Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Mari Hovem Leonhardsen

Underwater Pose Estimation with Deep Learning

Master's thesis in Engineering and ICT
Supervisor: Ingrid Schjølberg
June 2019

NTNU
Norwegian University of
Science and Technology

Mari Hovem Leonhardsen

Underwater Pose Estimation with Deep Learning

Master's thesis in Engineering and ICT
Supervisor: Ingrid Schjølberg
June 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

NTNU Trondheim
Norwegian University of Science and Technology
Faculty of Engineering Science and Technology
Department of MarineTechnology

MASTER THESIS IN MARINE TECHNOLOGY

Spring 2019

FOR

Mari Hovem Leonhardsen

Underwater Pose Estimation with Deep Learning

Increased autonomy in underwater vehicle operations is desired to reduced cost and increase efficiency of intervention missions. Such missions require precise maneuvering of the vehicle, and accurate localization systems are therefore crucial to achieve an increased level of autonomy.

One of the key challenges in underwater operations is to detect and estimate the surrounding environment. This includes pose estimation of the vehicle relative to a fixed object, which is important for performing manipulation work on subsea installments. Deep learning methods have shown promising results for camera-based pose estimation, and will be investigated for underwater application in this thesis.

Objective
The master thesis aims to investigate the application of a deep learning method for underwater pose estimation.

The work should be carried out in steps as follows:
1. Perform a literature review on relevant topics for this thesis
    a. State-of-the-art methods for underwater pose estimation
    b. State-of-the-art methods for pose estimation with deep learning
2. Implement a deep learning architecture for pose estimation
3. Collect real-world underwater datasets in the MC-laboratory
4. Train and test the model on both simulated and real-world data
5. Discuss results and methodology
6. Draw the conclusions from the studies and discuss possible further research steps.

Supervisor	: Ingrid Schjølberg
Co-supervisor	: Mikkel Cornelius Nielsen
Submitted	: January 15th 2019
Deadline	: June 11th 2019

Ingrid Schjølberg
Supervisor
I. Schjølberg

Preface

This work is submitted as a master thesis at the Department of Marine Technology at the Norwegian University of Science and Technology (NTNU). The thesis is a part of my Master of Science (MSc) degree with a specialization in Marine Cybernetics. The work of this project thesis was carried out during the spring semester of 2019, and is a continuation of the project thesis written in the fall of 2018.

This thesis is motivated by the demand for accurate localization systems in order to increase the level of autonomy in subsea operations, and investigates the opportunity of applying deep learning techniques for camera-based localization of underwater vehicles.

As a result of this work, an abstract is submitted for the OCEANS 2019 Seattle conference, attached in Appendix A.

Trondheim, June, 2019

Mari Hovem Leonhardsen

Mari Hovem Leonhardsen

Acknowledgement

I want to thank my supervisor Professor Ingrid Schjølberg, for her assistance and encouragement during the work of this master thesis. A special thanks goes to my co-supervisor Postdoctoral Fellow Mikkel Cornelius Nielsen, for his continuous support and rewarding discussions, as well as participation in the MC-lab experiments. I would also like to thank Postdoctoral Fellow Albert Sans Muntadas for his assistance in the laboratory activities.

M.H.L

Abstract

This thesis investigates the opportunity of applying deep learning, particularly convolutional neural networks (CNNs), for camera-based localization of underwater vehicles.

Increased level of autonomy in underwater vehicle operations is of high interest to reduce the cost of intervention missions and increase the frequency of inspections. Performing manipulation tasks on subsea installments require extremely precise maneuvering of the vehicle, addressing the need for a high-precision localization system. A system for estimating the relative 3D position and attitude, together referred to as the six degrees of freedom (6-DoF) *pose*, between an underwater vehicle and an object of fixed position, is desired. State-of-the-art methods rely on computer vision (CV) to provide the necessary localization accuracy. However, traditional CV methods rely on having pre-installed artificial markers available on the subsea structures, which is undesired. Recent advances within CNNs have resulted in promising methods for pose estimation based on imagery input. Motivated by this, the goal of this thesis is to investigate the application of such CNN methods to estimate 6-DoF pose in an underwater environment, as an alternative to existing artificial marker-based methods.

The CNN architecture PoseNet is shown successful for pose estimation in terrestrial domains and was chosen for further investigation in this work. The network was implemented with the machine learning framework TensorFlow, and trained and tested with both simulated and real-world data. To accomplish this, underwater datasets of images labeled with 6-DoF pose were produced in the MC-laboratory at NTNU. This includes both datasets with images of an artificial marker, and datasets with images of a ring object imitating a subsea valve.

The results showed that the implemented model regresses underwater 6-DoF pose successfully, based on imagery input of the mock-up valve. Accuracy in the range of 19 mm and 0.4 degrees for position and orientation, respectively, is achieved. The results revealed that the implemented model, in fact, performs better on images with the valve model, than images of the artificial marker. The need for artificial markers is therefore absent with this method.

Sammendrag

Denne masteroppgaven undersøker om dyp læring kan anvendes for kamerabasert lokalisering av undervannsfartøy. En undergren innenfor dype nevrale nettverk, nemlig convolutional neural networks (CNN), er valgt for dette formålet.

Det rettes et stadig økt fokus på autonome egenskaper hos undervannsfartøy, for å redusere operasjonelle kostnader relatert til vedlikehold, inspeksjon og reparasjon av subsea strukturer. Et undervannsfartøy som manipulerer komponenter på en subsea struktur krever ekstremt nøayktig manøvrering, som adresserer behovet for svært presise lokaliseringssystemer. Det er i den anledning ønskelig å utvikle et system som estimerer relativ 3D posisjon og rotasjon, som tilsammen utgjør *pose* i seks frihetsgrader (6-DoF, eng: six degrees of freedom), mellom et undervannsfartøy og et fastmontert objekt. Eksisterende metoder er basert på datasyn (CV, eng: computer vision) for å oppnå nødvendig nøyaktighet. Slike tradisjonelle CV metoder er imidlertid avhengig av at forhåndsinstallerte kunstige markører er tilgjengelige på strukturen som skal undersøkes, som gjør dem uegnet for bruk i praksis. Nyere forskning innenfor CNN har resultert i lovende metoder for å estimere relativ pose ut fra bilder. Motivert av dette undersøker denne oppgaven muligheten for å anvende slike CNN-metoder for å estimere 6-DoF pose i et undervannsmiljø, som et alternativ til eksisterende metoder basert på kunstige markører.

CNN-arkitekturen PoseNet har vist seg vellykket for å estimere pose på land, og ble valgt for videre undersøkelse i dette arbeidet. Nettverket ble implementert ved hjelp av maskinlæringsrammeverket TensorFlow, før det ble trent og testet på både simulerte og virkelige datasett. Som en del av dette arbeidet ble det produsert datasett bestående av undervannsbilder merket med 6-DoF pose i MC-laboratoriet ved NTNU. Dette inkluderer både datasett med bilder av en kunstig markør, og datasett med bilder av et ringobjekt som modellerer en subsea ventil.

Resultatene viste at den foreslåtte modellen estimerer 6-DoF pose basert på bilder av den modellerte ventilen på en tilfredsstillende måte. Estimatene leverer nøyaktighet i området 19 mm og 0.4 grader for henholdsvis posisjon og rotasjon. Resultatene viste videre at den implementerte modellen yter bedre på bilder av den modellerte ventilen enn på bilder av den kunstig markøren. Behovet for kunstige markører er derfor fraværende med denne metoden.

Contents

1 Introduction **1**
- 1.1 Background and Motivation . . . 1
- 1.2 Objective . . . 3
- 1.3 Contributions . . . 4
- 1.4 Structure of Report . . . 4

2 Related Work **5**
- 2.1 Underwater Localization . . . 5
  - 2.1.1 Underwater Vision-Based Pose Estimation . . . 6
- 2.2 Vision-Based Pose Estimation . . . 7
  - 2.2.1 Traditional CV Methods . . . 8
  - 2.2.2 RGB-D Methods . . . 9
  - 2.2.3 Deep Learning Methods . . . 9
- 2.3 PoseNet and Extensions . . . 10
  - 2.3.1 Learning . . . 11
  - 2.3.2 Architecture . . . 12
  - 2.3.3 Transfer Learning . . . 13
  - 2.3.4 Extensions to PoseNet . . . 13

3 Theory on Deep Neural Networks **17**
- 3.1 General Neural Networks . . . 18
  - 3.1.1 Artificial Neuron . . . 19
  - 3.1.2 Activation Function . . . 19
- 3.2 Learning . . . 20
  - 3.2.1 Cost Function . . . 21
  - 3.2.2 Gradient Descent with Backpropagation . . . 21
  - 3.2.3 Overfitting . . . 25
  - 3.2.4 Hyperparameters . . . 26
- 3.3 Convolutional Neural Networks . . . 26
  - 3.3.1 The Convolutional Operator . . . 27

3.3.2 Convolutional Layer 29
3.3.3 Pooling Layer 30
3.3.4 Architecture 31
3.4 Recurrent Neural Networks 32
3.4.1 Architecture 34

4 Model 36
4.1 Learning 36
4.2 Architecture 37
4.3 Training Algorithm 38
4.4 Hyperparameters 38
4.5 Implementation Details 40

5 Simulated Data 41

6 Real-world Data 44
6.1 Laboratory Setup 44
6.1.1 ROV 45
6.1.2 Qualisys Motion Capture 45
6.2 Kinematics 46
6.2.1 Reference Frames 46
6.2.2 Transformations 49
6.3 Data Processing 50
6.4 Datasets 51

7 Performance 55

8 Iterations and Results 57
8.1 Process Outline 57
8.2 Results 60
8.2.1 Case A: Simulated Data with AruCo 61
8.2.2 Case B: Real-World Data with AruCo 66
8.2.3 Case C: Real-World Data with Ring Object (Random) 71
8.2.4 Case D: Real-World Data with Ring Object (Sequential) 76
8.2.5 Additional Results 84

9 Discussion 89
9.1 Overall Remarks 89
9.2 Datasets 90
9.2.1 Incorrect Pose Labels 90
9.2.2 Value Range of Pose Labels 91

9.2.3 Image Contents . . . 94
9.3 Model . . . 96
9.3.1 Architecture . . . 96
9.3.2 Hyperparameters . . . 97
9.4 Feasibility . . . 98

10 Conclusion 100

11 Further Work 101

Bibliography 103

A Abstract Paper I

B Complementary Theory IV

C Details of Datasets VII
C.1 Dataset 0 . . . VIII
C.2 Dataset 1 . . . IX
C.3 Dataset 2 . . . X
C.4 Dataset 3 . . . XI
C.5 Dataset 4 . . . XII
C.6 Dataset 5 . . . XIII
C.7 Dataset 6 . . . XIV

D Additional Results XV
D.1 Case 1 . . . XV
D.2 Case 2 . . . XVIII
D.3 Case 3 . . . XXI
D.4 Case 4 . . . XXVI
D.5 Case 5 . . . XXXI
D.6 Case 6 . . . XXXVI
D.7 Case 7 . . . XXXIX
D.8 Case 8 . . . XLIV
D.9 Case 9 . . . XLIX
D.10 Case 10 . . . LIV
D.11 Case 11 . . . LIX
D.12 Case 12 . . . LXIV
D.13 Case 13 . . . LXIX
D.14 Case 14 . . . LXXIV
D.15 Case 15 . . . LXXIX

List of Figures

1.1 Camera localization problem for ROV 2

2.1 Research topics of Chapter 2 5
2.2 PoseNet: tuning of scaling factor β 11
2.3 GoogLeNet architecture 12
2.4 PoseNet and BranchNet architectures 15

3.1 Structure of a multilayer neural network 18
3.2 Artificial neuron 19
3.3 Forward and backward pass during gradient descent 22
3.4 2D convolution operation 28
3.5 3D convolution operation 28
3.6 Vertical edge detection filter 29
3.7 Max pooling operation 30
3.8 Inception module 32
3.9 Data flow in RNNs 33
3.10 Simple RNN 34
3.11 RNN architecture 34

4.1 Architecture of implemented CNN 37
4.2 Tuning of scaling factor β 39

5.1 Simulation setup in Blender 42
5.2 Sample images from simulation 43

6.1 Setup in MC-lab 45
6.2 Reference frames tracked during experiments 48
6.3 Total reference frames in experiment 49
6.4 Target objects 52
6.5 AruCo marker on template 52
6.6 Sample images from real-world datasets 54

8.1 Outline of iterative assessment process 58
8.2 Case A: Sample image 61
8.3 Case A: Rotation and translation errors 62
8.4 Case A: Pose errors in each DoF (histograms) 64
8.5 Case A: Pose errors in each DoF (plots) 65
8.6 Case B: Sample image 66
8.7 Case B: Rotation and translation errors 67
8.8 Case B: Pose errors in each DoF (histograms) 69
8.9 Case B: Pose errors in each DoF (plots) 70
8.10 Case C: Sample image 71
8.11 Case C: Rotation and translation errors 72
8.12 Case C: Pose errors in each DoF (histograms) 74
8.13 Case C: Pose errors in each DoF (plots) 75
8.14 Case D: Sample image 76
8.15 Case D: Rotation and translation errors 77
8.16 Case D: Pose errors in each DoF (histograms) 79
8.17 Case D: Pose errors in each DoF (plots) 80
8.18 Case D: True and predicted orientation 82
8.19 Case D: True and predicted position 83

B.1 Step function IV
B.2 Linear function V
B.3 Sigmoid function V
B.4 Tanh function VI
B.5 ReLU function VI

C.1 Relative poses dataset 0 VIII
C.2 Relative poses dataset 1 IX
C.3 Relative poses dataset 2 X
C.4 Relative poses dataset 3 XI
C.5 Relative poses dataset 4 XII
C.6 Relative poses dataset 5 XIII
C.7 Relative poses dataset 6 XIV

D.1 Case 1: Rotation and translation errors XVI
D.2 Case 1: Pose errors in each DoF XVII
D.3 Case 2: Rotation and translation errors XIX
D.4 Case 2: Pose errors in each DoF XX
D.5 Case 3: Rotation and translation errors XXII
D.6 Case 3: Pose errors in each DoF XXIII
D.7 Case 3: True and predicted orientation XXIV

D.8 Case 3: True and predicted position XXV
D.9 Case 4: Rotation and translation errors XXVI
D.10 Case 4: Pose errors in each DoF XXVIII
D.11 Case 4: True and predicted orientation XXIX
D.12 Case 4: True and predicted position XXX
D.13 Case 5: Rotation and translation errors XXXI
D.14 Case 5: Pose errors in each DoF XXXIII
D.15 Case 5: True and predicted orientation XXXIV
D.16 Case 5: True and predicted position XXXV
D.17 Case 6: Rotation and translation errors XXXVI
D.18 Case 6: Pose errors in each DoF XXXVIII
D.19 Case 7: Rotation and translation errors XXXIX
D.20 Case 7: Pose errors in each DoF XLI
D.21 Case 7: True and predicted orientation XLII
D.22 Case 7: True and predicted position XLIII
D.23 Case 8: Rotation and translation errors XLIV
D.24 Case 8: Pose errors in each DoF XLVI
D.25 Case 8: True and predicted orientation XLVII
D.26 Case 8: True and predicted position XLVIII
D.27 Case 9: Rotation and translation errors XLIX
D.28 Case 9: Pose errors in each DoF LI
D.29 Case 9: True and predicted position LII
D.30 Case 9: True and predicted position LIII
D.31 Case 10: Rotation and translation errors LIV
D.32 Case 10: Pose errors in each DoF LVI
D.33 Case 10: True and predicted orientation LVII
D.34 Case 10: True and predicted position LVIII
D.35 Case 11: Rotation and translation errors LIX
D.36 Case 11: Pose errors in each DoF LXI
D.37 Case 11: True and predicted orientation LXII
D.38 Case 11: True and predicted position LXIII
D.39 Case 12: Rotation and translation errors LXIV
D.40 Case 12: Pose errors in each DoF LXVI
D.41 Case 12: True and predicted orientation LXVII
D.42 Case 12: True and predicted position LXVIII
D.43 Case 13: Rotation and translation errors LXIX
D.44 Case 13: Pose errors in each DoF LXXI
D.45 Case 13: True and predicted orientation LXXII
D.46 Case 13: True and predicted position LXXIII
D.47 Case 14: Rotation and translation errors LXXIV

D.48 Case 14: Pose errors in each DoF LXXVI
D.49 Case 14: True and predicted orientation LXXVII
D.50 Case 14: True and predicted position LXXVIII
D.51 Case 15: Rotation and translation errors LXXIX
D.52 Case 15: Pose errors in each DoF LXXXI
D.53 Case 15: True and predicted orientation in roll, pitch and yaw LXXXII
D.54 Case 15: True and predicted position LXXXIII

List of Tables

4.1 Configuration of hyperparameters 40

6.1 Properties of simulation and real-world datasets 53

8.1 Case A: Details of train and test datasets 61
8.2 Case A: Summary statistics of pose errors 62
8.3 Case A: Summary statistics of pose errors in each DoF 63
8.4 Case A: Amount of test data satisfying the 10-percent requirements 66
8.5 Case B: Details of train and test datasets 66
8.6 Case B: Summary statistics of pose errors 67
8.7 Case B: Summary statistics of pose errors in each DoF 68
8.8 Case B: Amount of test data satisfying the 10-percent requirements 71
8.9 Case C: Details of train and test datasets 71
8.10 Case C: Summary statistics of pose errors 72
8.11 Case C: Summary statistics of pose errors in each DoF 73
8.12 Case C: Amount of test data satisfying the 10-percent requirements 76
8.13 Case D: Details of train and test datasets 76
8.14 Case D: Summary statistics of pose errors 77
8.15 Case D: Summary statistics of pose errors in each DoF 78
8.16 Case D: Amount of test data satisfying the 10-percent requirements 81
8.17 Results from cases with different relative distances 84
8.18 Results from cases with different light conditions 85
8.19 Results from cases with different background and different target object . . 86
8.20 Results from cases with random and sequential split 86
8.21 Results from cases transferring knowledge between different domains 87

D.1 Case 1: Details of train and test datasets XV
D.2 Case 1: Summary statistics of pose errors XV
D.3 Case 1: Summary statistics of pose errors in each DoF XVI
D.4 Case 1: Amount of test data satisfying the 10-percent requirements XVIII
D.5 Case 2: Details of train and test datasets XVIII

D.6 Case 2: Summary statistics of pose errors XVIII
D.7 Case 2: Summary statistics of pose errors in each DoF XIX
D.8 Case 2: Amount of test data satisfying the 10-percent requirements XXI
D.9 Case 3: Details of train and test datasets XXI
D.10 Case 3: Summary statistics of pose errors XXI
D.11 Case 3: Summary statistics of pose errors in each DoF XXII
D.12 Case 4: Details of train and test datasets XXVI
D.13 Case 4: Summary statistics of pose errors XXVI
D.14 Case 4: Summary statistics of pose errors in each DoF XXVII
D.15 Case 5: Details of train and test datasets XXXI
D.16 Case 5: Summary statistics of pose errors XXXI
D.17 Case 5: Summary statistics of pose errors in each DoF XXXII
D.18 Case 6: Details of train and test datasets XXXVI
D.19 Case 6: Summary statistics of pose errors XXXVI
D.20 Case 6: Summary statistics of pose errors in each DoF XXXVII
D.21 Case 7: Details of train and test datasets XXXIX
D.22 Case 7: Summary statistics of pose errors XXXIX
D.23 Case 7: Summary statistics of pose errors in each DoF XL
D.24 Case 8: Details of train and test datasets XLIV
D.25 Case 8: Summary statistics of pose errors XLIV
D.26 Case 8: Summary statistics of pose errors in each DoF XLV
D.27 Case 9: Details of train and test datasets XLIX
D.28 Case 9: Summary statistics of pose errors XLIX
D.29 Case 9: Summary statistics of pose errors in each DoF L
D.30 Case 10: Details of train and test datasets LIV
D.31 Case 10: Summary statistics of pose errors LIV
D.32 Case 10: Summary statistics of pose errors in each DoF LV
D.33 Case 11: Details of train and test datasets LIX
D.34 Case 11: Summary statistics of pose errors LIX
D.35 Case 11: Summary statistics of pose errors in each DoF LX
D.36 Case 12: Details of train and test datasets LXIV
D.37 Case 12: Summary statistics of pose errors LXIV
D.38 Case 12: Summary statistics of pose errors in each DoF LXV
D.39 Case 13: Details of train and test datasets LXIX
D.40 Case 13: Summary statistics of pose errors LXIX
D.41 Case 13: Summary statistics of pose errors in each DoF LXX
D.42 Case 14: Details of train and test datasets LXXIV
D.43 Case 14: Summary statistics of pose errors LXXIV
D.44 Case 14: Summary statistics of pose errors in each DoF LXXV
D.45 Case 15: Details of train and test datasets LXXIX

D.46 Case 15: Summary statistics of pose errors LXXIX
D.47 Case 15: Summary statistics of pose errors in each DoF LXXX

Acronyms

6-DoF six degrees of freedom.

