Courville (2016). *Deep Learning*. The MIT Press. URL: https://books.google.no/books?id=omivDQAAQBAJ&lpg=PR5&dq=deep%20learning&lr&hl=no&pg=PA7#v=onepage&q&f=false.

28. Guissous, Alla Eddine (2019). *Skin Lesion Classification Using Deep Neural Network.* DOI: 10.48550/ARXIV.1911.07817. URL: https://arxiv.org/abs/1911.07817.

29. Hartley, Richard (2003). *Multiple view geometry in computer vision.* eng. Second edition. Cambridge: Cambridge University Press. ISBN: 1-107-14169-9.

30. Hata, Kenji and Silvio Savarese (2021). *Camera Models.* lecture notes, Computer Vision, From 3D Reconstruction to Recognition CS231A, Stanford University, delivered 02 April 2021. URL: https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf.

31. Heinrich, Stuart B. (2013). 'Efficient and robust model fitting with unknown noise scale'. In: *Image and Vision Computing* 31.10, pp. 735–747. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2013.07.003. URL: https://www.sciencedirect.com/science/article/pii/S0262885613001078.

32. Hesch, Joel A. and Stergios I. Roumeliotis (2011). 'A Direct Least-Squares (DLS) method for PnP'. In: *2011 International Conference on Computer Vision*, pp. 383–390.

33. Hinton, Geoffrey E. (1992). 'How Neural Networks Learn from Experience'. In: *Scientific American* 267.3, pp. 144–151. URL: http://www.jstor.org/stable/24939221.

34. Hodan, Tomas, Daniel Barath and Jiri Matas (2020). *EPOS: Estimating 6D Pose of Objects with Symmetries.* DOI: 10.48550/ARXIV.2004.00605. URL: https://arxiv.org/abs/2004.00605.

35. Hol, Jeroen D. et al. (2009). 'Tightly coupled UWB/IMU pose estimation'. In: *2009 IEEE International Conference on Ultra-Wideband*, pp. 688–692. DOI: 10.1109/ICUWB.2009.5288724.

36. Hossein-Nejad, Zahra and Mehdi Nasri (2018). 'A-RANSAC: Adaptive random sample consensus method in multimodal retinal image registration'. In: *Biomedical Signal Processing and Control* 45, pp. 325–338. ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2018.06.002. URL: https://www.sciencedirect.com/science/article/pii/S174680941830154X.

37. Huang, Jonathan et al. (2016). *Speed/accuracy trade-offs for modern convolutional object detectors.* DOI: 10.48550/ARXIV.1611.10012. URL: https://arxiv.org/abs/1611.10012.

38. IBM (2021). URL: https://www.ibm.com/cloud/learn/overfitting.

39. Irina Peregud, Anastasiya Zharovskikh (2020). 'Computer Vision Applications Examples Across Different Industries'. In: URL: https://indatalabs.com/blog/applications-computer-vision-across-industries.

40. Jain, Abhinav et al. (2020). 'Overview and Importance of Data Quality for Machine Learning Tasks'. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery Data Mining.* KDD '20. Virtual Event, CA, USA: Association for Computing Machinery, pp. 3561–3562. ISBN: 9781450379984. DOI: 10.1145/3394486.3406477. URL: https://doi.org/10.1145/3394486.3406477.

41. Jo, Junho et al. (Nov. 2020). 'Handwritten Text Segmentation via End-to-End Learning of Convolutional Neural Networks'. In: *Multimedia Tools Appl.* 79.43–44, pp. 32137–32150. ISSN: 1380-7501. DOI: 10.1007/s11042-020-09624-9. URL: https://doi.org/10.1007/s11042-020-09624-9.

42. Josifovski, Josip et al. (2018). 'Object Detection and Pose Estimation Based on Convolutional Neural Networks Trained with Synthetic Data'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6269–6276. DOI: 10.1109/IROS.2018.8594379.

43. Khan, Zia et al. (2020). 'Evaluation of Deep Neural Networks for Semantic Segmentation of Prostate in T2W MRI'. In: *Sensors* 20.11. ISSN: 1424-8220. DOI: 10.3390/s20113183. URL: https://www.mdpi.com/1424-8220/20/11/3183.