Adam adaptive moment estimation.

ANN artificial neural network.

AUV autonomous underwater vehicle.

CNN convolutional neural network.

CV computer vision.

DNN deep neural network.

DoF degree of freedom.

DVL doppler velocity logger.

EKF extended Kalman filter.

FC fylly-connected.

GAN generative adversarial network.

GPU graphics processing unit.

HDF hierarchical data format.

HSE health, safety and environment.

IMR inspection, maintenance and repair.

IMU inertial measurement unit.

LBL long baseline.

LSTM long-short term memory.

MC marine cybernetics.

ML machine learning.

MSE mean squared error.

QTM qualisys track manager.

R-CNN regional convolutional neural network.

ReLU rectified linear unit.

RGB red, green and blue.

RGB-D RGB-depth.

RNN recurrent neural network.

ROS robotic operator system.

ROV remotely operated vehicle.

SGD stochastic gradient descent.

SLAM simultaneously localization and mapping.

USBL ultra short base line.

Chapter 1

Introduction

1.1 Background and Motivation

The demand for subsea inspection, maintenance and repair (IMR) operations is expected to increase in the coming years, of two reasons. First, the Norwegian continental shelf hosts more than 5000 subsea wells whose equipment is maturing. Second, the introduction of subsea processing of oil and gas is expected within the next few years. Such factories feature complex systems, contributing to the increased demands for IMR operations [1].

Remotely operated vehicles (ROV) are crucial for the execution of IMR operations. They are mostly manually operated by human operators from an offshore vessel, with a low level of autonomy. This is related to a high cost, and the operations rely on both human performance and weather conditions. Increased autonomy of IMR operations may introduce improvements in health, safety and environment (HSE), in addition to reduce costs and improve performance [2]. While fully autonomous operations are not necessarily a goal in itself, an increased level of autonomy from what is seen today is desired. Schjølberg et al. [1] suggest shared control, a control regime where certain modes are performed autonomously, and others are performed by the operator. The human operator remains in the loop and can interrupt any actions initiated by the autonomous system. This control configuration will reduce the workload on the operator, reduce errors in operations and increase efficiency, all contributing to cost reductions.

To increase the level of autonomy in ROV operations, localization systems are essential. Localization refers to an object's understanding of its position and attitude, relative to the surroundings [1]. 3D position and attitude are together referred to as the six degrees of freedom (6-DoF) *pose*. Performing manipulation tasks on subsea installments require extremely precise maneuvering of the ROV, to avoid collisions and damages on subsea equipment. Such failures can lead to huge expenses, but even worse, they can cause leakages of petroleum to the surrounding environment. Therefore, highly accurate localization systems are essential for an increased level of autonomy in IMR operations. This leads to the demand for a real-time algorithm for frequent 6-DoF localization with high accuracy.

There are different approaches to the localization problem. Map-based localization is possible in static surroundings if a precise map is available. This is seldom the case for underwater environments, and an alternative approach is to localize relative to a particular coordinate or a particular object. This approach imposes prior knowledge of the structure where the ROV is operating, as one or more reference points for relative localization is necessary. A third approach to the localization problem is simultaneously localization and mapping (SLAM), which simultaneously localizes a vehicle with respect to a map while updating the map on the go [3]. This thesis will focus on localization of underwater vehicles relative to a particular object in the environment.

State-of-the-art methods for estimation of the relative pose between an underwater vehicle and a fixed point in the environment rely on computer vision (CV) to provide the necessary localization accuracy. Most underwater vehicles are equipped with cameras, and due to improvements within both software and hardware in recent years, there has been an increased interest in underwater applications of CV methods. When estimating the relative pose between an underwater vehicle and some fixed reference point based on imagery input, the problem is reduced to estimate the relative pose between the camera on the vehicle and the reference point of fixed position. This corresponds to the estimation of the relative 6-DoF pose vector, denoted $\boldsymbol{\eta}_{rel}$, illustrated in Figure 1.1. This is referred to as the *camera localization* problem, or the *camera pose estimation* problem.

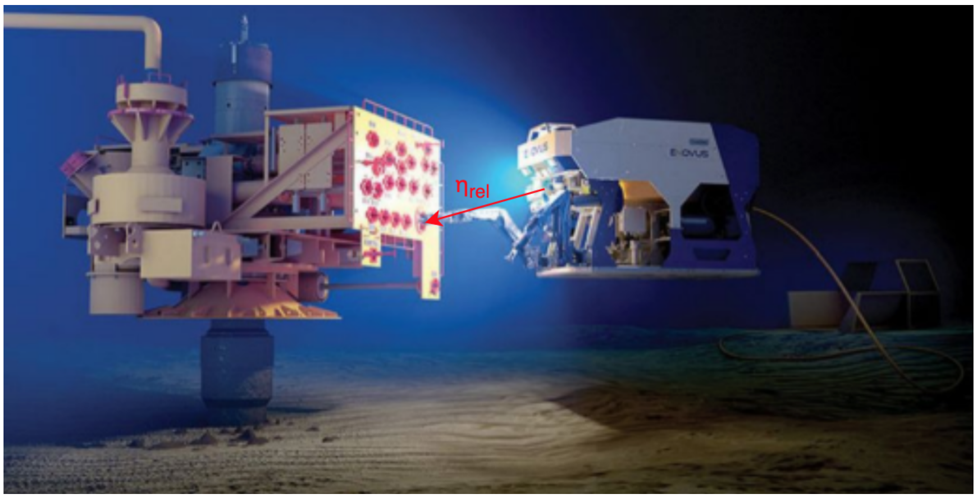


Figure 1.1: Camera localization problem for ROV: estimation of the relative pose vector, $\boldsymbol{\eta_{rel}}$, between the camera and the fixed reference point (reproduced from [4])

There are several examples of previous work applying traditional CV methods to estimate 6-DoF relative pose underwater [5],[6],[3],[7],[8]. However, these methods rely on having pre-installed artificial markers available on the subsea structure. The current generation of deployed subsea infrastructure does not have such markers available, rendering the CV methods infeasible to use in practice.

Recent advances within the research field of image processing with applications of deep learning techniques have resulted in promising methods for pose estimation. Such methods use the deep learning architecture convolutional neural networks (CNNs) to regress the 6-DoF pose from one single image, and they do not require artificial landmarks in the environment [9],[10],[11]. To the author's best knowledge, such methods have not yet been applied to the underwater camera localization problem.

The problem addressed in this thesis is to solve the underwater camera pose estimation problem without artificial markers.

1.2 Objective

The work of this thesis is motivated by the increasing need for high-precision underwater localization systems, and the developments of deep learning methods for camera-based pose estimation. This work will investigate the opportunity of such methods to be applied for underwater pose estimation in order to eliminate the need for artificial markers. A preliminary study for the work of this thesis was carried out in the project thesis [12]. A literature review on pose estimation with deep learning was performed, and the PoseNet model [9] is from this chosen for further investigation in this thesis.

The objective of this thesis is to further investigate and develop a deep learning method for solving the 6-DoF camera localization problem underwater. To achieve this, the tasks listed below have been identified:

1. Review related work on underwater localization, and pose estimation with deep learning.
2. Describe and research the theory of deep neural networks, in particular, CNNs, relevant for the model to be implemented. Research other deep learning techniques relevant for improvements of the model.
3. Implement a CNN model for pose regression.
4. Produce real-world datasets of underwater images labeled with 6-DoF relative poses from laboratory work.

5. Train, test and tune the suggested model on different datasets. These include simulated data of an AruCo marker, real-world data of an AruCo marker, and real-world data of an object modeling a subsea valve.

1.3 Contributions

The main contribution of this work was to implement and assess the performance of a deep learning algorithm for the underwater camera localization problem. In particular, the CNN architecture PoseNet was suggested to regress the relative 6-DoF pose between an underwater vehicle and a fixed object based on imagery input. As a part of this work, underwater datasets containing images labeled with 6-DoF relative pose were produced.

An abstract submitted for the OCEANS 2019 Seattle conference, attached in Appendix A, was an additional contribution of this thesis.

1.4 Structure of Report

The rest of this report is structured as follows.

Chapter 2 presents a literature review on underwater localization, and 6-DoF pose estimation with deep learning. The PoseNet model is dedicated its own section in this chapter.

Chapter 3 presents theory on deep neural networks, in particular, CNNs and recurrent neural networks. This chapter lays a theoretical foundation for the implemented model.

Chapter 4 describes the implemented model.

Chapter 5 describes the simulated dataset.

Chapter 6 describes the procedure for producing the real-world dataset in the MC-laboratory, the necessary data processing and the specifics of the resulting datasets.

Chapter 7 describes the metrics used to assess the performance of the model.

Chapter 8 describes the iterations in which the train and test scenarios were carried out, and presents the results.

Chapter 9 discusses the results and the method for obtaining them.

Chapter 10 concludes on the work of this thesis.

Chapter 11 presents suggestions for further work on the topic of this thesis.

Chapter 2

Related Work

This chapter presents previous work related to the topic of this thesis, and covers the two research topics underwater localization and vision based pose estimation. Related work on underwater localization is presented in Section 2.1 and work on vision-based pose estimation, including deep learning methods, is presented in Section 2.2. The particular deep learning model PoseNet is dedicated its own section in Section 2.3, as the methodology in this thesis is heavily based on this model. To clarify, Figure 2.1 shows the relation between the research topics of this chapter. Parts of this chapter were written as a part of the project thesis [12], subject to changes. The theory supporting the contents of this chapter is presented in Chapter 3.

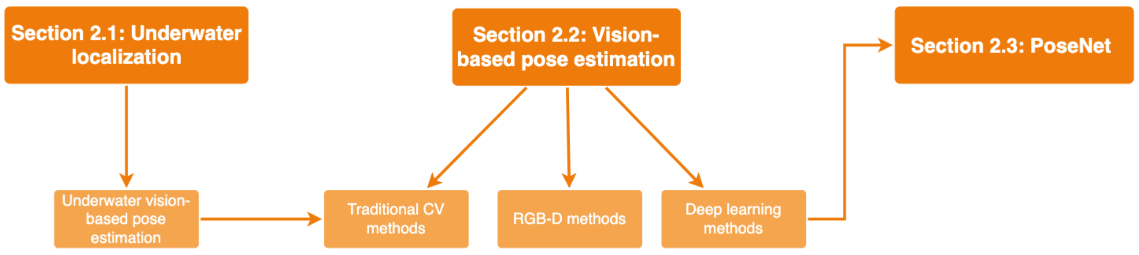

Section 2.1: Underwater localization
Section 2.2: Vision-based pose estimation
Section 2.3: PoseNet
Underwater vision-based pose estimation
Traditional CV methods
RGB-D methods
Deep learning methods

Figure 2.1: Relation between the research topics presented in Chapter 2

2.1 Underwater Localization

The underwater environment makes several challenges for localization, such as the absence of GPS signals. To cope with this, an ROV is normally equipped with several localization sensors such as inertial measurement units (IMU), compasses, doppler velocity logs

(DVL) and acoustic sensors. The industry standard for underwater localization is to use acoustic signals [6]. Acoustic localization employs hardware such as transducers installed on the seabed, to generate range measurements from acoustic signals. Ultra short base line (USBL) and long baseline (LBL) are two commonly used systems that feature such range measurements. These localization systems have a large range and are typically used for surveys, transits and coarse positioning. Due to low accuracy, low frequency and high latency, these systems fail to deliver the localization needed for the performance of manipulation tasks related to IMR. It is therefore suggested to use a complementary vision-based localization system. Compared to acoustic systems, vision-based localization serves shorter range, but increased accuracy and frequency, which is necessary when performing precise manipulation work.

2.1.1 Underwater Vision-Based Pose Estimation

A majority of the underwater vehicles of today are equipped with cameras. Because of the huge improvements within both software and hardware in recent years, there has been an increased interest in underwater application of CV methods [6]. The camera localization problem is therefore highly relevant for underwater purposes.

There are several examples of vision-based pose estimation for underwater application presented in literature, some of them discussed in the following. These studies have in common that they apply traditional CV methods with feature matching methods. Such methods compare images of a known landmark with an a priori known template of the landmark, and from this calculate the pose between the camera and the landmark. Such landmarks can either be artificial markers, or they can be real landmarks such as a component on a subsea installment. Traditional CV methods are further described in Section 2.2.1.

Palomeras et al. [3] suggested a SLAM system for autonomous underwater vehicles (AUVs) performing subsea intervention work. Their suggested algorithm fuses signals from several localization systems with an extended Kalman filter (EKF), where two visual position estimation algorithms make up one of these localization systems. Both of these algorithms are based on traditional CV feature matching methods. The first one uses a subsea panel as the landmark for feature matching, while the second algorithm uses the artificial marker system ARToolkit [13]. Their work concludes that the suggested vision-based localization component does improve the fused SLAM system. This work does however not take the orientation of the landmarks into consideration, only the position.

Henriksen et al. [5] conducted a study on vision-based localization of an underwater vehicle performing manipulation tasks on a subsea installment. This study was conducted with the artificial marker system AprilTags [14]. Their system was suggested as a complementary vision-based localization system, delivering pose estimates of an artificial marker relative to the camera on the ROV. The feasibility of the system was verified by using these pose estimates in a feedback loop for a pose controller.

Another suggestion for a precise vision-based localization system intended for underwater manipulation was presented by Chavez et al. [7]. This study was conducted with the artificial marker system AruCo [15], used to obtain the pose of a subsea panel relative to an ROV. Similar to the model in [3], the pose estimates are fed to an EKF to obtain the ROV pose over time. The obtained relative pose estimates are significantly more accurate in a noise-free environment compared to an underwater environment, demonstrating the difficulty of the problem addressed in this thesis. However, the achieved errors of the pose estimates were small enough for the EKF to deliver sufficient navigation performance.

It is emphasized that these studies differ slightly in terms of their end goals. Similar to the objective of this thesis, Henriksen et al. [5] developed a system for directly estimating the relative pose between an object and the ROV. In difference, both Chavez et al. [7] and Palomeras et al.[3] use these pose estimates one step further, by implementing them in sensor fusion-based ROV localization systems with application of EKF. However, these studies have in common that they all rely on artificial landmarks. This is undesired, as the current generation of deployed subsea infrastructure does not have such markers available.

To the author's best knowledge, no deep learning methods have been applied for underwater vision-based pose estimation.

2.2 Vision-Based Pose Estimation

The problem of determining exact localization is crucial in several robotic applications, such as object manipulation and learning from demonstration. Existing work on 6-DoF pose estimation based on imagery input includes traditional CV methods, RGB-depth (RGB-D) methods and more recent deep learning methods. A selection of this work is presented in the following.

When reviewing the work on pose estimation in literature, one should be aware of the distinction between camera pose estimation and object pose estimation. Camera pose estimation, also referred to as camera localization, is the problem addressed in this thesis. This regards predicting the pose of a fixed point in the scenery, typically placed on an object of interest, relative to the camera. Object pose estimation, on the other hand, requires the additional steps of detecting and classifying the object of interest, before predicting the relative pose between the object and the camera. The object pose estimation problem may deal with several objects at the same time. One can therefore argue that the camera pose estimation task is easier than the object pose estimation task [11]. The problems are highly coupled, and can together be generalized as the pose estimation problem. Existing work on both problems is reviewed in this section.

2.2.1 Traditional CV Methods

Traditional CV approaches for the pose estimation problem are typically feature-based, and consist of two steps. First, one or more features of interests in the image are detected, and the positions of the features in the image are estimated. Second, these 2D image points are translated to a 6-DoF camera pose by application of geometry theory. The second step is referred to as perspective-n-point, which is the problem of calculating the pose of a camera given a number of 3D points from the world and the corresponding 2D projection in the image. This approach, therefore, requires a set of pixel points with a known location on the object, to be matched with corresponding points on the camera image. These points can be natural features, extracted with methods such as SIFT [16] or ORB [17], or they can be points on artificial landmarks [18], e.g., a QR code. There are pros and cons related to both natural and artificial landmarks. Using natural landmarks is considered beneficial as the objects in the environment subject to pose estimation serve as reference points, and no intervention for inserting artificial markers are needed. The feature matching methods for natural landmarks do however require textured objects in high-resolution images, which is especially challenging in underwater sceneries. Artificial landmarks, on the other hand, are often easier to detect, especially when the camera is moving fast or the sight is blurry. Since IMR operations are performed under difficult underwater conditions, artificial markers are necessitated for traditional CV methods, as seen in the related work presented in Section 2.1.1.

Artificial marker systems

There exist several different artificial marker systems. Such a system is defined by a library of markers, and an algorithm to detect and estimate the 2D position of certain pixel points on the marker in the camera image. This makes up the first step of the twofold feature-based approach for camera localization. Planar quadratic marker systems such as QR codes are popular, and according to Henriksen [6] they are well suited for pose estimation. They are easy to generate, and only one marker is needed to estimate relative pose. The quadratic markers can be assigned a unique ID, which is used both to separate objects placed close to each other and to verify correct detection of objects. The four corners of the markers are used to find the correspondences between points with a known location in the environment and the projection on the camera image. ARToolKit

[13], ArUco [15] and AprilTags [14] are three quadratic marker systems that are commonly used.

2.2.2 RGB-D Methods

The development of cameras providing both color and depth images, so-called RGB-D images, has led to work suggesting different RGB-D methods for pose estimation. First, RGB-D images have been used with feature matching methods [19],[20]. Such methods have been shown to be more robust to difficult lighting conditions than the conventional feature-matching methods described in Section 2.2.1, and they handle texture less natural landmarks better. However, according to Xiang et al. [10], the performance of such methods is significantly reduced by occlusions. Another RGB-D approach is to map the pixels of a depth image to global coordinates [21],[22],[23]. This mapping can then be used to train a regressor to regress the global coordinates of the pixels in an image to predict the pose of the camera. Available depth sensors are however huge power consumers, making these RGB-D methods infeasible to implement on underwater vehicles.

2.2.3 Deep Learning Methods

Deep learning techniques for vision-based pose estimation has been a hot topic of research in recent years. Suggested methods are based on CNNs, a particular type of neural networks operating on multidimensional data, shown to be extremely successful in computer vision applications [24]. Theory governing CNNs is presented in Section 3.3.

Classification Problem

Neural networks can be constructed for either classification or regression problems. CNNs are mostly used for classification problems, and it is therefore suggested to cast the pose estimation problem as a classification problem [25],[26]. This requires discretizing the pose space, dividing it into a number of possible classes, which in turn limits the possible size of the pose domain and/or the possible accuracy to obtain with the method.

Regression Problem

While CNNs are mostly used for classification purposes, they can also be applied to regression problems. Since the pose domain is indeed continuous, one can argue that the pose estimation problem is more suited as a regression problem than a classification problem.

Recently there has been suggested several systems for solving the camera pose estimation problem with CNNs [9],[10],[11]. Kendall et al. were the first to introduce such a method with the PoseNet in 2015 [9]. This model directly regresses the 6-DoF camera pose from an RGB image with a CNN. Translation and orientation are regressed simultaneously together in this model. Related to this work, several extensions and modifications of the PoseNet have been suggested [27],[28],[29]. Since the work in this thesis is heavily based on the PoseNet model, a more detailed review on PoseNet and its extensions is dedicated its own section, see Section 2.3.

CNN regressors have also been suggested applied for the object pose estimation problem. PoseCNN [10] and Deep-6DPose [11] are two such architectures, regressing 6-DoF object pose directly from RGB images. Both methods detect and predict the 2D coordinates of the object of interest, and based on these 2D coordinates regress the 6-DoF object pose. In difference from PoseNet, both of these methods decouples the translation and orientation predictors.

2-step: CNN Combined With Perspective-n-point

As an alternative to train a CNN to regress the 6-DoF pose, other work in literature suggest to split the pose estimation into two steps, and use CNN only for the first step [30],[31]. In these methods, a CNN is trained to detect an object in the image and predict the 2D location of the object bounding box in the image. These 2D points are then used to obtain the 6-DoF object pose with a perspective-n-point algorithm. This two-step pipeline, therefore, combines a deep learning approach with a classical CV approach.

2.3 PoseNet and Extensions

As mentioned in the previous section, Alex Kendall et al. introduced the PoseNet [9] in 2015, which is a CNN for real-time 6-DoF camera relocalization. In their work, they trained a network to regress the 6-DoF pose of a camera relative to a scene, from only one RGB image. PoseNet is thus a regression network. The algorithm calculates the pose given an image in only 5 ms, making it applicable for real-time usage. The specifics of the PoseNet model is described in the following, in particular, the learning configuration in Section 2.3.1 and the architecture in Section 2.3.2. The application of transfer learning with PoseNet is described in Section 2.3.3. The contents of this section are based on [9].

2.3.1 Learning

The convolutional neural network outputs a pose vector, $\boldsymbol{p}$, of 7 entries, containing the 3D camera position, $\boldsymbol{x}$, and the orientation represented by the quaternion, $\boldsymbol{q}$. The notation of this section follows the PoseNet paper, differing slightly from the one used in the remainder of this thesis where $\boldsymbol{\eta}$ denotes the pose vector, $\boldsymbol{p}$ denotes the position vector, and $\boldsymbol{q}$ remains the rotation vector. Quaternions were chosen to represent orientation since such 4D values can be converted to unique rotations when normalizing them to unit length, avoiding the periodicity problem related to Euler angles. The CNN was trained with stochastic gradient descent (see Section 3.2.2) with a modified mean squared error loss function (see Section 3.2.1) defined in Equation 2.1.

$$L = ||\hat{\boldsymbol{x}} - \boldsymbol{x}||_2 + \beta \left|\left| \hat{\boldsymbol{q}} - \frac{\boldsymbol{q}}{||\boldsymbol{q}||} \right|\right|_2 \tag{2.1}$$

In Equation 2.1, notation $\hat{(\cdot)}$ denotes an estimate of a variable, such that $\boldsymbol{x}$, $\hat{\boldsymbol{x}}$, $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$ represents the true and predicted position and orientation, respectively. β represents a scaling factor that is used to keep the expected value of the position error and the quaternion error in the same value range. An optimal value for β was found after training with grid search, from the ratio between the mean error for position and the mean error for orientation. During the work of PoseNet, β turned out to vary for outdoor and indoor scenes, as the position error was greater for outdoor scenes. The value for β was tuned with grid search, where the relative error of position and orientation were plotted for a range of scaling factors for a given training set. Figure 2.2 illustrates this grid search, performed on one of the scenes from the 7 scenes dataset. In this case, an optimal β can be read to be approximately 350.

The network regresses position and orientation simultaneously as one entity, rather than either regressing them completely separately, or branching the network into two components regressing position and orientation separately. The authors of PoseNet argued that such a separation denies information between the two variables that are in fact coupled, hence the combined pose regression.

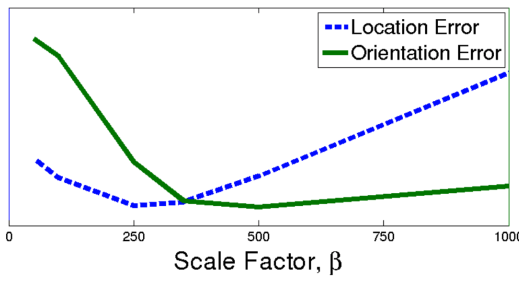


Figure 2.2: Grid search for tuning of the scaling factor β on the 7 scenes dataset [9]

2.3.2 Architecture

The PoseNet network is 23 layers deep, and its architecture is based on the classification network GoogLeNet [32]. GoogLeNet is a state of the art network architecture for classification purposes, having nine inception modules (see Section 3.3.4) and two extra classifiers inserted between the layers of the network. These intermediate classifiers are used during training, for additional assessments of the loss, and are discarded at test time.

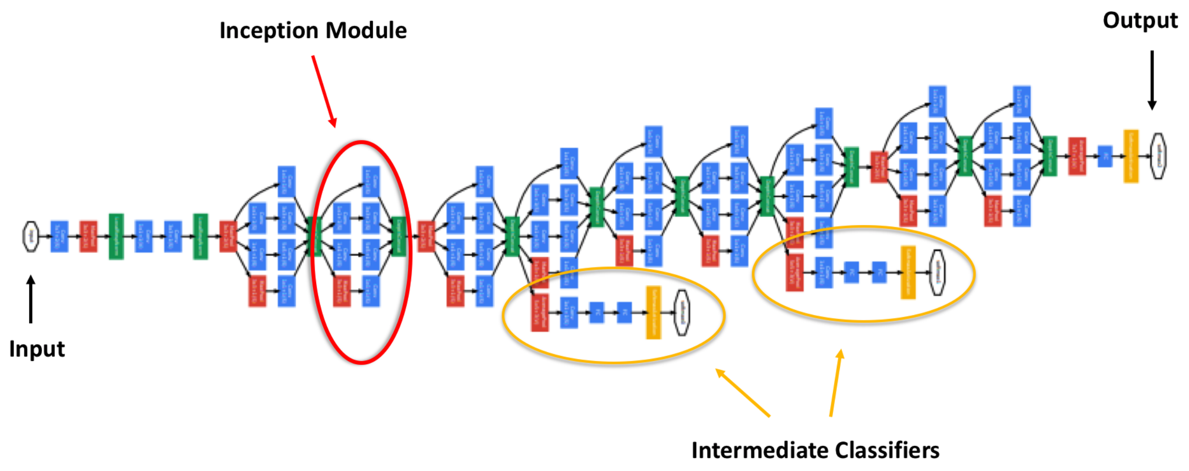


Figure 2.3: Architecture of GoogLeNet (reproduced from [32])

The architecture of GoogLeNet is illustrated in Figure 2.3, showing the additional intermediate classifiers in yellow circles and the structure of an inception module in the red circle. Blue boxes represents convolutional and fully connected (FC) layers, each including the rectified linear unit (ReLU) activation. Red boxes represents pooling layers, yellow boxes represents softmax activation and green boxes represents concatenating layers, that concatenates the feature maps from the parallel layers in the inception modules.

As GoogLeNet is a classifier, it had to be slightly modified for the regression task of PoseNet. The softmax classifiers of GoogLeNet are replaced with linear functions as regressors. Also, one extra fully connected layer is added at the very end before the final regressor. The quaternion vector $\boldsymbol{q}$ is normalized to unit length before returned as a component of the final pose vector $\boldsymbol{p}$.

2.3.3 Transfer Learning

A great challenge with regression problems in neural networks is the need for enormous training datasets, that can be costly to produce. Classification problems normally have several training examples for every possible category, which may be realistic as the output domain is both finite and discrete. For regression problems, the output domain is infinite and continuous, and it is therefore impossible to have training examples for every possible outcome. Regardless, to cover most of the output domain during training, a huge training dataset is needed for regression networks. During the work of developing the PoseNet architecture, it was demonstrated how transfer learning could be used to overcome this problem. Transfer learning refers to the reuse of network parameters learned for solving one specific problem when solving a different problem. That is, knowledge obtained from one problem is transferred to another problem. It has previously been shown that CNN representations that are trained on specific classification problems can be generalized to solve other classification problems. The researchers behind PoseNet showed that CNN parameters obtained from training on such classification problems also could be applied to the 6-DoF pose regression problem. They suggested using already existing large classification datasets, in the size range of 7 - 14 million images, to pre-train the weights for the PoseNet network. The network was trained further on a smaller regression dataset, where each image corresponds to a 6-DoF pose in the continuous, infinite domain.

2.3.4 Extensions to PoseNet

Several extensions and improvements to PoseNet have been presented in literature, discussed in the following.

Geometric Loss

One of the main drawbacks with the PoseNet architecture is the scaling factor β in the loss function seen in Equation 2.1. This hyperparameter was tuned with a grid search, which is an expensive tuning algorithm. One suggested solution to this is to apply a geometric loss function that is based on the geometry of the image scene [27]. This loss function imposes knowledge about the camera setup geometry, and the image projection on the camera, in similarity with the perspective-n-point problem discussed in Section 2.2.1. The work of [27] demonstrated that the geometric loss function improves the accuracy of the pose estimates compared to the original PoseNet model. However, the model requires a good initial weight during training which can be hard to identify. Li et al. [33] followed up on this and suggested a new angle-based reprojection loss function. This loss function allows training of the CNN without being sensitive to the initial weights. However, the models in both [27] and [33] require geometric modeling of the image scene.

Euler Angle Representation

To avoid the problem of tuning the scaling coefficient β, another suggested approach is to simply regress the Euler angles instead of the quaternion [34]. By using Euler angles as the angle representation, the network regresses translation and rotation to 1 meter and 1 degree respectively. The authors of [34] therefore argued that no scaling coefficient is needed, corresponding to $\beta = 1$. To overcome the periodicity issue with Euler angles, the angles are constrained to the interval $(-\pi, \pi]$. This model overcomes the problem of expensive parameter tuning, but does not achieve the accuracy of PoseNet.

BranchNet and Angle Representation

The employment of quaternions as orientation representation is not necessarily optimal, because a quaternion represents the same orientation as its additive inverse, that is $\boldsymbol{q} = -\boldsymbol{q}$. This, together with the regression of translation and orientation as a whole in PoseNet, is challenged in the work by Wu et al. [35]. They suggested a variant of Euler angles named Euler6, as an alternative to quaternions, and a network structure called *BranchNet* to present the relationship between translation and rotation. The Euler6 representation is intended to cope with both the periodicity issue with Euler angles, and the additive inverse issue with quaternions. Euler6 is a 6D vector defined in Equation 2.2, where ϕ, θ and ψ corresponds to the original Euler angles.

$$e = [\sin\phi, \cos\phi, \sin\theta, \cos\theta, \sin\psi, \cos\psi] \tag{2.2}$$

The relationship between translation and rotation is captured in BranchNet, which is a version of PoseNet that introduces two branches used to predict translation and orientation separately. BranchNet consists of two parts, the first part being the shared layers and the second part being the specific layers. This removes the need for the challenging scaling factor, β, in the loss function of PoseNet, seen in Equation 2.1. The two components are illustrated in Figure 2.4, showing PoseNet on top in comparison with BranchNet below.

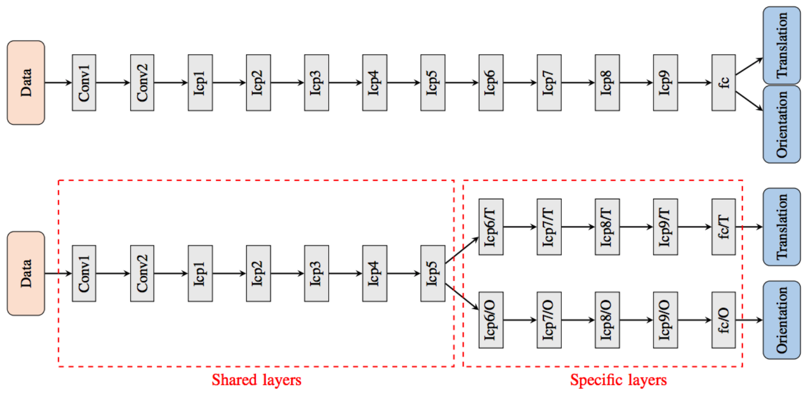


Figure 2.4: Architecture of PoseNet(top) and BranchNet(bottom) [35]

This structure allows for the network to extract features shared by the orientation and translation predictors in the shared layers, before the two are predicted separately in the specific layers. The channels of the specific layers are reduced to avoid an increased number of parameters in this new structure. The experiments of [35] revealed that both the introduction of Euler6 representation of rotation, and the BranchNet structure improve the localization accuracy compared to PoseNet. The fact that the branched learning improved performance contradicts the corresponding discussion in [9], where the opposite was concluded.

Bayesian Model

Kendall et al. suggested extending the PoseNet model with a Bayesian model, that features modeling of uncertainty [29]. The Bayesian PoseNet model is in short obtained by adding dropout (see Section 3.2.3) to some of the layers. This model represents a probabilistic approach that estimates the metric relocalization error, making it possible to understand to what degree the predicted pose is trustworthy. The estimated uncertainty is also taken into account to adjust the predictions, by placing a probability distribution over the weights. With this incorporation, the Bayesian PoseNet was shown to deliver localization with higher accuracy than the non-probabilistic PoseNet.

Feature Selection

Another extension combines the PoseNet architecture with Long-Short Term Memory (LSTM) units [28]. The LSTM units are inserted into the FC output layer of PoseNet, to reduce the dimensionality of this layer. The LSTM units choose the most useful features for the pose regression task, which is more effective than directly reducing the dimensions of the FC layer. The purpose of this reduction was to prevent overfitting.

Chapter 3

Theory on Deep Neural Networks

This chapter presents the relevant theory for the work of this thesis. The content of this chapter is mainly based on the books by Goodfellow et al. [24] and Gibson et al. [36], in addition to lecture notes from the UC Berkeley class Introduction to Artificial Intelligence [37]. Additional sources are specified when used.

Deep learning is a subset of machine learning (ML), that allows for computers to learn complex relations from experience. Deep learning can be applied in all three categories of ML; supervised-, unsupervised- and reinforced learning. A deep neural network (DNN) is an instance of supervised machine learning and can be applied to both regression and classification problems. DNNs are applied to a wide range of fields such as robotics, computer vision and speech recognition. This thesis focuses mainly on the category deep feedforward neural networks, which aims to predict an output $\boldsymbol{y}$ given an input $\boldsymbol{x}$. These networks are designed to approximate a mathematical function f, that maps an input $\boldsymbol{x}$ to an output $\boldsymbol{y}$, with $\boldsymbol{y} = f(\boldsymbol{x})$. The feedforward network represents the mapping $\boldsymbol{y} = \hat{f}(\boldsymbol{x}; \boldsymbol{W})$, and is trained to learn the network parameters, also referred to as the network weights, $\boldsymbol{W}$, that gives the best approximation of f. Such DNNs are called feedforward since they feed the input values $\boldsymbol{x}$ through the multiple layers until it reaches the output $\boldsymbol{y}$. Section 3.1 explains the structure of such deep neural networks, and Section 3.2 describes the procedure of how the networks learn.

Convolutional neural networks are a particular type of neural networks operating on multidimensional data, shown to be extremely successful in computer vision applications. The deep learning methods for pose estimation presented in Chapter 2 are based on CNN architectures, as is the model implemented in this thesis. Theory on CNN is therefore dedicated its own section in Section 3.3.

Recurrent neural networks (RNNs) are another type of neural networks, specialized to process sequential data. This network structure is considered relevant for further improvements of existing pose estimation models, and is presented in Section 3.4.

3.1 General Neural Networks

An artificial neural network (ANN) models a learning process, and was initially intended to represent learning in biological brains, hence the name artificial neural network. While neural networks perform excellently in many applications, they are not designed to be realistic models of biological brains.

The building blocks of a neural network are called artificial neurons, each having an activation function. These artificial neurons are organized in multiple layers of the network; an input layer, hidden layers and an output layer. The input values, $\boldsymbol{x}$, are fed into the input layer, and the output values, $\boldsymbol{y}$, are returned by the output layer. A network consisting of several hidden layer is considered a deep neural network (DNN). The structure of a neural network with two hidden layers is shown in Figure 3.1.

The architecture of a DNN can be defined in terms of:

- Number of hidden layers
- Number of neurons per layer
- Type of activation functions

These parameters are not trainable and are design parameters that must be set by the constructor of a network.

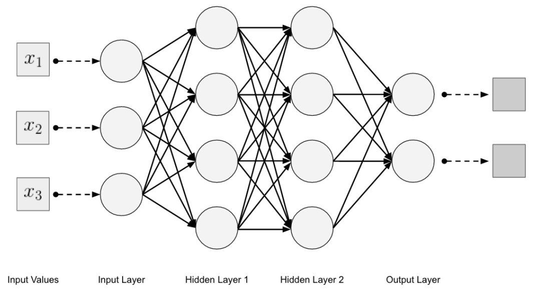


Figure 3.1: Structure of a neural network with two hidden layers [36]

Each of the interconnections, j, within the network is associated with a weight, w_j, describing to which extent a specific neuron in one layer influences a specific neuron in the next layer. The total weights, $\boldsymbol{W}$, make up the trainable parameters of the network, and encodes the knowledge of the network. The network learns and improves by updating $\boldsymbol{W}$, following the procedure described in Section 3.2. A DNN can represent any mathematical function f, as long as it consists of enough neurons and layers. DNN makes it possible to model the relationship between some input values, $\boldsymbol{x}$, and output values, $\boldsymbol{y}$, without having to select a model explicitly.

3.1.1 Artificial Neuron

Artificial neurons, referred to as neurons from now on, are the fundamental building blocks in DNNs. A neuron i takes in n inputs, x_j for $j = 1, ..., n$, from the previous layer, and returns an output called an activation, a_i. The neuron contains n weights, w_j for $j = 1, ..., n$, each associated to one of the n inputs. Both the inputs and the weights can be stored in vectors, $\boldsymbol{x_i}$ and $\boldsymbol{w_i}$, respectively, and the dot product of the two are fed into an activation function, g. The weights, $\boldsymbol{w}_i$, for all the neurons, i, together make up the total network weights $\boldsymbol{W}$. The activation function returns the activation value, a_i, which is further fed into the neurons of the next layer, through the neuron's outgoing connections. Whenever the neuron feeds a nonzero activation value to the next neurons, it is said to be activated. The structure of a neuron is illustrated in Figure 3.2.