44. Kingma, Diederik P. and Jimmy Ba (2014). *Adam: A Method for Stochastic Optimization.* DOI: 10.48550/ARXIV.1412.6980. URL: https://arxiv.org/abs/1412.6980.

45. Kothari, C.R (2004). *Research methodology : methods techniques.* 2nd. New Age International (P) Ltd., Publishers.

46. Kriegler, Andreas and Wilfried Wöber (Sept. 2020). 'Vision-based Docking of a Mobile Robot'. In: DOI: 10.3217/978-3-85125-752-6.

47. Lab, Stanford Vision (2021). *ImageNet.* Last accessed 25 May 2022. URL: https://image-net.org/index.php.

48. Lane, Thom (2018). *Transposed Convolutions explained with... MS Excel!* URL: https://medium.com/apache-mxnet/transposed-convolutions-explained-with-ms-excel-52d13030c7e8.

49. Lepetit, Vincent, Francesc Moreno-Noguer and P. Fua (2008). 'EPnP: An Accurate O(n) Solution to the PnP Problem'. In: *International Journal of Computer Vision* 81, pp. 155–166. URL: https://doi.org/10.1007/s11263-008-0152-6.

50. Li, Junjie and Ding Liu (Apr. 2021). 'Information Bottleneck Theory on Convolutional Neural Networks'. In: *Neural Processing Letters* 53, pp. 1–16. DOI: 10.1007/s11063-021-10445-6.

51. Li, Shiqi, Chi Xu and Ming Xie (2012). 'A Robust O(n) Solution to the Perspective-n-Point Problem'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34, pp. 1444–1450.

52. Li, Xiaolong et al. (2019). *Category-Level Articulated Object Pose Estimation.* DOI: 10.48550/ARXIV.1912.11913. URL: https://arxiv.org/abs/1912.11913.

53. Li, Z. et al. (Feb. 2022). *cardiGAN: A Generative Adversarial Network Model for Design and Discovery of Multi Principal Element Alloys.*

54. Lin, Tsung-Yi et al. (2014). 'Microsoft COCO: Common Objects in Context'. In: *Computer Vision – ECCV 2014.* Ed. by David Fleet et al. Springer International Publishing, pp. 740–755.

55. Liu, Liangliang et al. (2020). 'Multi-Receptive-Field CNN for Semantic Segmentation of Medical Images'. In: *IEEE Journal of Biomedical and Health Informatics* 24.11, pp. 3215–3225. DOI: 10.1109/JBHI.2020.3016306.

56. Long, Jonathan, Evan Shelhamer and Trevor Darrell (June 2015). 'Fully Convolutional Networks for Semantic Segmentation'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR).*

57. Loquercio, Antonio et al. (2020). 'Deep Drone Racing: From Simulation to Reality With Domain Randomization'. In: *IEEE Transactions on Robotics* 36.1, pp. 1–14. DOI: 10.1109/TRO.2019.2942989.

58. Loshchilov, Ilya and Frank Hutter (2017). *Decoupled Weight Decay Regularization.* DOI: 10.48550/ARXIV.1711.05101. URL: https://arxiv.org/abs/1711.05101.

59. Lv, Yaowen et al. (2015). 'A new robust 2D camera calibration method using RANSAC'. In: *Optik* 126.24, pp. 4910–4915. ISSN: 0030-4026. DOI: https://doi.org/10.1016/j.ijleo.2015.09.117. URL: https://www.sciencedirect.com/science/article/pii/S0030402615011869.

60. Ma, Rui, Pin Tao and Huiyun Tang (2019). 'Optimizing Data Augmentation for Semantic Segmentation on Small-Scale Dataset'. In: *Proceedings of the 2nd International Conference on Control and Computer Vision.* ICCCV 2019. Jeju, Republic of Korea: Association for Computing Machinery, pp. 77–81. ISBN: 9781450363228. DOI: 10.1145/3341016.3341020. URL: https://doi.org/10.1145/3341016.3341020.