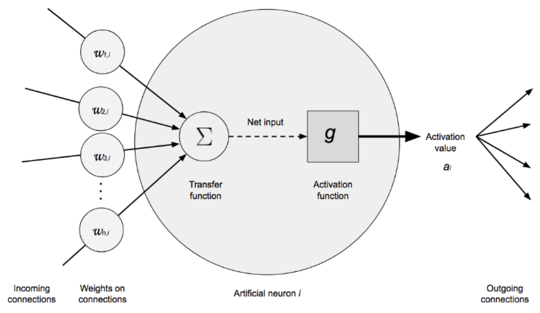


Figure 3.2: Artificial neuron [36]

A bias term is also added to each neuron, to define a lower or upper limit for an activation to be nonzero. This bias term, b, can be incorporated in the neuron by adding a dummy entry, $x_0 = 1$, to the input vector, $\boldsymbol{x}_i$, and an associated weight, w_0, to the weight vector, $\boldsymbol{w}_i$. The w_0 term thus represents the bias value, and is included as a trainable parameter of the network. The bias is therefore considered to be part of the weights, and is included when referring to the weights in the following. The activation value, a_i, of neuron i is given in Equation 3.1, where $g(\cdot)$ is the activation function.

$$a_i = g(\boldsymbol{w_i} \cdot \boldsymbol{x_i}) \tag{3.1}$$

3.1.2 Activation Function

The activation functions bring nonlinearity into neural networks. Without them, the networks would present linear relations between input and output, which is simply a matrix multiplication representation. The activation function transforms the linear combination of inputs and weights into the neuron's activation value, a_i. It takes a scalar input, and returns a scalar output. Some of the activation functions are designed to output values within some specific range, typically [0,1] or [-1,1]. Some commonly used activation functions are presented in Appendix B.

Which activation functions to apply to a neural network is a design decision, and it usually varies between the output layer and the hidden layers. For the hidden layers, commonly used activation functions include sigmoid, tanh and the rectified linear unit (ReLU). The latter is mostly used in recent years, and different versions of ReLU are investigated in research. It is shown that the usage of ReLUs in the hidden layers of DNNs reduces the training time, compared to tanh and sigmoid [38]. This is due to its non-saturated gradient, which accelerates the convergence of the gradient descent algorithm described in Section 3.2.2.

For the output layer, the choice of activation function depends on the task of the network. A DNN can be used for either classification or regression purposes. In the case of a regression network, the desired output is an unbounded real-valued number, and a linear activation function is typically used. Classification networks, on the other hand, typically return probabilities associated with each of the possible output classes, meaning the desired output is a probability distribution between 0 and 1. If the classification problem is binary, only one number is needed, and the sigmoid function can be used. In the case of multiclass classification, softmax is used since it returns a probability distribution over all the possible output classes. If the network is a hard classifier, it returns the single class predicted for the data point. To achieve this, an arg-max function can be used on top of either sigmoid or softmax.

3.2 Learning

This section explains the procedure in which a neural network learns from experience. During learning, the network weights, $\boldsymbol{W}$, are adjusted to improve its performance. This process is referred to as learning or training in literature. Training a neural network requires a labeled dataset, $\mathcal{D}$, containing n data points $d_j, j = 1 \dots n$, where each data point consists of input and output pairs $\{\boldsymbol{x}_j, \boldsymbol{y}_j\}$. The dataset is divided into training and test sets, and the network adjusts its weights based on its performance on the training set. To make sure these adjustments generalize well to unseen data, the network is tested on the test set.

For the network to improve its performance on the training set, a measurement of the performance is defined with a cost function, also referred to as loss function, error function, or objective function in literature. The cost function describes the difference between the predicted output of the neural network, $\hat{\boldsymbol{y}}$, and the true value, $\boldsymbol{y}$. The goal of the learning process is to adjust the weights such that the value of the cost function is minimized, and the learning process is thus reduced to an optimization problem. The cost function is described in Section 3.2.1. To optimize the cost function with respect to the weights of the neural network, the optimization algorithm gradient descent is widely used. This is an iterative algorithm that updates the weights based on the gradient of the cost function in each iteration. The gradient of the cost function can be calculated with backpropagation, which, together with gradient descent, is explained in Section 3.2.2. The problem of overfitting, and techniques to avoid it, is described in Section 3.2.3. The hyperparameters of neural networks are presented in Section 3.2.4.

3.2.1 Cost Function

The cost function quantifies the error of the predictions of a neural network, and is the objective function that is to be minimized with respect to the weights. The cost function is thus a function of the true and predicted output, $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$, respectively. With $\hat{\boldsymbol{y}}$ being the output of the neural network for input $\boldsymbol{x}$, the cost function can also be expressed as a function of the network weights, $\boldsymbol{W}$, according to Equation 3.2. During training, the cost can be averaged over all the data points in the training set, and thus represent the network's performance. Similar to the activation function, the choice of cost function depends on the task of the network. For a regression network, it is normal to use a mean squared error loss, while hinge loss and logistic loss functions are common for classification networks. The model implemented in this thesis is a regression network, and therefore, only the mean squared error loss function is defined in the following.

Mean Squared Error Loss

The mean squared error function, also referred to as the Euclidean loss function, is defined in equation Equation 3.2.

$$L(\hat{\boldsymbol{y}}, \boldsymbol{y}) = J(\boldsymbol{W}) = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2 = \frac{1}{M}||\hat{\boldsymbol{y}} - \boldsymbol{y}||_2 \tag{3.2}$$

In Equation 3.2, $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}$ are the predicted and true output vectors, respectively, M is the number of entries in the output vector and $\boldsymbol{W}$ represent the network weights.

3.2.2 Gradient Descent with Backpropagation

The problem of minimizing the cost function with respect to the weights of the network, is hard to solve analytically and is normally solved with the algorithm gradient descent combined with backpropagation. Gradient descent is an iterative algorithm, that calculates the gradient of the cost function with the current values of the weights. The weights are then updated along the direction of this gradient, scaled by the learning rate, α. The gradient of the cost function measures the direction of steepest ascent, implying the weights should be adjusted along the negative gradient. Each iteration during training is referred to as an epoch. The algorithm is defined in Algorithm 1.

Algorithm 1 Gradient descent

```
initialize weights $\boldsymbol{W}$
for $ep$ in $episodes$ do
    calculate gradient $\nabla_W J$
    update weights $\boldsymbol{W} = \boldsymbol{W} - \alpha \nabla_W J$
end
```

The algorithm needs to calculate the gradient for each weight in the network, which is done with the backpropagation algorithm. This algorithm represents the neural network as a computational graph. The graph structure is effective for both calculating the cost function value, and the gradient of the cost with respect to each of the weights. Calculation of the cost function value is done in forward pass, meaning that each node's operation is applied to the input value coming from the node's parents. Calculations of the gradients are done in backward pass, meaning the gradient is passed and updated backward in the graph, starting at the final node. The gradient for each weight is found by using the chain rule.

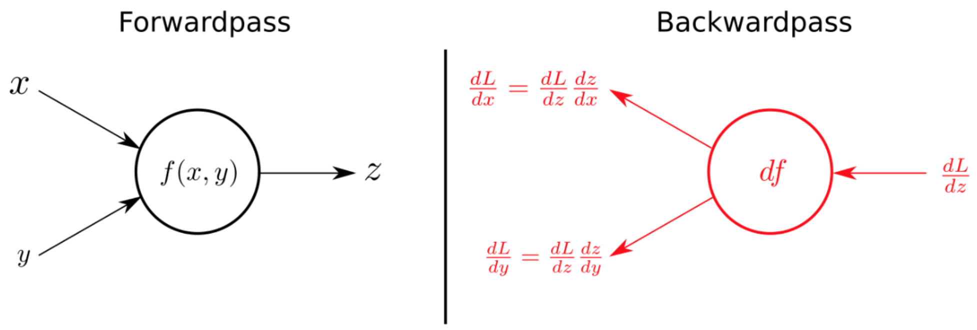

Forwardpass
Backwardpass
x
y
$f(x,y)$
z
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$
df
$\frac{dL}{dz}$

Figure 3.3: Forward and backward pass during gradient descent [39]

The forward pass and the backward pass are illustrated in Figure 3.3. In this figure, x and y represents input values to a node, while z is the output values of a node. (This should not be confused with the previous notation where $\boldsymbol{x}$ and $\boldsymbol{y}$ represents input and output, respectively, of a network.) The forward pass to the left calculates z as a function of values from the node's parents, x and y. The right side of the figure shows the backward pass. The node receives the gradient of the loss function with respect to parameter z, $\frac{dL}{dz}$, from the node's child. The gradients of the loss function with respect to x and y, $\frac{dL}{dx}$ and $\frac{dL}{dy}$, can then be found from calculating $\frac{dz}{dx}$ and $\frac{dz}{dy}$ and apply the chain rule as illustrated. Since the gradients of the activation function of all nodes are calculated, the activation functions of the network are required to be continuously differentiable. Further, for the gradient descent algorithm to converge, it is desired to use activation functions whose gradients are non-saturating.

When the gradient descent algorithm calculates the gradient based on all the data points in the training set in one iteration, it is called batch gradient descent. For large datasets, the computation of all these gradients is very slow. To reduce training time, mini-batching is commonly used, a technique that uses a batch of k data points for calculation of the gradient. This accelerates the computation time for each gradient update, while it still makes a fast progression to the minimum of the cost function. The mini-batch gradient descent algorithm rotates in batches of size k through the entire training set in each epoch. If the batch size k is set to 1, the algorithm is known as stochastic gradient descent (SGD). It is stochastic since the single data point is chosen randomly from the training set in each epoch. The stochastic approach is also extended to include several random data points in each iteration. The use of SGD is a common approach for training neural networks.

Improvements to the gradient descent algorithm have evolved into a research field in itself. Momentum is one commonly applied method, described in the following.

Momentum

Momentum is a method that can be applied to gradient descent, which speeds up the convergence of the algorithm [40]. In cases when the gradient is much larger in some directions than others, the gradient descent algorithm might get stuck, by updating the weights in an oscillatory manner. Momentum takes care of this by adding a portion of the previous weight update to the current weight update. The gradient descent algorithm with momentum is defined as in Algorithm 2.

Algorithm 2 Gradient descent with momentum

```
initialize weights $\boldsymbol{W}$
for ep in episodes do
    calculate gradient $\nabla_W J$
    $\boldsymbol{W}^n_{\text{update}} = \gamma \boldsymbol{W}^{n-1}_{\text{update}} - \alpha \nabla_W J$
    update weights $\boldsymbol{W}^{n+1} = \boldsymbol{W}^n - \boldsymbol{W}^n_{\text{update}}$
end
```

The weight update is now modified to include an amount γ of the previous weight update, yielding a decaying average over the past gradients, $\nabla_W J$. This decaying rate, γ, is commonly set to be around 0.9. The usage of momentum reinforces weight updates in the same direction of the previous update, and reduces weight updates whenever the update changes direction.

Several extensions of momentum are suggested to be combined with gradient descent [41],[42],[43],[44]. They have in common that they try to damp oscillatory updates to speed up the convergence, by updating the learning rate for each parameter in an adaptive manner. Only the Adam algorithm will be explained in the following, since it is applied in the work of this thesis.

Adam

Adaptive Moment Estimation (Adam) is an optimization algorithm that calculates adaptive learning rates individually for each parameter in the network [45],[44]. Similar to momentum, Adam calculates the exponentially decaying average over the last gradients, denoted m_n. In addition, the exponentially decaying average over the last squared gradients, denoted v_n, is calculated as in Equation 3.3.

$$\begin{aligned} m_n &= \beta_1 m_{n-1} + (1-\beta_1)g_n \\ v_n &= \beta_2 v_{n-1} + (1-\beta_2)g_n^2 \end{aligned} \tag{3.3}$$

In Equation 3.3, g_n corresponds to the gradient at time step n, and β_1 and β_2 are decay rates. (g_n thus corresponds to $\nabla_w J$, and β_1 corresponds to γ in the momentum algorithm.) m_n and v_n are referred to as the first and second moment, respectively. In order to prevent that these moment estimates are biased towards zero, which happens as a consequence of being initialized to zeros, their bias-corrected values are calculated with Equation 3.4.

$$\begin{aligned} \hat{m_n} &= \frac{m_n}{1-\beta_1^n} \\ \hat{v_n} &= \frac{v_n}{1-\beta_2^n} \end{aligned} \tag{3.4}$$

Finally, each individual weight, w^{n+1}, are updated according to Equation 3.5, where α is still the learning rate. ϵ is a term added to avoid division by zero, typically in the order 1e-08.

$$w^{n+1} = w^n - \frac{\alpha}{\sqrt{\hat{v_n}} + \epsilon}\hat{m_n} \tag{3.5}$$

The authors of the Adam paper [44] showed that the optimization algorithm outperforms other adaptive learning algorithms, and it has become popular within the field of deep learning optimization. The hyperparameters of the Adam optimizer are the decay rates β_1 and β_2, the learning rate α and the smoothing term ϵ.

3.2.3 Overfitting

After training, the network is tested on the training set to make sure it generalizes well to unseen data. If the test errors are large, the network is likely fit too closely on the training data, referred to as overfitting. This is a challenge within DNNs, especially when the available data is limited. While overfitting typically occurs for small training sets, it can be prevented with regularization techniques, such as dropout or L1 and L2 penalties.

Dropout regulates a network by omitting one or more hidden units. During training, a number of neurons are randomly selected to be muted, implying they will not affect the current iteration of forward pass or backpropagation. Dropout is, therefore, a way of averaging the network weights. In addition to prevent overfitting, dropout also reduces training time.

A DNN can also be regulated by adding a penalty term to the cost function, to prevent unlimited growth of the weights. With this approach, the weights yield smaller values, and thus represent smaller assumptions based on the training data, improving the ability of the network to generalize to unseen data. The penalizing term is typically the L1- or L2-norm of the weights, scaled by some coefficient λ. The L2 norm is the 2-norm, that is the Euclidean norm, squared. These norms are defined in Equation 3.6.

$$\begin{aligned} L1(\boldsymbol{x}) &= \sum_{i=1}^{n} |x_i| \\ L2(\boldsymbol{x}) &= \sum_{i=1}^{n} x_i^2 \end{aligned} \tag{3.6}$$

L1 regularization, referred to as Lasso regression, makes the weights sparse meaning it provides feature selection. L2 regularization, referred to as ridge regression, limits the value of all the weights as it adds the sum of the squares of all the weights to the cost function. L2 regularization, therefore, prevents the too large impact of each individual weight. L1 regularization is less computationally efficient than L2, since the L1-norm does not have an analytic solution.

3.2.4 Hyperparameters

The parameters of a neural network can be split into trainable parameters, being the weights, $\boldsymbol{W}$, and tuning parameters referred to as hyperparameters. The choice of hyperparameters is a design decision, and some of the most essential are listed below.

- Number of hidden layers
- Layer size - number of neurons in each layer
- Activation functions
- Cost functions
- Weight initialization
- Learning rate,α, decay rate, γ
- Batch size
- Number of epochs during training
- Regularization approach

There are several approaches on how to identify the best hyperparameters. The goal is to choose the hyperparameters that give the smallest test error, which in principle could be defined as an optimization problem. Such hyperparameter optimization problems may require their own hyperparameters, and can be hard to solve. Different techniques for selecting the hyperparameters are researched, but they are not covered in this thesis. For further reading see [24].

3.3 Convolutional Neural Networks

This section is based on the online course Convolutional Neural Networks taught by Andrew Ng at Coursera [46], lecture notes from the UC Berkeley class Introduction to Machine Learning [47] and the Stanford class Convolutional Neural Networks for Image Recognition [48].

CNNs, also referred to as convnets, are a special type of neural networks that are designed to process data that comes in a grid-like format. Fukushima was the first to introduce the structure of CNN, by another name at that time, in 1980 [49]. LeCun et al. published the first CNN network LeNet in 1998 [50], and CNNs have been a hot topic of research since this. These types of networks are mostly applied to image data, and has turned out to be extremely successful in applications within computer vision such as image classification and object detection. This chapter describes the most important aspects of CNNs, and lays a theoretical foundation for the implemented model described in Chapter 4.

The main drawback of processing large images with the general neural network structure presented in Section 3.1 is the large number of network weights. The size of an image is W x H x C, where C is the number of channels in the image, equal to 3 for RGB images. W and H correspond to the image width and the image height, respectively. A standard DNN takes this image flattened out as input, and passes it through several fully connected (FC) layers. For a 200 x 200 RGB image, this gives 200 x 200 x 3 = $120,000$ weights from the input layer only. With several layers in the neural network, this adds up to a huge total number of weights, and it is extremely computationally expensive to train such a network. A large amount of weights also implies a high variance in the network, and it is difficult to get enough data to prevent overfitting. Besides, when using FC layers, the structure of the input is discarded, which may contain valuable information. CNNs solve these problems by considering local areas of the input, and exploit the 2D structure of images to reduce the number of weights.

The name convolutional neural networks refer to the convolution operation, described in Section 3.3.1. CNNs are thus neural networks where the convolutional operation is applied to at least one of the layers. A layer where this operation is applied is called a convolutional layer, which is described in Section 3.3.2. In addition to convolutional layers, CNNs introduce pooling layers, described in Section 3.3.3. The architecture of a CNN is finally described in Section 3.3.4.

3.3.1 The Convolutional Operator

The convolutional operator operates on two real valued functions , $x(t)$ and $f(t)$, and is defined in Equation 3.7.

$$s(t) = (x * f)(t) \tag{3.7}$$

In the context of machine learning, the first argument, x, is referred to as the input, the second argument, f, as the filter or kernel, and the output, s, is referred to as the feature map. In a convolutional layer, the input is convolved with the filter. The purpose of applying this convolutional operator is to extract features from the input images with the filter.

For the 2D-case, corresponding to an image with only 1 channel, the image of dimensions W x H is convolved with a filter of dimensions w x h. The convolution is performed by sliding the w x h filter over the W x H image by moving 1 pixel at a time. For each position, it calculates the dot product of the filter matrix and the local subset of the image matrix, which is returned as one element of the output matrix. This output matrix, referred to as the feature map, is of dimension $(W - w + 1)$ x $(H - h + 1)$. An example of the 2D convolutional operator is shown in Figure 3.4. The input image is to the left, the filter in the middle and the feature map to the right. In this example, the image is 7 x 7, the filter is 3 x 3, hence the feature map is of dimension $(7 - 3 + 1)$ x $(7 - 3 + 1) = 5$ x 5.

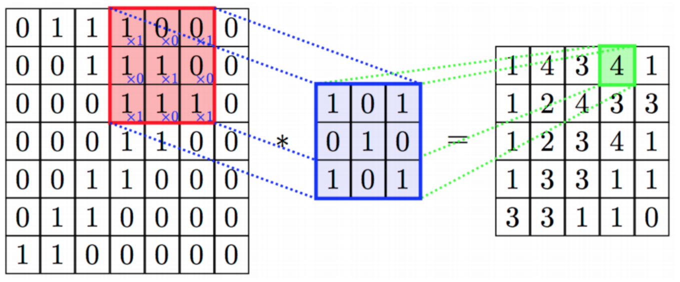


Figure 3.4: Example of 2D convolution operation [47]

The 2D convolution operator explained above can easily be extended to a 3D operator. In the case of 3D convolution, the depth of the filter corresponds to the number of channels, C, in the input image. That is, the depth of the filter and the image must match. This makes it possible to use different filters for the different channels in the input. The filter is slid over the image as before, and the matrix dot product is added together along the depth direction of the filter. The dimension of the resulting feature map is hence still 2-dimensional. An example of this 3D-convolution operator is given in Figure 3.5, where a 6 x 6 x 3 image convolved with a 3 x 3 x 3 filter, yielding a 4 x 4 feature map.