61. Masters, Dominic and Carlo Luschi (2018). *Revisiting Small Batch Training for Deep Neural Networks.* DOI: 10.48550/ARXIV.1804.07612. URL: https://arxiv.org/abs/1804.07612.

62. MathWorld, Wolfram (2022). Last accessed 27 May 2022. URL: https://mathworld.wolfram.com/EulerAngles.html.

63. Menze, Moritz and Andreas Geiger (2015). 'Object scene flow for autonomous vehicles'. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3061–3070. DOI: 10.1109/CVPR.2015.7298925.

64. Mittal, Abhinav Agrawal Namita (2019). 'Using CNN for facial expression recognition: a study of the effects of kernel size and number of filters on accuracy'. In: *Vis Comput* 36, pp. 405–412. DOI: https://doi.org/10.1007/s00371-019-01630-9.

65. Movshovitz-Attias, Yair, Takeo Kanade and Yaser Sheikh (2016). 'How Useful Is Photo-Realistic Rendering for Visual Learning?' In: ed. by Gang Hua and Hervé Jégou, pp. 202–217.

66. MUNIN (2016). *MUNIN's Rationale.* URL: http://www.unmanned-ship.org/munin/about/munins-rational/.

67. Nagi, Jawad et al. (2011). 'Max-pooling convolutional neural networks for vision-based hand gesture recognition'. In: *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pp. 342–347. DOI: 10.1109/ICSIPA.2011.6144164.

68. Nasir, Ahmad Kamal and Hubert Roth (2012). 'Pose Estimation By Multisensor Data Fusion Of Wheel Encoders, Gyroscope, Accelerometer And Electronic Compass'. In: *IFAC Proceedings Volumes* 45.4. 1st IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control, pp. 49–54. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120403-3-DE-3010.00068. URL: https://www.sciencedirect.com/science/article/pii/S1474667015404410.

69. Nayar, Shree (2021a). *Linear Camera Model — Camera Calibration.* Last accessed 07 May 2022. URL: https://www.youtube.com/watch?v=qByYk6JggQU.

70. — (2021b). *Pinhole and Perspective Projection—Image Formation.* Last accessed 07 May 2022. URL: https://www.youtube.com/watch?v=_EhY31MSbNM.

71. Netherlands, CBS - Statistics (2021). *TEU (Twenty foot equivalent unit).* Last accessed 15 May 2022.

72. Neumann-Cosel, Kilian von et al. (2009). 'Testing of Image Processing Algorithms on Synthetic Data'. In: *2009 Fourth International Conference on Software Engineering Advances*, pp. 169–172. DOI: 10.1109/ICSEA.2009.34.

73. Ngo, Quang Hieu and Keum-Shik Hong (2012). 'Sliding-Mode Antisway Control of an Offshore Container Crane'. In: *IEEE/ASME Transactions on Mechatronics* 17.2, pp. 201–209. DOI: 10.1109/TMECH.2010.2093907.

74. Nikolenko, Sergey I. (2021). *Synthetic data for deep learning.* Vol. Volume 174. Springer optimization and its applications ; Springer. ISBN: 3-030-75178-3.

75. Nordal, Håvard (2021). Last accessed 07 May 2022. URL: https://blogg.sintef.no/sintefocean-nb/autonome-kraner-skal-gi-gronnere-logistikk/.

76. O'Shea, Keiron and Ryan Nash (2015). *An Introduction to Convolutional Neural Networks.* DOI: 10.48550/ARXIV.1511.08458. URL: https://arxiv.org/abs/1511.08458.

77. Oberweger, Markus, Mahdi Rad and Vincent Lepetit (2018). *Making Deep Heatmaps Robust to Partial Occlusions for 3D Object Pose Estimation.* arXiv: 1804.03959 [cs.CV].

78. OpenCV (2022). *Perspective-n-Point (PnP) pose computation.* Last accessed 07 May 2022. URL: https://docs.opencv.org/4.x/d5/d1f/calib3d_solvePnP.html.

79. Opromolla, Roberto et al. (2015). 'Uncooperative pose estimation with a LIDAR-based system'. In: *Acta Astronautica* 110. Dynamics and Control of Space Systems, pp. 287–297. ISSN: 0094-5765. DOI: https://doi.org/10.1016/j.actaastro.2014.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0094576514004354.

80. Ozguc, Ozgur (2021). 'The assessment of impact damage caused by dropped objects on floating offshore structures'. In: *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment* 235.2, pp. 491–510. DOI: 10.1177/1475090220972586. eprint: https://doi.org/10.1177/1475090220972586. URL: https://doi.org/10.1177/1475090220972586.

81. Papandreou, George et al. (2018). *PersonLab: Person Pose Estimation and Instance Segmentation with a Bottom-Up, Part-Based, Geometric Embedding Model.* DOI: 10.48550/ARXIV.1803.08225. URL: https://arxiv.org/abs/1803.08225.

82. Park, Kiru, Timothy Patten and Markus Vincze (Oct. 2019). 'Pix2Pose: Pixel-Wise Coordinate Regression of Objects for 6D Pose Estimation'. In: *2019 IEEE/CVF International Conference on Computer Vision (ICCV).* IEEE. DOI: 10.1109/iccv.2019.00776. URL: https://doi.org/10.1109%2Ficcv.2019.00776.

83. Peng, Sida et al. (2018). *PVNet: Pixel-wise Voting Network for 6DoF Pose Estimation.* eprint: 1812.11788. URL: https://arxiv.org/abs/1812.11788.

84. Pitteri, Giorgia et al. (2019). *On Object Symmetries and 6D Pose Estimation from Images.* DOI: 10.48550/ARXIV.1908.07640. URL: https://arxiv.org/abs/1908.07640.

85. Price, Leon C. et al. (2021). 'Multisensor-driven real-time crane monitoring system for blind lift operations: Lessons learned from a case study'. In: *Automation in Construction* 124, p. 103552. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2021.103552. URL: https://www.sciencedirect.com/science/article/pii/S0926580521000030.

86. Python (2022). Last accessed 07 June 2022. URL: https://www.python.org/doc/essays/comparisons/#:~:text=for%5C%20this%5C%20comparison.-,Java,types%5C%20and%5C%20its%5C%20dynamic%5C%20typing.

87. PyTorch (2022a). Last accessed 05 June 2022. URL: https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html.

88. — (2022b). Last accessed 05 June 2022. URL: https://pytorch.org/docs/stable/generated/torch.nn.HuberLoss.html.

89. — (2022c). Last accessed 05 June 2022. URL: https://pytorch.org/docs/stable/generated/torch.nn.SmoothL1Loss.html#:~:text=Creates%5C%20a%5C%20criterion%5C%20that%5C%20uses,sensitive%5C%20to%5C%20outliers%5C%20than%5C%20torch..

90. Rad, Mahdi and Vincent Lepetit (2017). 'BB8: A Scalable, Accurate, Robust to Partial Occlusion Method for Predicting the 3D Poses of Challenging Objects without Using Depth'. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3848–3856.

91. Raza, Ali (2019). *Type of convolutions: Deformable and Transformable Convolution.* Last accessed 8. May 2022. URL: https://towardsdatascience.com/type-of-convolutions-deformable-and-transformable-convolution-1f660571eb91.

92. Richter-Klug, Jesse and Udo Frese (2020). *Handling Object Symmetries in CNN-based Pose Estimation.* DOI: 10.48550/ARXIV.2011.13209. URL: https://arxiv.org/abs/2011.13209.

93. Rojas, Raúl (1996). 'Neural Networks'. In: Springer Berlin, Heidelberg. DOI: https://doi.org/10.1007/978-3-642-61068-4. URL: http://page.mi.fu-berlin.de/rojas/neural/chapter/K8.pdf.

94. Romero Aquino, Marcelo et al. (Oct. 2017). 'The Effect of Data Augmentation on the Performance of Convolutional Neural Networks'. In: DOI: 10.21528/CBIC2017-51.

95. Ronneberger, Olaf, Philipp Fischer and Thomas Brox (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation.* DOI: 10.48550/ARXIV.1505.04597. URL: https://arxiv.org/abs/1505.04597.