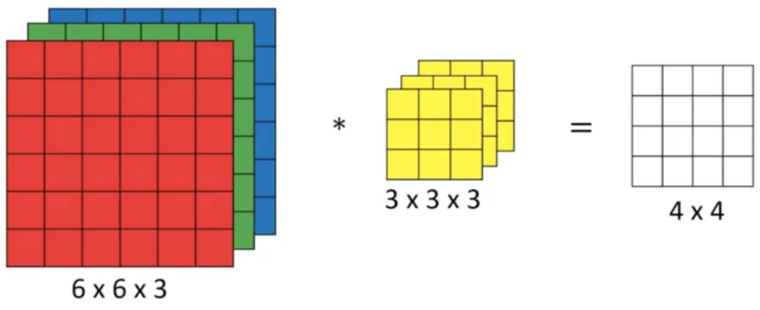
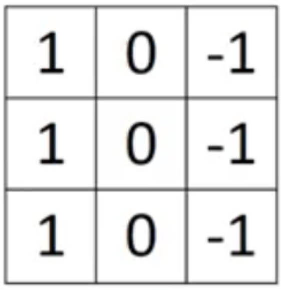


Figure 3.5: Example of 3D convolution operation [51]

In order to adjust the output dimensions of a convolution operator, the parameters stride and padding can be adjusted. The stride is the number of pixels the filter is slid at each step. Increased stride means that the filter jumps multiple steps at a time, and it requires a reduced number of positions until the filter has covered the entire image. Thus, increasing the stride means reducing the output volume. Another feature used to control the output dimensions is to pad the input image with zero-entries along the border. If the input is padded such that the input and output dimension are equal, the convolution is called same convolution. Convolution without padding is referred to as valid convolution.

The convolution operator is extremely useful for extracting features from an image. Filters can be constructed to detect all kinds of features in an image, such as edges, curves or objects. The feature to be detected is encoded in the filter.

As an example, a vertical edge detector filter can be seen in Figure 3.6. The intuition behind this filter is that the left side captures a bright area, the middle column is indifferent (it does not capture anything due to the zeros) and the right column captures dark areas. The filter will, therefore, output greater values when the filter is placed over a part of the image containing this transition from the bright to the dark area, namely a vertical edge. A filter convolved with an image will in general return large values in areas of the image that are similar to the filter.



1	0	-1
1	0	-1
1	0	-1

Figure 3.6: Vertical edge detection filter [51]

3.3.2 Convolutional Layer

A convolutional layer, a conv layer, is a layer that consists of one or more filters. Each filter in the layer outputs a 2-dimensional feature map, implying several filters produce several feature maps. When stacking these maps together in the depth dimension, the output of a convolutional layer becomes 3-dimensional, where the depth corresponds to the number of filters in the layer. This is why convolutional layers are referred to as volume layers. When using several filters in a layer, different patterns in the image can be detected in the same layer. In CNNs, the entries of the filters are included as part of the network weights, $\boldsymbol{W}$. That is, the network learns its own filters, meaning it learns which features to detect in each layer to best predict the output of the network.

A convolutional layer introduces weight sharing, which is a huge advantage with CNNs. Through the usage of filters, the same weights are used along all the pixels of the input, and each component of the feature map is determined by the same weights. The applications of filters also impose sparse connections, since they reduce the dimensions from input to output. The combination of weight sharing and sparse connections implies a significant weight reduction compared to fully connected layers, where each input-output pair has an associated weight. In addition to reduced training time, the variance of the network is reduced with convolutional layers, due to fewer weights.

The hyperparameters of a convolutional layer are given in the list below.

- Number of filters, i.e. output depth
- Filter size
- Stride
- Padding

3.3.3 Pooling Layer

In addition to the convolution layer introduced in the previous section, most CNN architectures employ pooling layers between the conv layers. The purpose of these layers is to reduce the dimensions of the network successively, hence reduce the weight parameters of the network. This will, in turn, further reduce both the computation cost and variance of the network, preventing overfitting.

Pooling layers apply a pooling function, in which each output component represents a statistical summary of a fixed size part of the input. As an example, the max pooling function returns the maximum of a local area of the input. The pooling filter is applied to every depth slice of the input independently; hence, the pooling layer is 2-dimensional. Such layers are defined by filter size, stride and the pooling function itself. In addition to max pooling, suggested pooling functions include the average pooling an L2 norm pooling.

Max pooling layers are most used in practice, normally with stride $S = 2$, and filter size $F = 2$ x 2 or $F = 3$ x 3. If the pooling size becomes big, the pooling layer may be too destructive discarding valuable information. An example of a max pooling layer with $S = 2$ and $F = 2$ x 2 can be seen in Figure 3.7. The max of each of the colored 2 x 2 areas of the input images make up the output. This pooling layer reduces an input of 4 x 4 to 2 x 2, implying the layer discards 75% of the parameters.

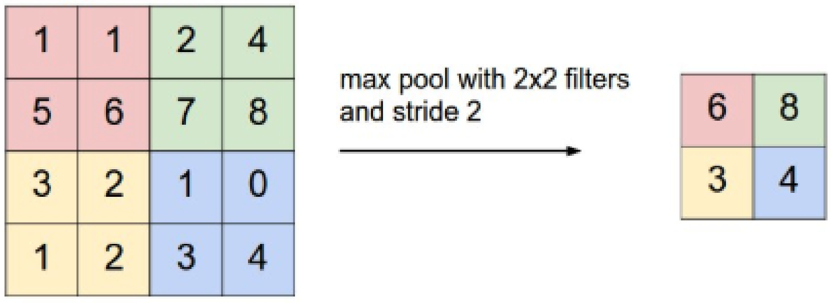


Figure 3.7: Example of the max pooling operation [48]

In addition to reducing dimensions and providing a statistical summary, the employment of pooling layers makes the network invariant to small translations of the input. That is, the network is robust to small translations of the position of the objects in the input image. In cases where the determination of a feature's presence in an image is important, this property is especially useful.

The hyperparameters of a pooling layer are given in the list below.

- Pooling operator
- Filter size
- Stride

3.3.4 Architecture

Convolutional neural networks commonly consist of the three layer types - conv layer (CONV), pooling layer (POOL) and fully connected layer (FC). CONV layers are applied together with an activation function - commonly the ReLU activation, referred to as the detector layer RELU in the following. These different layers stacked together make up a CNN. The typical pattern is to stack a couple of CONV-RELU alternations, followed by POOL. This is repeated a number of times, before one or more FC-RELU combinations are applied at the end where the last FC layer holds the output. This common pattern can be summarized as:

$$\boxed{\text{INPUT} \rightarrow [\ [\text{CONV} \rightarrow \text{RELU}]^{*m} \rightarrow \text{POOL}\ ]^{*n} \rightarrow [\text{FC} \rightarrow \text{RELU}\ ]^{*k} \rightarrow \text{FC}}$$

where * represents a repetition, and m, n, and k are the number of times the expression in the associated bracket is repeated. It is common that small-scale features of images are detected closer to the input, implying the filter size of the CONV layers is reduced in the last layers. Although this linear pattern is common for CNNs, constructing a high-performance architecture is a difficult design decision. It is common to use the design of already established convnets that are shown to have high performance, such as the current ImageNet architecture [38].

The linear architecture pattern discussed above was challenged with the introduction of inception modules, suggested by Szegedy et al. in the architecture of GoogLeNet [32]. Inception modules consist of parallel CONV layers, where the resulting feature maps are concatenated to the final output of the inception module. The motivation behind inception modules is to avoid having to choose the filter size in a conv layer, and rather try out different convolutions in each parallel leaving the network with the decision of which one to use. The inception modules make it possible to discover both local features with the small filters, and global more abstract features with the larger filters at the same time. It is also suggested to add a pooling layer as one of the parallels, since it has turned out to be such a successful component of state-of-the-art CNNs. An inception module is illustrated in Figure 3.8. This module is referred to as a naive inception module, not taking necessary dimensionality reduction into account. For further reading, see [32].

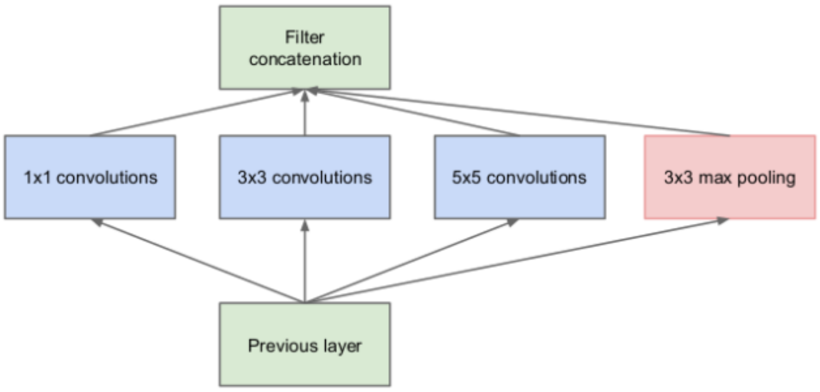


Figure 3.8: Example of inception module [32]

In addition to inception modules, residual networks have been suggested as an alternative to the linear pattern CNN architecture. This was introduced with ResNet by He et al. [52]. This architecture includes the special feature of skipped connections, and has been shown to deliver high performance. Details of residual networks are excluded from this thesis, for further reading see [52].

3.4 Recurrent Neural Networks

This section is based on lecture notes from the Stanford class Convolutional Neural Networks for Image Recognition [53].

Recurrent neural networks are a special type of neural networks that are designed to work in the temporal domain. Similar to how CNNs, introduced in section 3.3, are designed to process data that comes in a grid-like format, RNNs are designed to process sequences of data. Such networks have a unique architecture including loops, making them able to keep information from historical input data in the network. This way, the network can make persistent predictions. Due to these loops, RNNs are not considered a feedforward neural network. RNNs have been successfully applied to a variety of fields such as speech recognition, language translation and stock prediction. This section gives a brief introduction to RNNs.

Traditional neural networks take input of fixed size and return an associated output of fixed size, doing a one-to-one mapping. That means the network starts from scratch every time it evaluates a new input, not taking into account the evaluation of the previous input. Recurrent networks, on the other hand, process sequences of data, having either sequences in the input data, sequences in the output data, or both. Examples of this can be seen in Figure 3.9, where the red boxes represent input vectors, green boxes represent the neural network components, and the blue boxes represent output vectors. From left to right: The first figure illustrates a traditional neural network, mapping one input to one output. The second and third figure illustrates RNNs with sequenced input and sequenced output, respectively. The fourth and the fifth figure illustrate RNNs with both sequenced input and output, so-called many-to-many mappings. RNNs doing many-to-many mappings are considered relevant in this thesis, and will be of focus in the following. RNNs can typically process sequences of variable lengths.

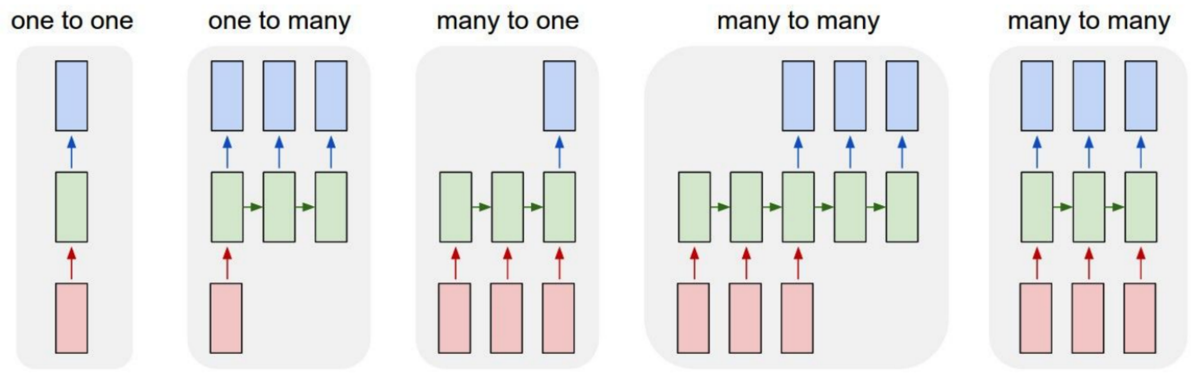


Figure 3.9: Data flow in different RNN structures [53]

3.4.1 Architecture

The architecture of an RNN can be illustrated by adding a loop to a traditional neural network, seen in Figure 3.10. The idea is that the RNN holds an internal state, which is updated in a loop as a sequence is processed by the network.

At each time step, t, the internal state, denoted h_t, is calculated with a recurrence formula, being a function, f_W, of the current input, x_t, and the previous internal state, h_{t-1}. Each output of the network, y_t, can then be represented as a weight, W_{hy}, multiplied with the internal state, h_t. These relations are presented in Equation 3.8.

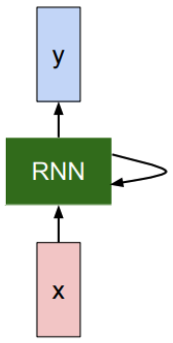


Figure 3.10: Simple RNN [53]

$$\begin{aligned} h_t &= f_W(h_{t-1}, x_t) = f(W_{hh}h_{t-1}, W_{xh}x_t) \\ y_t &= W_{hy}h_t \end{aligned} \tag{3.8}$$

An important concept of RNNs is that the same function, f, and trainable weights, W, are shared across the time steps, in line with the weight sharing feature of CNNs. This is important for the recurrent neural network to be able to generalize to unseen sequence lengths, and to share statistical insight across different time steps. A recurrent neural network can, therefore, be interpreted as several copies of the same network, working together in a chain. The looped illustration of the RNN architecture in Figure 3.10 can thus be unrolled to the network chain seen in Figure 3.11. This network is an example of a many-to-many network, as the ones to the right in Figure 3.9.

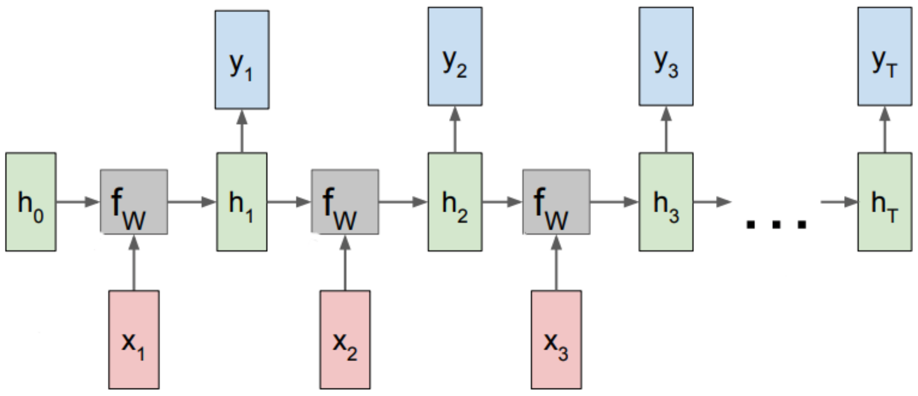


Figure 3.11: RNN architecture [53]

The illustrations and explanation of the RNN architecture above represent a rather simple structure, with only one hidden layer. Similar to CNNs and neural networks in general, RNNs typically work better when they have several hidden layers, yielding deep recurrent neural networks.

Chapter 4

Model

For the experiments in this thesis, the state-of-the-art CNN architecture for pose regression, PoseNet, introduced in Section 2.3, was implemented. That is, the original PoseNet model was chosen, rather than any of its extensions presented in Section 2.3.4. This chapter describes the details of the implemented model. The contents of this chapter are based on [9], but the equations and figures are adjusted for the notation chosen for this thesis

The implemented model outputs the relative 6-DoF pose, $\boldsymbol{\eta}$, between a camera and a target object, from one single input image, $\boldsymbol{X}$. The regressed pose vector, $\boldsymbol{\eta}$, consists of 7 entries, composed of the 3D camera position, $\boldsymbol{p} \in \mathbb{R}^3$, and orientation represented by the quaternion, $\boldsymbol{q} \in \mathcal{S}^3$, according to Equation 4.1.

$$\boldsymbol{\eta} = [\boldsymbol{p}^T, \boldsymbol{q}^T]^T \in \mathbb{R}^3 \times \mathcal{S}^3 \tag{4.1}$$

This chapter describes the learning configuration in Section 4.1, the architecture of the model in Section 4.2 and the training algorithm in Section 4.3. Following, the hyperparameter configuration is given in Section 4.4, and the implementation details are specified in Section 4.5.

4.1 Learning

The network regresses position and orientation simultaneously, and outputs only the pose vector, $\boldsymbol{\eta}$. The network was trained with stochastic gradient descent and an Adam optimizer (see Section 3.2.2), minimizing the loss function in Equation 4.2. This equation corresponds to Equation 2.1, modified for the notation chosen for this thesis. That is, the position vector $\boldsymbol{p}$ in Equation 4.2 corresponds to $\boldsymbol{x}$ in Equation 2.1. Similar, the pose vector $\boldsymbol{\eta}$ corresponds to $\boldsymbol{p}$ in Section 2.3.1. Recall from Section 2.3.1 that the parameter β in Equation 4.2 is a scaling factor used to balance the translation and orientation error. Notation $\hat{(\cdot)}$ denotes an estimate of a variable, such that $\boldsymbol{\eta}$ and $\hat{\boldsymbol{\eta}}$ represents true and predicted pose, and $\boldsymbol{p}$, $\hat{\boldsymbol{p}}$, $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$ represents the true and predicted position and orientation, respectively.

$$L(\hat{\boldsymbol{\eta}}, \boldsymbol{\eta}) = ||\hat{\boldsymbol{p}} - \boldsymbol{p}||_2 + \beta \left|\left| \hat{\boldsymbol{q}} - \frac{\boldsymbol{q}}{||\boldsymbol{q}||} \right|\right|_2 \tag{4.2}$$

4.2 Architecture

The architecture of the implemented model is similar to the PoseNet architecture described in Section 2.3.2. The network consists of 2 convolutional blocks, 9 inception modules and 1 fully connected (FC) layer before the regression layer at the very end. According to the notation in Section 3.3.4, each convolutional block consist of a CONV-RELU alternation followed by a POOL layer. Further, each inception module consists of two layers of parallel CONV-RELU alternations, including one POOL layer. In total, this makes up 23 layers of trainable parameters. The architecture can be seen in Figure 4.1.

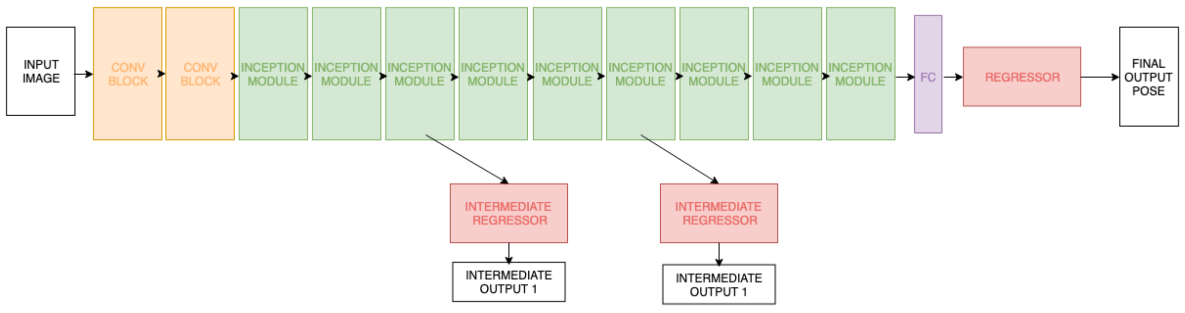


Figure 4.1: Architecture of the implemented CNN regressing 6-DoF pose from one input image

In addition to the output layer, there are two additional intermediate regression layers, inserted after the third and the sixth inception modules, as seen in Figure 4.1. The outputs from these layers are used during training for additional loss assessment. A factor of 0.3 weights the loss from each of these layers, and the total loss that is minimized during training is given in in Equation 4.3. The outputs from these intermediate regressors are discarded at test time.

$$L_{\text{tot}}(\hat{\boldsymbol{\eta}}, \boldsymbol{\eta}) = 0.3L(\hat{\boldsymbol{\eta}}_{\text{int1}}, \boldsymbol{\eta}) + 0.3L(\hat{\boldsymbol{\eta}}_{\text{int2}}, \boldsymbol{\eta}) + L(\hat{\boldsymbol{\eta}}_{\text{final}}, \boldsymbol{\eta}) \tag{4.3}$$

4.3 Training Algorithm

With the network architecture described in the previous section implemented, the training procedure was carried out according to the pseudocode given in Algorithm 3. According to previous notation, $\boldsymbol{W}$ represents the network weights, and $\boldsymbol{\eta}$ and $\hat{\boldsymbol{\eta}}$ represent true and predicted pose, respectively. Introduced in this algorithm, $\mathcal{T}$ represents the training dataset, where each data point consists of an image, $\boldsymbol{X}$, and a pose, $\boldsymbol{\eta}$. The specifics of the Adam optimizer is described in Section 3.2.2.