96. Sahin, Caner et al. (2020). 'A review on object pose recovery: From 3D bounding box detectors to full 6D pose estimators'. In: *Image and Vision Computing* 96, p. 103898. ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2020.103898. URL: https://www.sciencedirect.com/science/article/pii/S0262885620300305.

97. Saini, Vinod Kumar and Arnab Maity (2022). 'Random sample consensus in decentralized Kalman filter'. In: *European Journal of Control* 65, p. 100617. ISSN: 0947-3580. DOI: https://doi.org/10.1016/j.ejcon.2022.100617. URL: https://www.sciencedirect.com/science/article/pii/S0947358022000103.

98. Semeniuta, Oleksandr (2016). 'Analysis of Camera Calibration with Respect to Measurement Accuracy'. In: *Procedia CIRP* 41, pp. 765–770. ISSN: 2212-8271. DOI: https://doi.org/10.1016/j.procir.2015.12.108. URL: https://www.sciencedirect.com/science/article/pii/S2212827115011877.

99. Shahinfar, Saleh, Paul Meek and Greg Falzon (2020). '"How many images do I need?" Understanding how sample size per class affects deep learning model performance metrics for balanced designs in autonomous wildlife monitoring'. In: *Ecological Informatics* 57, p. 101085. ISSN: 1574-9541. DOI: https://doi.org/10.1016/j.ecoinf.2020.101085. URL: https://www.sciencedirect.com/science/article/pii/S1574954120300352.

100. Sharma, Siddharth, Simone Sharma and Anidhya Athaiya (2020). 'ACTIVATION FUNCTIONS IN NEURAL NETWORKS'. In: *International Journal of Engineering Applied Sciences and Technology* 4.2455-2143, pp. 310–316. URL: https://www.ijeast.com/papers/310-316,Tesma412,IJEAST.pdf.

101. SINTEF (2021). Last accessed 07 May 2022. URL: https://www.sintef.no/en/projects/2021/race-crane-operations-in-aquaculture/.

102. Själander, Magnus et al. (2019). *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure.* arXiv: 1912.05848 [cs.DC].

103. Smith, Philip and Cuixian Chen (Nov. 2018). *Transfer Learning with Deep CNNs for Gender Recognition and Age Estimation.*

104. Snyder, Hannah (2019). 'Literature review as a research methodology: An overview and guidelines'. In: *Journal of Business Research* 104, pp. 333–339. ISSN: 0148-2963. DOI: https://doi.org/10.1016/j.jbusres.2019.07.039. URL: https://www.sciencedirect.com/science/article/pii/S0148296319304564.

105. Sølund, Thomas (2017). *Towards Plug-n-Play robot guidance: Advanced 3D estimation and pose estimation in Robotic applications.* URL: https://backend.orbit.dtu.dk/ws/portalfiles/portal/138803120/phd424_Soelund_T.pdf.

106. Srivastava, Nitish et al. (2014). 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting'. In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

107. Stachniss, CYrill (2021). *Projective 3-Point (P3P) Algorithm / Spatial Resection.* Last accessed 07 June 2022. URL: http://www.ipb.uni-bonn.de/html/teaching/photo12-2021/2021-pho1-23-p3p.pptx.pdf.

108. Standley, Trevor et al. (13–18 Jul 2020). 'Which Tasks Should Be Learned Together in Multi-task Learning?' In: *Proceedings of the 37th International Conference on Machine Learning.* Ed. by Hal Daumé III and Aarti Singh. Vol. 119. Proceedings of Machine Learning Research. PMLR, pp. 9120–9132. URL: https://proceedings.mlr.press/v119/standley20a.html.

109. Store, Unity Asset (2021). *Container Boxes.* Last accessed 15 May 2022. URL: https://assetstore.unity.com/packages/3d/props/container-boxes-200551.

110. Stratospark (2019). *Unity Image Synthesis.* Last accessed 07 June 2022. URL: https://github.com/stratospark/UnityImageSynthesisTutorial1.

111. Sturm, Peter and Koichiro Deguchi (2021). 'Pinhole Camera Model'. In: *Computer Vision.* Springer International Publishing, pp. 983–986. ISBN: 3030634159.

112. Sunde, Leif Magne (2018). Last accessed 07 May 2022. URL: https://www.sintef.no/en/projects/2016/artifex/.

113. Tekin, Bugra, Sudipta N. Sinha and Pascal Fua (2017). *Real-Time Seamless Single Shot 6D Object Pose Prediction.* DOI: 10.48550/ARXIV.1711.08848. URL: https://arxiv.org/abs/1711.08848.

114. Tian, Keyu et al. (2020). 'Improving Auto-Augment via Augmentation-Wise Weight Sharing'. In: *Advances in Neural Information Processing Systems.* Ed. by H. Larochelle et al. Vol. 33. Curran Associates, Inc., pp. 19088–19098. URL: https://proceedings.neurips.cc/paper/2020/file/dc49dfebb0b00fd44aeff5c60cc1f825-Paper.pdf.

115. U3DC (2017). *Image Synthesis for Machine Learning.* Last accessed 07 June 2022. URL: https://github.com/U3DC/Image-Synthesis-for-Machine-Learning.

116. UCB (2018). Last accessed 07 May 2022. URL: http://uc-r.github.io/page2/.

117. Unity (2021a). *Render pipelines introduction.* Last accessed 15 May 2022. URL: https://docs.unity3d.com/Manual/render-pipelines-overview.html.

118. — (2021b). *Unity - Scripting API: Camera.ViewportToWorldPoint.* Last accessed 15 May 2022. URL: https://docs.unity3d.com/ScriptReference/Camera.ViewportToWorldPoint.htm.

119. — (2021c). *Unity Physical Cameras.* Last accessed 06 May 2022. URL: https://docs.unity3d.com/Manual/PhysicalCameras.html.

120. — (2021d). *Unity: Case study.* Last accessed 15 May 2022. URL: https://unity.com/case-study.

121. — (2021e). *Unity: Getting Started.* Last accessed 15 May 2022. URL: https://docs.unity3d.com/520/Documentation/Manual/GettingStarted.html.

122. — (2022). *Coding in C# in Unity for beginners.* Last accessed 15 May 2022. URL: https://unity.com/how-to/learning-c-sharp-unity-beginners.

123. Valtchev, Svetozar Zarko and Jianhong Wu (2021). 'Domain randomization for neural network classification'. In: *Journal of Big Data* 8.

124. Wang, He et al. (June 2019). 'Normalized Object Coordinate Space for Category-Level 6D Object Pose and Size Estimation'. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* DOI: 10.1109/CVPR.2019.00275.

125. Wang, Yikai et al. (2021). 'Channel Exchanging Networks for Multimodal and Multitask Dense Image Prediction'. In: *ArXiv* abs/2112.02252.

126. Weng, Lilian (2019). Last accessed 05 June 2022. URL: https://lilianweng.github.io/posts/2019-05-05-domain-randomization/.

127. Xiang, Yu et al. (2017). *PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes.* DOI: 10.48550/ARXIV.1711.00199. URL: https://arxiv.org/abs/1711.00199.

128. Xu, Junjie et al. (2020). 'An Improved Deep Keypoint Detection Network for Space Targets Pose Estimation'. In: *Remote Sensing* 12.23. ISSN: 2072-4292. DOI: 10.3390/rs12233857. URL: https://www.mdpi.com/2072-4292/12/23/3857.

129. Ying, Xue (Feb. 2019). 'An Overview of Overfitting and its Solutions'. In: *Journal of Physics: Conference Series* 1168, p. 022022. DOI: 10.1088/1742-6596/1168/2/022022. URL: https://doi.org/10.1088/1742-6596/1168/2/022022.

130. Yousoff, Siti Noorain Mohmad, Amirah Baharin and Afnizanfaizal Abdullah (2016). 'A review on optimization algorithm for deep learning method in bioinformatics field'. In: *2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES)*, pp. 707–711. DOI: 10.1109/IECBES.2016.7843542.

131. Yu, Xin et al. (2020). '6DoF Object Pose Estimation via Differentiable Proxy Voting Regularizer'. In: *BMVC*. eprint: 2002.03923. URL: https://www.semanticscholar.org/paper/6DoF-Object-Pose-Estimation-via-Differentiable-Yu-Zhuang/0b476357adc649b67d2d9682db574b68905ccb6b.