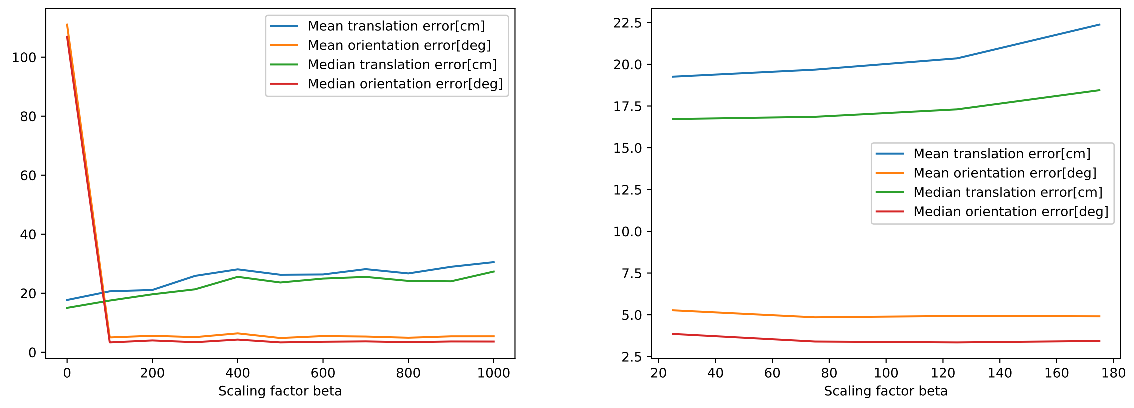
Algorithm 3 Training procedure of implemented network

```
initialize weights $\boldsymbol{W}$ according to weight initialization strategy
for $ep$ in $episodes$ do
    randomly shuffle data points in training set $\mathcal{T}$
    while $\mathcal{T}$ not empty do
        obtain batch of size $k$ from $\mathcal{T}$ consisting of data points $\{\boldsymbol{X}_i, \boldsymbol{\eta}_i\}$ for $i = 1, .., k$
        calculate the averaged loss $J(\boldsymbol{W}) = \frac{1}{k}\Sigma_{i=1}^{k} L_{\text{tot}}(\hat{\boldsymbol{\eta}}_i, \boldsymbol{\eta}_i)$ with forward pass
        calculate the gradients $\nabla_w J$ with backward pass
        update $\boldsymbol{W}$ according to the Adam optimizer
    end
end
```

4.4 Hyperparameters

The hyperparameters used for learning were set based on related work [9],[28], default values in the Tensorflow library, a coarse grid search and by trial and error.

The scaling factor, β, in the loss function in Equation 4.2 is a crucial hyperparameter in this model, as it balances the orientation and translation error. Tuning of this factor with grid search is computationally expensive, requiring training of the network several times. In this project, the value for β is identified by a coarse grid search on dataset 1 (see Table 6.1). First, a coarse search of β values between 0 and 1000 was performed. Based on this, a finer search in the range of interest was conducted. Plots illustrating this tuning can be seen in Figure 4.2. For each value of β, the network was trained on 8000 images and tested on 3000 images. The mean and the median of both the translation error and the orientation error (defined in Chapter 7) were plotted against the β values. Based on this, the value for the scaling factor β was set to 75.



(a) Coarse tuning (b) Finer tuning

Figure 4.2: Grid search for tuning of the scaling factor β for implemented model

The number of necessary training epochs depends on the size of the training data set. For an infinitely large data set, the network only needs to see the data once in theory. That is, the number of epochs is typically smaller for larger datasets. The number of epochs used for learning in this project is therefore adjusted for the size of dataset it learns from in each case, and varies in the range between 100-200.

Although the batch size used in the development of PoseNet is 75, the batch size was set to 10 in this work. According to more recent work, smaller batches have shown better generalization performance [54]. It is stated that even though larger mini-batches exploit the GPU capacity more efficient, batch sizes between 2 and 32 delivers the best performance. The batch size of 10 was identified by trial and errors, and the network performed significantly better with this batch size compared to a batch size of 75.

The values set for the parameters for the Adam optimizer, that is the learning rate α, the decaying rates β_1 and β_2 and the smoothing term ϵ, can be seen together with the remaining hyperparameters in Table 4.1.

Hyperparameter	**Value**
Scaling factor, β	75
Epochs	100 - 200
Batch size	10
Learning rate, α	0.0001
β_1	0.9
β_2	0.999
ϵ	1e-08

Table 4.1: Configuration of hyperparameters

Inspired by the successful application of transfer learning in the development of PoseNet, described in Section 2.3.3, a similar approach was also applied in this work. The network weights were initialized from the GoogLeNet weights trained on the Places image set [55]. The final position regressor layer was initialized with random weights.

4.5 Implementation Details

The code for the implementation of the network and the training algorithm was written in Python, with application of the open source machine learning library TensorFlow [56]. Parts of the implementation was based on available code on GitHub [57], subject to some changes. To be able to train on very large datasets (8-18K), the existing code was rewritten to be more memory efficient.

The datasets used for testing and training are stored in HDF5 files [58], which is a hierarchical data format (HDF) for storing scientific data. The HDF5 format is designed to manage large and complex data collections, and is built for high-speed I/O processing and storage. The latter was crucial in this work, and the use of this file format reduced the training time significantly, compared to other file formats such as pickle files.

Due to the huge size of the network and the datasets, the network was trained on a GeForce GTX 1080 Ti GPU. This accelerated the training significantly, and training with a batch size of 10 took approximately 10-30 hours, depending on the size of the dataset.

The network takes input images of size 224 x 224 pixels, requiring some preprocessing of the images before fed into the network. The images were padded to become square before they were rescaled and cropped to the desired size.

Chapter 5

Simulated Data

A simulated dataset was generated with the 3D modeling software Blender [59]. It was desired to produce a labeled dataset with images of an artificial marker, and the AruCo marker system was chosen for this [15]. The dataset contains images of an AruCo marker taken from several different camera poses. Particularly, the dataset $\mathcal{D}$ contains n data points, $\boldsymbol{d}_i, i = 1 \dots n$, where a data point, $\boldsymbol{d}_i$, contains an image from a camera, as $\boldsymbol{X}_i$, and a relative pose between the camera and the marker, as $\boldsymbol{\eta}_i$, according to Equation 5.1. This chapter describes the procedure of the simulation.

$$\boldsymbol{d}_i = \{\boldsymbol{X}_i, \boldsymbol{\eta}_i\} \tag{5.1}$$

In the Blender environment, a quadratic AruCo marker was placed in space with the middle of the marker being the origin of the global frame, denoted $\{g\}$. A camera was inserted in the environment, with the associated camera frame, denoted $\{c\}$. Figure 5.1 illustrates the setup, where the camera is seen together with its own coordinate frame $\{c\}$ facing the AruCo board. It is emphasized that the underwater vehicle housing the camera was not part of the simulation environment, as it was not necessary to obtain the desired images. The camera frame was aligned to have its x-axis pointing out of the camera lens, its y-axis pointing in its transverse direction, and the z-axis pointing downwards. The x, y and z, axes correspond to the red, green and blue axes in Figure 5.1. When using the camera in the front of an underwater vehicle, the x, y and z-axes represent respectively surge, sway and heave directions, respectively, and the corresponding orientations represent roll, pitch and yaw.

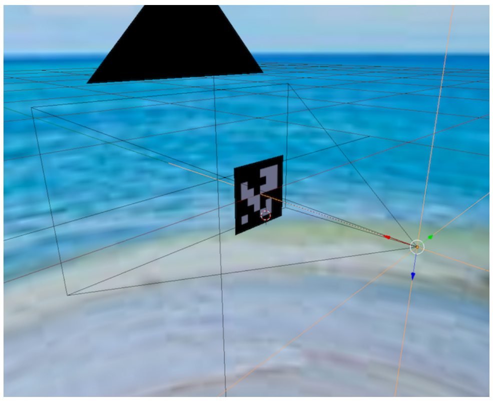


Figure 5.1: Setup in Blender environment, illustrating the camera together with the camera frame in front of the AruCo marker

The camera was set to arbitrary positions along hemispheres with radii in the range between 0.3 m and 1.0 m from the AruCo board. When translated in x, y, z-direction to a point on the hemisphere, the camera was oriented to look at a random point behind the AruCo marker, generating random orientations of the camera. When the camera was translated and oriented, a picture of the board from the current camera pose was captured. The pose vector $\boldsymbol{\eta}^c_{g/c}$ denotes the pose of the global frame $\{g\}$ (and hence the AruCo marker) relative to the camera $\{c\}$ in the camera frame and consists of the position $\boldsymbol{p}^c_{g/c} \in \mathbb{R}^3$ and a unit quaternion $\boldsymbol{q}_{g/c} \in \mathcal{S}^3$. This pose vector was obtained from Blender, and serves as the label for the image.

In each image, a random background was inserted behind the artificial marker. The Open Images Dataset V4 [60] was used for these backgrounds. The images generated in simulation are of size 1280 x 720 pixels. It is emphasized that these images do not represent an underwater environment.

The simulation procedure was programmed to be done in iterations, making it easy to generate datasets of desired sizes. Examples of some images of the AruCo board taken from different poses are illustrated in Figure 5.2. The pose labels of a simulated dataset containing 2000 data points can be seen in Appendix C.1, where the quaternion $\boldsymbol{q}$ is transformed into the Euler Angles around the primary axes of rotations.

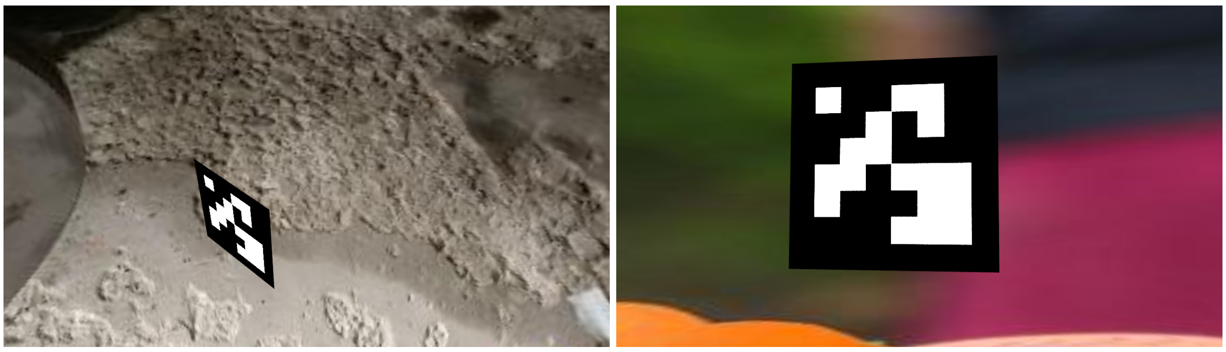


Figure 5.2: Samples from the simulated dataset containing images of the AruCo marker taken from different camera poses

Chapter 6

Real-world Data

The real-world datasets were obtained from underwater experiments in the marine cybernetics laboratory (MC-lab), operated by the Department of Marine Technology at NTNU. The goal of the laboratory work was to generate labeled underwater datasets containing images of a submerged target object taken with a camera on an ROV from varying poses. The AruCo marker used in simulation was also used as the target object for the real-world datasets, in addition to a ring object modeling a subsea valve. Similar to the simulated dataset, the real-world datasets $\mathcal{D}$ contains n data points $\boldsymbol{d}_i, i = 1 \dots n$, where a data point, $\boldsymbol{d}_i$, contains an image from the ROV camera as $\boldsymbol{X}_i$, and a relative pose between the camera and the object as $\boldsymbol{\eta}_i$, according to Equation 5.1.

$$\boldsymbol{d}_i = \{\boldsymbol{X}_i, \boldsymbol{\eta}_i\} \tag{5.1 revisited}$$

This chapter describes the laboratory setup in Section 6.1, the kinematic relations in Section 6.2, and the data processing necessary to obtain the desired datasets in Section 6.3. Section 6.4 describes the properties of the resulting datasets.

6.1 Laboratory Setup

The target object was mounted on a template that was placed in the MC-lab basin, with a fixed position. Two different templates were used to create different backgrounds in the different datasets, namely a metal stick and a yellow box. An ROV equipped with a camera captured pictures of the object from different poses. Figure 6.1 shows the ROV together with the target object attached to the metal stick template. The pose labels were obtained with the Qualisys localization system available in the MC-lab.

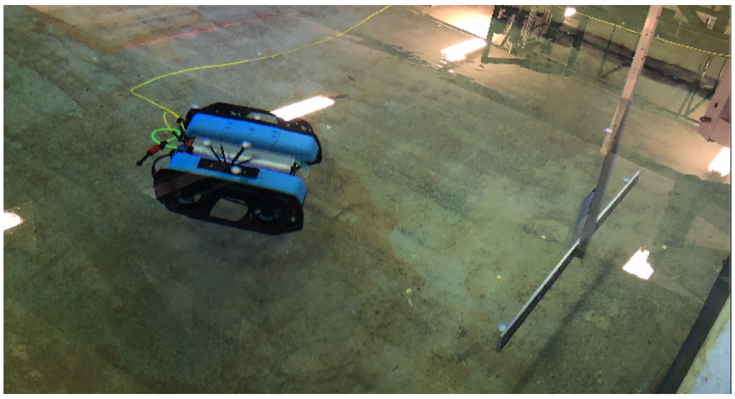


Figure 6.1: Laboratory setup illustrating the BlueROV2 together with a target object attached to the template metal stick in the MC-lab pool

6.1.1 ROV

The ROV model BlueROV2, as seen in Figure 6.1, was used in this experiment. BlueROV2 is a small ROV delivered by Blue Robotics [61], with a thruster configuration that makes it controllable in 6-DoF. This ROV is well suited for research as it can be rigged with different equipment, such as a manipulator arm or different sensors. The ROV was equipped with a Raspberry Pi camera with a wide angle lens [62], as this was the only necessary sensor on the ROV in this experiment. The ROV was steered with an Xbox gamepad controller from the control room in the MC-lab, and was steerable in surge, sway, heave and yaw. The camera on the BlueROV2 was configured to take pictures at a frequency of 15 Hz.

6.1.2 Qualisys Motion Capture

A real-time localization system delivered by Qualisys [63] is available in the MC-lab. The system can measure 6-DoF position and orientation of objects in motion with millimeter accuracy. The measurements from this system are therefore considered to be the ground truth, and serve as labels for the pictures in the data set. The system consists of 6 Oqus cameras placed in the basin, and the software Qualisys Track Manager (QTM). The cameras feature deep blue LEDs, and can illuminate and track markers mounted on the objects to be tracked. The markers, being part of the Qualisys hardware, are small spherical bodies covered in retro-reflective tape designed for underwater purposes. Five markers were mounted on the ROV, and three markers were mounted on the template with the object. One marker on the ROV and one marker on the template were set to serve as the origins of their respective local body frames. With the positions of the markers on the objects known, the pose of the body frames is calculated by the Qualisys software with triangulation theory. The orientation measurements are presented with quaternions, $\boldsymbol{q} \in \mathcal{S}^3$. The pose measurements from Qualisys were generated at a frequency of 25 Hz.

The open-source framework Robotic Operator System (ROS) [64] is used as middleware in the MC-lab. The ROS application subscribes to one image topic and two pose measurement topics (one for the ROV and one for the template). One message is received on these topics for every generated measurement, and this message data is stored in bag files as they are received.

6.2 Kinematics

This section describes the kinematic relations between the lab components, both the reference frames used and the necessary transformations to obtain the relative pose between the camera and the object. The relevant theory presented in this section is based on Chapter 2 in [65].

6.2.1 Reference Frames

When measuring pose, it is necessary to define one or more reference frames. Reference frames can either be earth-centered, with the origin located at the center of the earth, or they can be geographic reference frames with the origin located at other expedient locations. Only the geographic frames are relevant in this work, since the work considers a laboratory environment. The frames NED and BODY are two such geographic frames, and are used to describe the motions in this experiment.

NED

The North-East-Down (NED) system $\{n\} = (x_n, y_n, z_n)$ is defined by a tangent plane on the surface of the Earth. The origin o_n is defined to be a fixed point on the surface of the earth. The x-axis is aligned with the north direction, the y-axis with the east direction and the z-axis points downwards. It is also common to use a rotated version of NED, where the frame is rotated around the z-axis allowing the x- and y-axis to point in other directions. Since the NED frame is defined by a plane tangent to the earth, it cannot be considered inertial. It is however assumed inertial whenever the operational area is local, meaning the longitude and latitude is approximately constant, and Newton's laws are valid. This approximation applies for the MC-lab.

The Qualisys Motion Capture system operates with a NED frame $\{n\}$ defined with the origin placed in a fixed point within the area covered by the Oqus cameras in the basin.

BODY

A body-fixed reference system $\{b\} = (x_b, y_b, z_b)$ is moving with an object, with the origin o_b defined to be at a fixed point on the object.

The Qualisys system defines a body frame for each object it is tracking, where one of the markers mounted on the object is chosen to serve as the origin of this body frame. In this experiment, this means Qualisys defines one body-fixed frame for the ROV, denoted $\{r\}$, and another one for the template, denoted $\{t\}$. The origin of the ROV frame $\{r\}$ is set to be a marker on the top, port, aft part of the ROV, with body axes defined as:

- x_r - longitudinal axis, directed from aft from fore
- y_r - transversal axis, directed from port to starboard
- z_r - normal axis, directed from top to bottom

The origin of the template frame $\{t\}$ is set to be the rightmost marker on the template, with body axes defined as:

- x_t - longitudinal axis, normal to the front plane of the template, aligned with the x-axis in $\{n\}$
- y_t - transversal axis
- z_t - normal axis, directed from top to bottom

The body-fixed reference frames $\{r\}$ and $\{t\}$ can be seen together with $\{n\}$ in Figure 6.2, where the yellow box serves as the template.

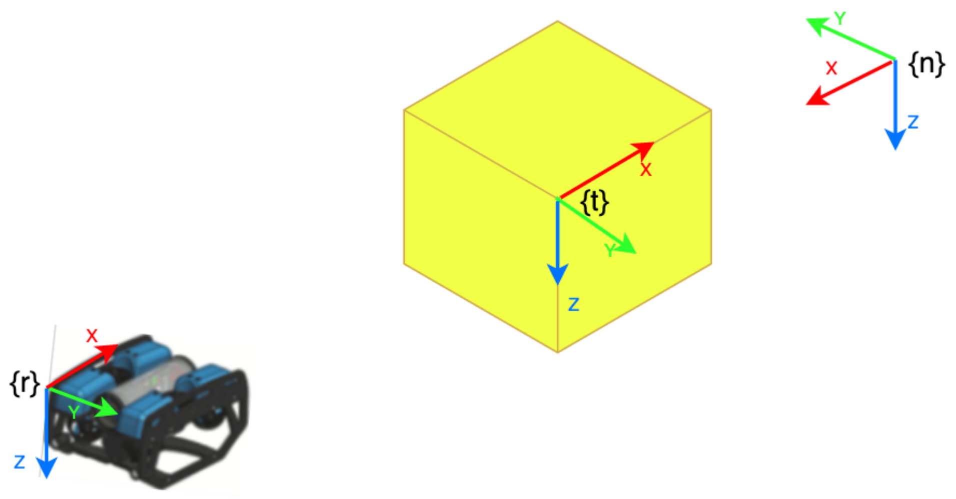


Figure 6.2: Reference frames in MC-lab defined by the Qualisys software

The measurements obtained with the Qualisys software in this experiment is therefore the pose of these two body frames relative to $\{n\}$. The position vector of $\{r\}$ relative to $\{n\}$ expressed in $\{n\}$ is notated $\boldsymbol{p}^n_{r/n} \in \mathbb{R}^3$, and the orientation of $\{r\}$ relative to $\{n\}$ represented in quaternions is notated $\boldsymbol{q}_{r/n} \in \mathcal{S}^3$. Together they make up the pose vector $\boldsymbol{\eta}^n_{r/n} \in \mathbb{R}^3 \times \mathcal{S}^3$. The same applies for $\{t\}$, and the two pose vectors $\boldsymbol{\eta}^n_{r/n}$ and $\boldsymbol{\eta}^n_{t/n}$ make up the total measurements from Qualisys in the experiment.