132. Zhang, Chiyuan et al. (2018). *A Study on Overfitting in Deep Reinforcement Learning.* DOI: 10.48550/ARXIV.1804.06893. URL: https://arxiv.org/abs/1804.06893.

133. Zhang, Guodong et al. (2018). *Three Mechanisms of Weight Decay Regularization.* DOI: 10.48550/ARXIV.1810.12281. URL: https://arxiv.org/abs/1810.12281.

134. Zhang, Haotian, Lin Zhang and Yuan Jiang (2019). 'Overfitting and Underfitting Analysis for Deep Learning Based End-to-end Communication Systems'. In: *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pp. 1–6. DOI: 10.1109/WCSP.2019.8927876.

135. Zhang, Zhendong, Xinran Wang and Cheolkon Jung (2019). 'DCSR: Dilated Convolutions for Single Image Super-Resolution'. In: *IEEE Transactions on Image Processing* 28, pp. 1625–1635.

136. Zhao, Xin et al. (2019). 'Use of Unmanned Aerial Vehicle Imagery and Deep Learning UNet to Extract Rice Lodging'. In: *Sensors* 19.18. ISSN: 1424-8220. DOI: 10.3390/s19183859. URL: https://www.mdpi.com/1424-8220/19/18/3859.

137. Zhou, Ying et al. (2021). 'Image-based onsite object recognition for automatic crane lifting tasks'. In: *Automation in Construction* 123, p. 103527. ISSN: 0926-5805. DOI: https://doi.org/10.1016/j.autcon.2020.103527. URL: https://www.sciencedirect.com/science/article/pii/S0926580520311079.

138. Zhuo, Wei et al. (Apr. 2018). '3D Box Proposals From a Single Monocular Image of an Indoor Scene'. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32.1. DOI: 10.1609/aaai.v32i1.12314. URL: https://ojs.aaai.org/index.php/AAAI/article/view/12314.

Appendix

```python
import torch
import torch.nn as nn
import torchvision.transforms.functional as TF


#Implementation of U-Net

class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(),
            nn.Dropout(p=0.2),
            nn.Conv2d(out_channels, out_channels, 3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(),
            nn.Dropout(p=0.2),
        )

    def forward(self, x):
        return self.conv(x)


class UNET(nn.Module):
    def __init__(
            self, in_channels=3, out_channels=1, features=[64, 128, 256, 512],
    ):
        super(UNET, self).__init__()
        self.ups = nn.ModuleList()
        self.downs = nn.ModuleList()
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)

        # Down part of UNET
        for feature in features:
            self.downs.append(DoubleConv(in_channels, feature))
            in_channels = feature

        # Up part of UNET
        for feature in reversed(features):
            self.ups.append(
                nn.ConvTranspose2d(
                    feature*2, feature, kernel_size=2, stride=2,
                )
            )
            self.ups.append(DoubleConv(feature*2, feature))

        self.bottleneck = DoubleConv(features[-1], features[-1]*2)
        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size=1)

    def forward(self, x):
        skip_connections = []

        for down in self.downs:
            x = down(x)
            skip_connections.append(x)
            x = self.pool(x)

        x = self.bottleneck(x)
        skip_connections = skip_connections[::-1]

        for idx in range(0, len(self.ups), 2):
            x = self.ups[idx](x)
            skip_connection = skip_connections[idx//2]

            if x.shape != skip_connection.shape:
                x = TF.resize(x, size=skip_connection.shape[2:])

            concat_skip = torch.cat((skip_connection, x), dim=1)
            x = self.ups[idx+1](concat_skip)

        return self.final_conv(x)
```
Listing 1: Source code for U-Net

```
import torch
import numpy as np
import math
import warnings
import torch.optim as optim
import torchvision as torchvision
import torch.nn as nn
import torchvision.transforms.functional as TF


#DVFnet for predicting keypoint vector fields

class DoubleDilatedConv(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleDilatedConv, self).__init__()
        self.DoubleDconv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=9,
                      stride=1, padding=1, dilation=2, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(),
            nn.Dropout(p=0.2),
            nn.Conv2d(out_channels, out_channels, kernel_size=9,
                      stride=1, padding=1, dilation=2, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(),
            nn.Dropout(p=0.2),
        )

    def forward(self, x):
        return self.DoubleDconv(x)


class DilatedConv(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DilatedConv, self).__init__()
        self.Dconv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=9,
                      stride=1, padding=1, dilation=2, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(),
            nn.Dropout(p=0.2),
        )

    def forward(self, x):
        return self.Dconv(x)


class Conv(torch.nn.Module):
    def __init__(self, in_channels, out_channels):
        super(Conv, self).__init__()
        self.Conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, kernel_size=3,
                      stride=1, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.LeakyReLU(),
            nn.Dropout(p=0.2),
        )

    def forward(self, x):
        return self.Conv(x)


class DVFnet(torch.nn.Module):
    def __init__(
        self, in_channels=3, out_channels=18, features=[64, 128, 256],
    ):
        super(DVFnet, self).__init__()
        self.downs = nn.ModuleList()
        self.ups = nn.ModuleList()
        self.pool = nn.MaxPool2d(kernel_size=1, stride=1)

        for feature in features:
            self.downs.append(Conv(in_channels, feature))
            in_channels = feature

        for feature in reversed(features):
            self.ups.append(
                nn.ConvTranspose2d(
                    feature*2, feature, kernel_size=2, stride=1,
                )
            )
            self.ups.append(Conv(feature*2, feature))

        self.bottleneck = DoubleDilatedConv(features[-1], features[-1]*2)
        self.final_conv = nn.Conv2d(features[0], out_channels, kernel_size=1)

    def forward(self, x):
        skip_connections = []

        for down in self.downs:
            x = down(x)
            skip_connections.append(x)
            x = self.pool(x)

        x = self.bottleneck(x)

        skip_connections = skip_connections[::-1]

        for idx in range(0, len(self.ups), 2):
            x = self.ups[idx](x)
            skip_connection = skip_connections[idx//2]

            if x.shape != skip_connection.shape:
                x = TF.resize(x, size=skip_connection.shape[2:])

            concat_skip = torch.cat((skip_connection, x), dim=1)
            x = self.ups[idx+1](concat_skip)
        return self.final_conv(x)
```

Listing 2: Source code for DVFnet

```python
import torch
import numpy as np
import math
import warnings

def crop_from_prediction(image, prediction, threshold=0.6):
    """
    Function for croping an images on the predicted masks. Crops the image at a threshold to
    eliminate outliers in the prediction.

    args:
        image: Tensor of images
        mask: Tensor of masks
        threshold: threshold for filtering weak predicitons (Default: 0.6)

    return:
        cropedImage: a croped image by using the predicted mask

    """
    old_height, old_width = image.shape[2], image.shape[3]
    for i in range(image.shape[0]):
        # Fetch coordinates of mask wiht a threshold
        coords = torch.where(prediction[i] >= threshold)[1:3]  # [x,y]

        # Setting top coordinate of the crop, the largest corner of the mask
        top_y = (min(coords[0]) - 20) if min(coords[0]
                                              ) > 10 else 0
        top_x = min(coords[1]) - 20 if min(coords[1]) > 10 else 0
        # setting the width and height accrording to the biggest corner of the mask
        new_height = max(coords[0])-top_y + 20 if max(coords[0]
                                                       )-top_y < old_height-20 else max(coords[0])-top_y + 20
        new_width = max(coords[1]) - top_x + 20 if max(coords[1]
                                                        ) - top_x < old_width - 21 else max(coords[1]) - top_x + 20

        # make sure the dimentions are even numbers for the neural network
        max_dim = max(new_height, new_width)

        if not max_dim % 2 == 0:
            max_dim += 1

        # Crop the image and mask on the given point and dimentions
        crop = TVF.crop(
            image[i], top_y, top_x, max_dim, max_dim)
        # unsqueeze to add batch dimention and append the cropped images and masks
        return torch.unsqueeze(crop, 0)
```

Listing 3: Source code for cropping an image based on its prediction