The desired label for each image is the pose of the object relative to the camera. It is therefore introduced two additional body-fixed frames, namely an object frame, denoted $\{o\}$, and a camera frame, denoted $\{c\}$. The object frame is a translation of $\{t\}$, with the origin o_o placed in the middle of the object that is mounted on the template. Similar, the camera frame is a translation of $\{r\}$, with the origin o_c placed in the position of the camera lens. These translations are given by $\boldsymbol{p}^t_{o/t}$ and $\boldsymbol{p}^r_{c/r}$, respectively. The position and orientation of the object relative to the camera is then the position and orientation of $\{o\}$ relative to $\{c\}$ expressed in $\{c\}$, notated $\boldsymbol{p}^c_{o/c}$ and $\boldsymbol{q}_{o/c}$, respectively. Together they make up the relative pose $\boldsymbol{\eta}^c_{o/c}$, being the desired label for the images in the datasets. How to obtain this pose from the Qualisys measurements is described in the next section. All the relevant frames introduced in this section, $\{r\}$, $\{t\}$, $\{o\}$, $\{c\}$ and $\{n\}$, can be seen together in Figure 6.3.

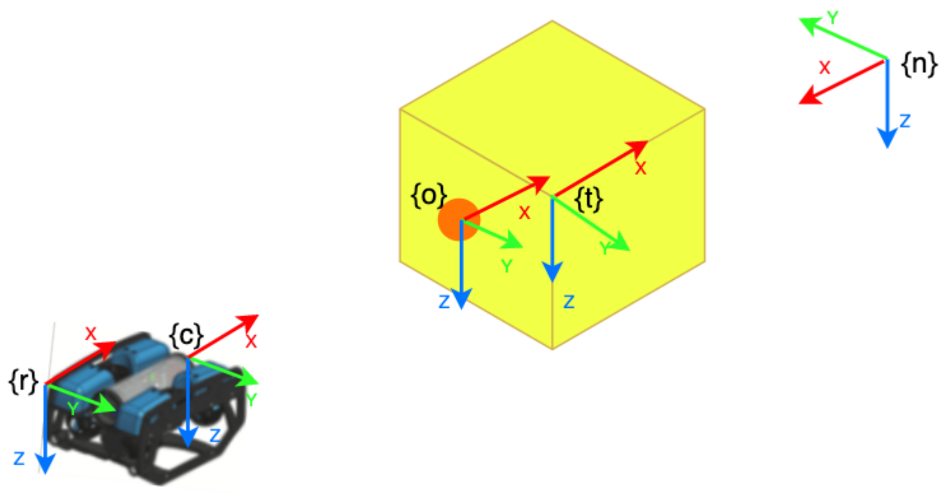


Figure 6.3: Total reference frames in the MC-lab necessary to obtain desired pose labels

6.2.2 Transformations

Rotation Matrices

The rotation matrix between two frames $\{a\}$ and $\{b\}$, denoted $\boldsymbol{R}_a^b$, is used to transform position vectors between the frames. $\boldsymbol{R}_a^b$ is the matrix transforming a vector from $\{a\}$ to $\{b\}$.

$$\boldsymbol{p}^b = \boldsymbol{R}_a^b \boldsymbol{p}^a \tag{6.1}$$

A vector can be transformed back again with the matrix transpose, yielding the relationship $\boldsymbol{R}_b^a = \boldsymbol{R}_a^{b\,T} = \boldsymbol{R}_a^{b\,-1}$, where the last equality holds since rotation matrices are orthogonal.

The rotation matrix $\boldsymbol{R_a^b}$ can be derived from the quaternion representation of the orientation of frame $\{a\}$ with respect to $\{b\}$, $\boldsymbol{q} = [\eta, \epsilon_1, \epsilon_2, \epsilon_3]$, according to Equation 6.2.

$$\boldsymbol{R_a^b}(\boldsymbol{q}) = \begin{bmatrix} 1-2(\epsilon_2^2+\epsilon_3^2) & 2(\epsilon_1\epsilon_2-\epsilon_3\eta) & 2(\epsilon_1\epsilon_3+\epsilon_2\eta) \\ 2(\epsilon_1\epsilon_2+\epsilon_3\eta) & 1-2(\epsilon_1^2+\epsilon_3^2) & 2(\epsilon_2\epsilon_3-\epsilon_1\eta) \\ 2(\epsilon_1\epsilon_3-\epsilon_2\eta) & 2(\epsilon_2\epsilon_3+\epsilon_1\eta) & 1-2(\epsilon_1^2+\epsilon_2^2) \end{bmatrix} \tag{6.2}$$

Position Transformation

With $\boldsymbol{q}_{r/n}$ and $\boldsymbol{q}_{t/n}$ measured in Qualisys, $\boldsymbol{R}_r^n$ and $\boldsymbol{R}_t^n$ are calculated with Equation 6.2. Since $\{o\}$ and $\{c\}$ are translated and not rotated in $\{r\}$ and $\{t\}$, respectively, the rotation matrices remain unchanged yielding $\boldsymbol{R}_o^n = \boldsymbol{R}_t^n$ and $\boldsymbol{R}_c^n = \boldsymbol{R}_r^n$. The position of the origin of the $\{o\}$ frame is defined in the $\{t\}$ frame as $\boldsymbol{p}_{o/t}^t$, and the position of $\{c\}$ is defined in $\{r\}$ as $\boldsymbol{p}_{c/r}^r$. The relative position between the camera and the object, $\boldsymbol{p}_{o/c}^c$, is found with the relation in Equation 6.3.

$$\boldsymbol{p}_{o/c}^c = \boldsymbol{R}_c^{nT}(\boldsymbol{p}_{t/n}^n + \boldsymbol{R}_t^n \boldsymbol{p}_{o/t}^t - \boldsymbol{p}_{r/n}^n) - \boldsymbol{p}_{c/r}^r \tag{6.3}$$

Attitude Transformation

The relative rotation between two quaternions, $\boldsymbol{q}_a$ and $\boldsymbol{q}_b$, can be found from equation Equation 6.4.

$$\boldsymbol{q}_{b/a} = \boldsymbol{q}_a^{-1} * \boldsymbol{q}_b \tag{6.4}$$

In Equation 6.4, the inverse ($^{-1}$) and multiplication ($*$) operations refer to the quaternion inverse and quaternion multiplications. The definitions of these operations are not included in this thesis, for further reading see [66].

With the Qualisys measurements $\boldsymbol{q}_{r/n} = \boldsymbol{q}_{c/n}$ and $\boldsymbol{q}_{t/n} = \boldsymbol{q}_{o/n}$, the relative rotation of the object with respect to the the camera $\boldsymbol{q}_{o/c}$ is found with Equation 6.5.

$$\boldsymbol{q}_{o/c} = \boldsymbol{q}_{c/n}{}^{-1} * \boldsymbol{q}_{o/n} \tag{6.5}$$

The relative pose vector, $\boldsymbol{\eta}_{o/c}^c$, serving as the label for each image is finally obtained according to Equation 6.6.

$$\boldsymbol{\eta}_{o/c}^c = [\boldsymbol{p}_{o/c}^c{}^T, \boldsymbol{q}_{o/c}{}^T]^T \tag{6.6}$$

6.3 Data Processing

This section describes the necessary data processing to obtain the labeled datasets from the raw MC-lab data. Recall from Section 6.1.2 that the data generated in the MC-lab are stored in bag files. Two bag files were created for each recording, one containing the pose measurements from Qualisys and one containing the images. It was therefore necessary to convert the data from bag files, match each image with the corresponding Qualisys measurements, and calculate the relative poses from these measurements. The final labeled datasets were stored in HDF5 format, introduced in Section 4.5. The data processing procedure is described in the following.

First, the data is read from the bag files and written into HDF5 files. An HDF5 file can be organized in groups, where each group can contain either subgroups or datasets. An HDF5 dataset is a single array of data elements, such as a pose vector or an image in this case. This is not to be confused with the labeled datasets $\mathcal{D}$ described in this chapter, as an HDF5 dataset corresponds to one single data point. Each dataset can also be assigned an attribute. The HDF5 files for the lab data are organized into three groups, one for the images, one for the ROV poses, $\boldsymbol{\eta}^n_{r/n}$, and one for the template poses, $\boldsymbol{\eta}^n_{t/n}$. Each of these groups contains thousands of datasets, being images in the first case, and pose vectors in the two latter cases. All datasets are assigned an attribute representing time, storing the time slot in seconds in which the data point was measured.

Second, each image is matched with its corresponding pose measurement, and the relative pose between the object and the camera is calculated from this. Recall from Section 6.1 that images and pose measurements were sampled at different frequencies. Images were sampled at a frequency of 15 Hz, while pose measurements were sampled at a frequency of 25 Hz. This implies that the maximum time difference between an image data point and the closest pose data point should be $\frac{25^{-1}}{2} = 0.02$ seconds. Each image is therefore matched with a pose measurement that is sampled within a time interval of 0.02 seconds. When an image is assigned an ROV pose, $\boldsymbol{\eta}^n_{r/n}$, the relative pose between the object and the camera, $\boldsymbol{\eta}^c_{o/c}$, can be calculated. Since the template has a fixed position, the median of all the measured template poses is used for the template pose, $\boldsymbol{\eta}^n_{t/n}$, for all images. Further, the position of the camera on the ROV, $\boldsymbol{p}^r_{c/r}$, and the position of the object on the template, $\boldsymbol{p}^t_{o/t}$, is fixed, and written directly into the script for processing. The relative pose is then calculated according to Equation 6.3, Equation 6.5 and Equation 6.6, and serves as the label for each image. The resulting labeled dataset is stored in HDF5 format.

6.4 Datasets

The real-world datasets were obtained from two different sessions in the MC-lab. Two different target objects were used, and mounted on two different templates yielding different backgrounds for the images. In particular, an AruCo marker and a plastic ring modeling a subsea value, were used as the target objects. These objects can be seen in Figure 6.4, and they were mounted on either a the metal stick, seen in Figure 6.1, or the yellow metal box seen in Figure 6.5. The lighting conditions in the MC-lab were also varied for some of the datasets.

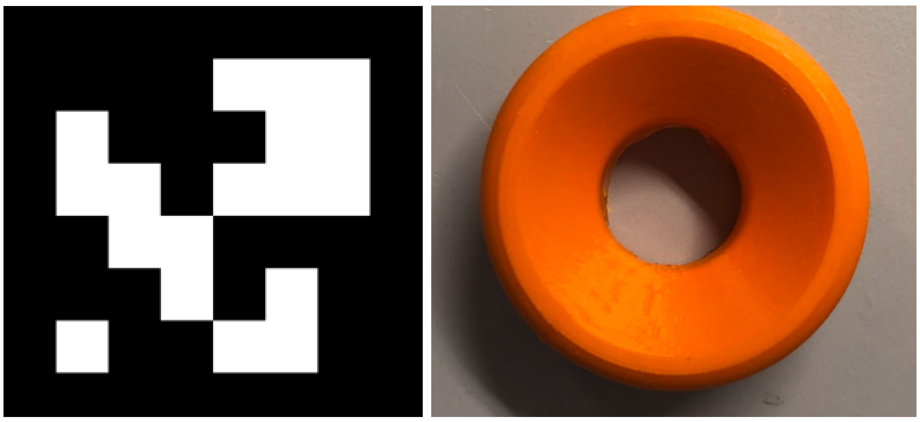

(a) (b)

Figure 6.4: Target objects used for the real-world datasets: AruCo marker (a) and ring object modeling a subsea valve (b)

In the first session, only one dataset was recorded. An AruCo marker was used as the target object, and was mounted on the yellow metal box serving as the template. The lights in the MC-lab were kept on during this recording. The AruCo marker is the same as the one used in the simulated dataset described in Chapter 5, and can be seen in Figure 6.4a. The AruCo marker on the yellow template box can be seen in Figure 6.5, where the AruCo marker of interest is placed in the upper right corner.

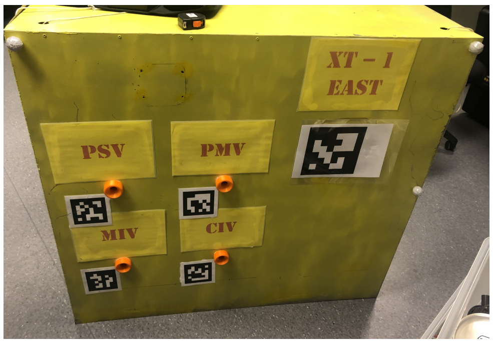


Figure 6.5: AruCo marker attached to the yellow metal box serving as a background template for underwater images

In the second laboratory session, five different datasets were recorded. First, the same AruCo marker as in the first session was used as the target object, but now mounted on a metal stick yielding a different background. Three datasets with this setup were recorded, with varying light conditions. There are two rows of light over the basin in the MC-lab. For the first recording, the lights on both rows were kept on, similar to what was done in the first lab session. In the next recording, the lights on the row furthest away from the target object were turned off, from now on referred to as *reduced light A*. In the third recording, the lights on the row closest to the target object were turned on, from now on referred to as *reduced light B*.

For the next datasets, the ring object of diameter 5cm, as seen in Figure 6.4b, was used as the target object. The ring was first mounted on the metal stick, and later mounted on the yellow metal box from the first session. This was done to obtain images with different backgrounds, similar to what was done with the AruCo marker. For the datasets with the mock-up valve being the target object, the lights in the MC-lab were kept on.

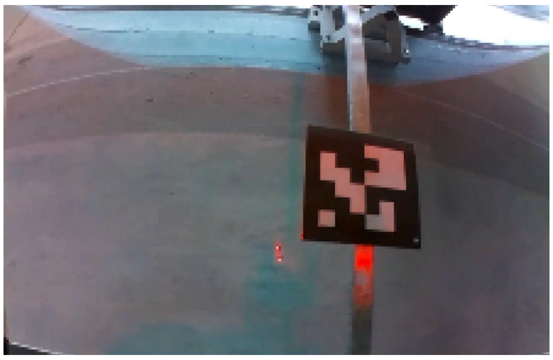
The details of all the datasets obtained in the MC-lab are listed in Table 6.1. Each dataset is defined by the lab session in which it was recorded, the target object, the template serving as the background, the lighting conditions and the number of images in the dataset. The details of the simulated dataset described in Chapter 5 is also included in Table 6.1.

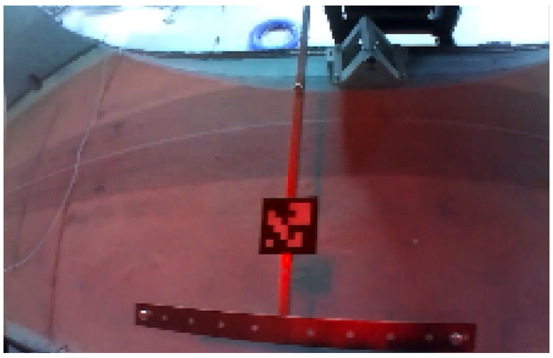
ID	**Lab session**	**Object**	**Background**	**Light conditions**	**# Images**
0	Simulation	AruCo marker	Random image	–	2 000
1	1	AruCo marker	Yellow box	All lights on	11 978
2	2	AruCo marker	Metal stick	All lights on	10 542
3	2	AruCo marker	Metal stick	Reduced light A	3 035
4	2	AruCo marker	Metal stick	Reduced light B	2 813
5	2	Ring object	Metal stick	All lights on	14 045
6	2	Ring object	Yellow box	All lights on	8 594

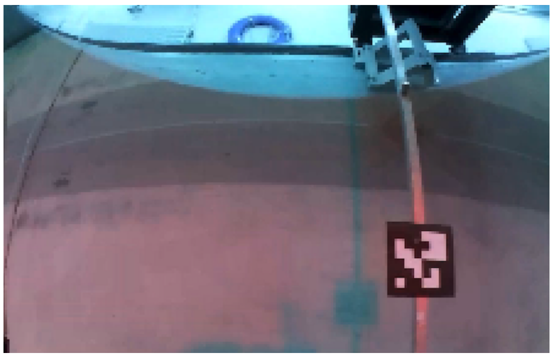
Table 6.1: Properties of simulation and real-world datasets

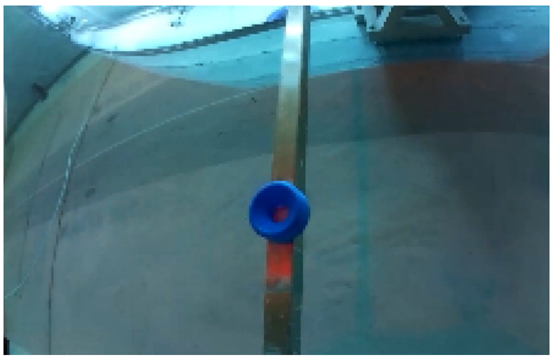
The datasets also varies in terms of the values of the 6-DoF relative poses. For example, the relative distances in dataset 1 are significantly larger compared to the other datasets. Plots of the relative poses for each dataset are attached in Appendix C, where the quaternion $\boldsymbol{q}$ is transformed into the Euler Angles around the primary axes of rotations. Image samples from each of the real-world datasets obtained in the MC-lab are seen in Figure 6.6. This figure reveals that the yellow metal box and the orange valve model both appear in blue on the images. This is likely due to some issues when converting the data from bag files, see further discussion in Section 9.2.

(a) Dataset 1


(b) Dataset 2


(c) Dataset 3


(d) Dataset 4


(e) Dataset 5

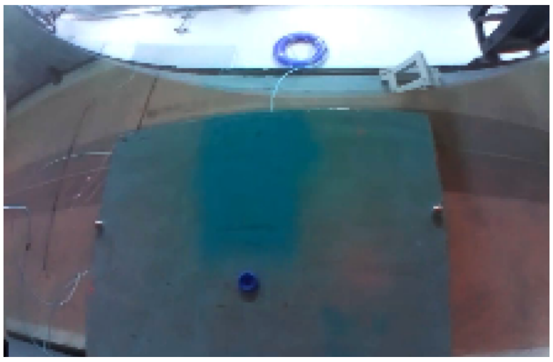

(f) Dataset 6

Figure 6.6: Sample images from all real-world datasets produced in the MC-lab

Chapter 7

Performance

The performance of the network under different conditions is evaluated based on the measurements listed below.

1. **Euclidean translation error**: The Euclidean distance between true and predicted relative position, also referred to as the mean squared error. The Euclidean translation error is defined in in Equation 7.1, where $\boldsymbol{p}$ and $\hat{\boldsymbol{p}}$ are the true and predicted position, respectively.

$$\text{error}_{\text{euc}} = ||\hat{\boldsymbol{p}} - \boldsymbol{p}||_2 \tag{7.1}$$

2. **Relative angle error from quaternion**: The relative angle between the true and predicted relative orientation. When representing these orientations with quaternions, $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$ respectively, the quaternion error between them can be found from Equation 7.2 [34].

$$\delta\boldsymbol{q} = \boldsymbol{q} * \hat{\boldsymbol{q}} = [\delta q_1, \delta q_2, \delta q_3, \delta q_4] \tag{7.2}$$

   In Equation 7.2, the $*$ operator refers to the quaternion multiplication [66]. The relative angle error between $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$ can then be found from Equation 7.3 [66].

$$\text{error}_{\text{rel\_angle}} = 2\text{arccos}(|\delta q_1|) \tag{7.3}$$

3. **6-DoF errors**: The difference between true and predicted pose in all 6-DoF, as defined in Equation 7.4. Since orientation is presented with quaternions $\boldsymbol{q}$, it is transformed into the Euler angles, Θ, around the primary axes of rotations to obtain

the 6-DoF errors. The pose vector with orientation represented as Euler angles is noted $\boldsymbol{\eta}_e$.

$$\text{error}_{\text{diff}} = \boldsymbol{\eta}_e - \hat{\boldsymbol{\eta}}_e \tag{7.4}$$

Here $\boldsymbol{\eta}_e$ and $\hat{\boldsymbol{\eta}}_e$ are the true and predicted pose respectively. The Euler angle errors, being roll, pitch and yaw errors, are wrapped to lie in the range $[-\pi, \pi)$.

4. **10-percent requirement:** The error values should be seen together with the relative distance and orientation between the camera and the target object for each image. This is reasonable as one should expect higher precision for a closeup image, than one taken from a further distance. The Euclidean translation error (metric 1) and the relative angle error (metric 2) can be represented as a percentage amount of the Euclidean distance and the relative angle between the camera and the target object. These amounts are represented by $T_{\%}$ and $A_{\%}$, and can be seen in Equation 7.5.

$$\begin{aligned} T_{\%} &= \frac{\text{error}_{\text{euc}}}{\text{euc\_dist}} &&= \frac{||\hat{\boldsymbol{p}} - \boldsymbol{p}||_2}{||\boldsymbol{p}||_2} \\ A_{\%} &= \frac{\text{error}_{\text{rel\_angle}}}{\text{rel\_angle}} &&= \frac{\arccos(|\delta q_1|)}{\arccos(|q_1|)} \end{aligned} \tag{7.5}$$

The amount of test data that yield less than 10% for these two measurements will be measured.

For each train and test scenario presented in Chapter 8, the performance of the network was assessed based on the measurements listed above.

Chapter 8

Iterations and Results

With the available datasets listed in Table 6.1 and the implemented model described in Chapter 4, the model was trained and tested in several iterations. In each iteration, the network was trained on a training dataset according to Algorithm 3, and validated on a test dataset. Different train and test configurations were carried out based on the provisional results and the available data at the current time in the process. Each train and test case differs in terms of which data the training and test datasets consist of, and how these were composed of the available datasets. The process started by evaluating the model with simulated data of the artificial AruCo marker, before considering real-world data of the same marker. Finally, the real-world data of a ring object modeling a subsea valve, was used to evaluate the model.

The first section of this chapter gives a brief outline of the iterative process of assessing the network. The main results are then presented in Section 8.2. Additional results can be seen in Appendix D. More than 70 different train and test cases were investigated in this process, and only the most relevant cases are included in the following.

8.1 Process Outline

The iterative process of training and testing the network was grouped into 4 main steps, illustrated in Figure 8.1. The circles in this figure represent the production of more datasets. The contents of each step are described in the following.

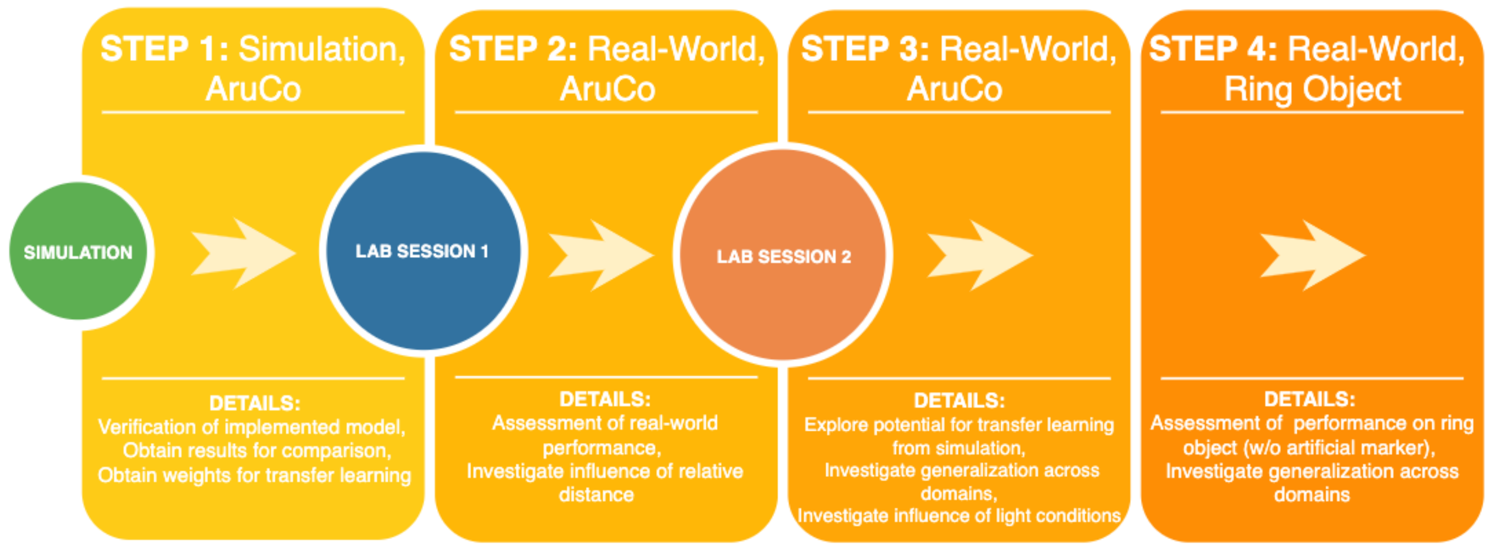

STEP 1: Simulation, AruCo
STEP 2: Real-World, AruCo
STEP 3: Real-World, AruCo
STEP 4: Real-World, Ring Object
SIMULATION
LAB SESSION 1
LAB SESSION 2
DETAILS: Verification of implemented model, Obtain results for comparison, Obtain weights for transfer learning
DETAILS: Assessment of real-world performance, Investigate influence of relative distance
DETAILS: Explore potential for transfer learning from simulation, Investigate generalization across domains, Investigate influence of light conditions
DETAILS: Assessment of performance on ring object (w/o artificial marker), Investigate generalization across domains

Figure 8.1: Outline of iterative assessment process

Step 1: Simulation Data with AruCo

To begin with, the model was trained and tested on the simulated data described in Chapter 5. This was done to verify the performance implemented model, and to obtain results to be used for comparison when assessing the performance of the network with real-world data. As the images from simulation are less noisy and considered easier to evaluate by the network than real-world images, this experiment was conducted to give insight into the potential of the network. It was considered likely that the accuracy of the pose estimation in this experiment would serve as an upper limit for the accuracy to expect from real-world experiments.

The network weights obtained from training on simulation data was also intended to be tested on real-world data at later stages. The purpose of this was to investigate the opportunity for the network to transfer its knowledge from the simulation domain to the real-world domain. This would be of high interest if the localization system was to be applied in industry, as labeled real-world training data from subsea facilities are unavailable.

Step 2: Real-World Data with AruCo

The next step was to both train and test the model on the real-world data of the AruCo marker. To begin with, this was done on the dataset produced in the first lab session, containing images of the AruCo marker mounted on the yellow metal box, corresponding to dataset 1 in Table 6.1. This was done in different configurations, with training and testing sets consisting of images from various subsets of the dataset.

The training and test sets were composed in two different ways. They were first composed of random data points from the entire dataset, discarding the sequenced order of the dataset, from now on referred to as *random split*. The alternative approach was to keep the sequence in which the data was recorded, and split the data in an ordered manner. Since the images were sampled at a frequency of 15 Hz, the sequenced order of the datasets represents a temporal encoding. Given one of the available datasets in Table 6.1, the first X number of images were put in the train set, and the following Y number of images were put in the test set. By splitting the dataset into train and test sets this way, the temporal encoding of the data is maintained when training and testing the network. From now on this is referred to as *sequential split*.

In addition, dataset 1 was also filtered by distance in this step. The network was trained and tested on different subsets of the dataset, with the relative distance between the camera and the target object lying within specific value ranges. This was done to investigate how the performance of the network was influenced by the relative distance.

Step 3: Real-World Data with AruCo

Based on the provisional results on the data from the first lab session, a second lab session was planned and completed. With more real-world data available, more train and test scenarios were investigated. Continuing the work on the images of the AruCo marker, the datasets 2, 3 and 4 in Table 6.1 were relevant. Recall from Section 6.4 that these sets contain images of the AruCo marker mounted on a metal stick under three different lighting conditions. The ROV with the camera was kept closer to the target object in this second session. These datasets are considered to be more similar to each other in terms of both the image contents and the value range of poses, compared to dataset 1 from the first lab session. For more details, plots of the relative poses for each dataset can be seen in Appendix C. The network was trained and tested on these three datasets, both separated and concatenated. These experiments were conducted to continue the model evaluation, and investigate the influence of the lighting conditions in the MC-lab on the network performance. All train and test configurations were done with both random and sequential split.

Further, the network trained on the data of the AruCo marker on the metal stick (datasets 2, 3, 4) was tested on the images of the AruCo marker on the yellow box (dataset 1) and vice versa. The network was also trained on one huge dataset containing images of the AruCo marker from both domains, and tested on the datasets with different backgrounds. These assessments were done to investigate how the network generalizes its knowledge between different domains. The network trained on the simulated data in step 1, was also tested on the real-world data of the AruCo marker in this step.

Step 4: Real-World Data with Ring Object

After assessing the network performance in multiple different train and test configurations with images of the AruCo marker, the network performance was finally evaluated on the datasets with images of the ring object. These train and test cases were done to investigate the network's performance on a natural landmark, and compare it to its performance on the artificial AruCo marker. The datasets 5 and 6 in Table 6.1 were relevant in this step, containing images of the ring mounted on the metal stick and the yellow box, respectively. The network was trained on these datasets both separated and concatenated, with both random and sequential splitting of the data.

Similar to what was done in step 3, the network performance was assessed across the two domains. That is, the network trained on the images of the ring on the stick (dataset 5 in Table 6.1) was tested on the images of the ring on the box (dataset 6 in Table 6.1) and vice versa.

8.2 Results

This section presents the main results and findings from the train and test iterations carried out in the process described in the previous section. To keep this section reasonably short, detailed results are only presented for four selected cases. Each case is described by the properties of the train and test sets used to assess the network. This includes the IDs of the datasets (referring to Table 6.1) they are composed of, how this data is split into train and test sets (random or sequential), and the size of each set. For each case, the performance according to the four metrics described in Chapter 7 are presented.

It is emphasized that the size of the Euclidean translation error and the relative angle error always will be large compared with the corresponding 6-DoF errors in translation and rotation directions, respectively. This is reasonable as the two former metrics describe the total error for translation and rotation. The errors in each DoF are rarely zero at the same time, which is why the Euclidean translation error and the relative angle error seldom reaches zero. These measurements are therefore more conservative than the individual 6-DoF errors.

Detailed results are presented in the following for four different selected cases. Aligned with the process outline described in the previous section, one result with the simulation data and one result with the real-world data of the AruCo marker are presented. Following this, results for two different cases with the ring object are presented, one with random split of datasets and one with sequential split of datasets. These four cases are listed below.

- **Case A**: Results on simulation data, Section 8.2.1
- **Case B**: Results on real-world data with AruCo marker, Section 8.2.2
- **Case C**: Results on real-world data with ring object (random split), Section 8.2.3
- **Case D**: Results on real-world data with ring object (sequential split), Section 8.2.4

In addition to these cases, a statistical summary of other relevant results is presented in Section 8.2.5.

8.2.1 Case A: Simulated Data with AruCo

This section presents the results obtained from training and testing the model on simulated data with images of the AruCo marker, as described in Chapter 5. Table 8.1 lists details of the train and test datasets, and Figure 8.2 illustrates a sample image from the dataset.

Figure 8.2: Sample image from dataset utilized in case A

Data ID	**Description**	**# Train images**	**# Test images**
0	Simulated data with AruCo marker	1 600	400

Table 8.1: Case A: Details of train and test datasets

Figure 8.3 shows histograms of the relative angle error and the Euclidean translation error, presenting the model's overall performance for estimating attitude and position, respectively. The median, mean and standard deviation for these measurements are listed in Table 8.2. The difference between the mean and median values is small for both measurements, and the standard deviations are small. This is also confirmed in the histograms, where no outliers of significantly large magnitude are observed.

Measurement	**Median**	**Mean**	**Standard deviation**
Relative angle error (from quaternions) [deg]	0.628	0.777	0.624
Euclidean translation error [mm]	19.23	20.63	10.00

Table 8.2: Case A: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

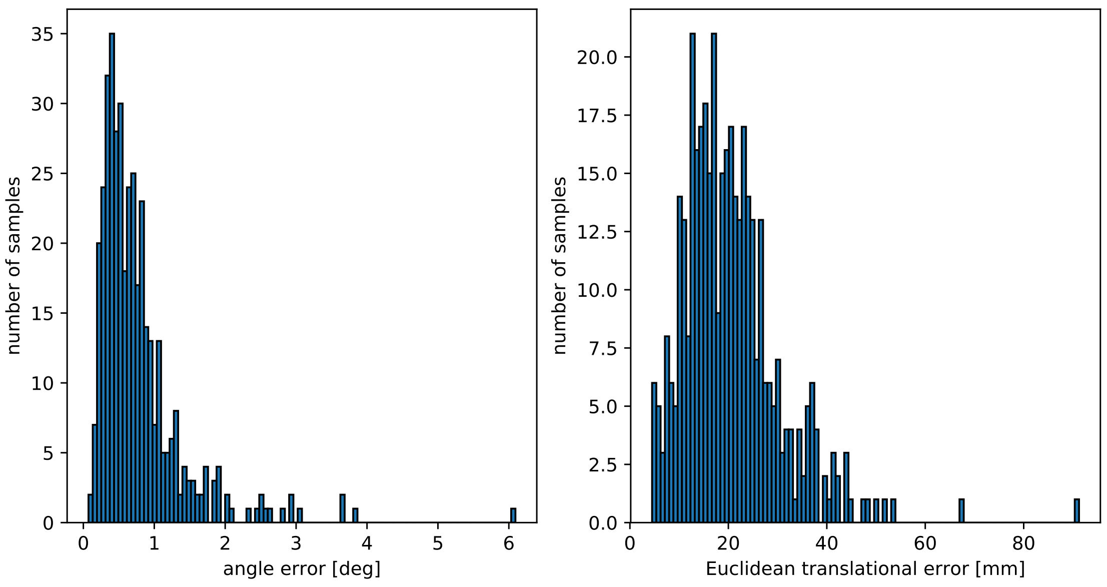


Figure 8.3: Case A: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

Figure 8.4 and Figure 8.5 show histograms and plots, respectively, for the pose errors in all 6-DoF; roll, pitch, yaw, surge, sway and heave direction. The associated mean and standard deviation values are listed in Table 8.3. The errors in all 6-DoF are approximately normally distributed around zero. Regarding attitude, the smallest standard deviation is seen in roll direction, while the mean valued error in pitch direction is closest to zero. For the translational errors, the largest errors are seen in surge direction, where the standard deviation is nearly twice as large compared with sway and heave. However, the errors in all 6-DoF are considered small in this case.

DoF	**Mean**	**Standard deviation**
Roll [deg]	0.132	0.094
Pitch [deg]	0.099	0.707
Yaw [deg]	-0.157	0.659
X [mm]	7.02	15.9
Y [mm]	-3.62	9.29
Z [mm]	-7.48	8.06

Table 8.3: Case A: Summary statistics of pose errors in each DoF

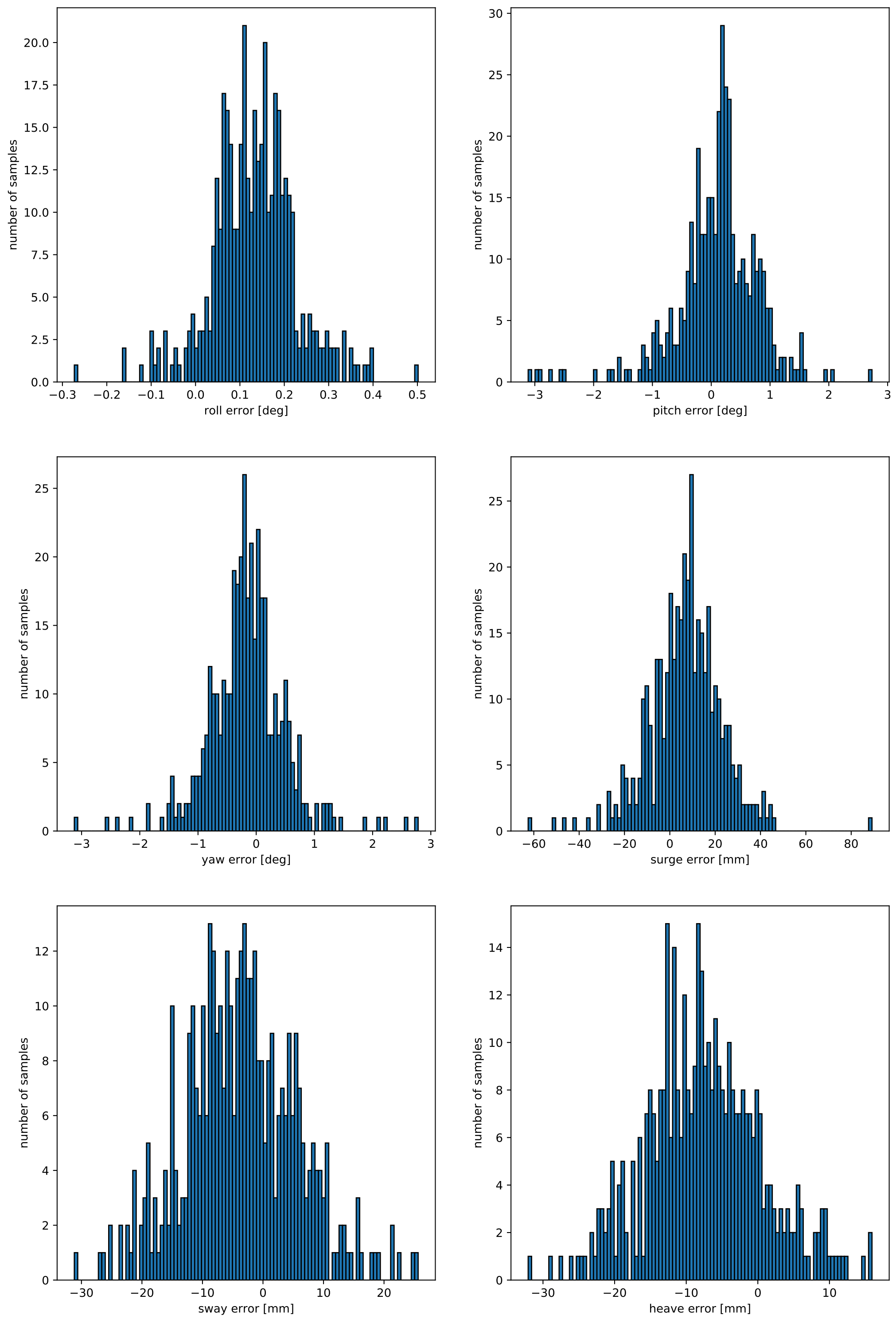


Figure 8.4: Case A: Pose errors in each DoF

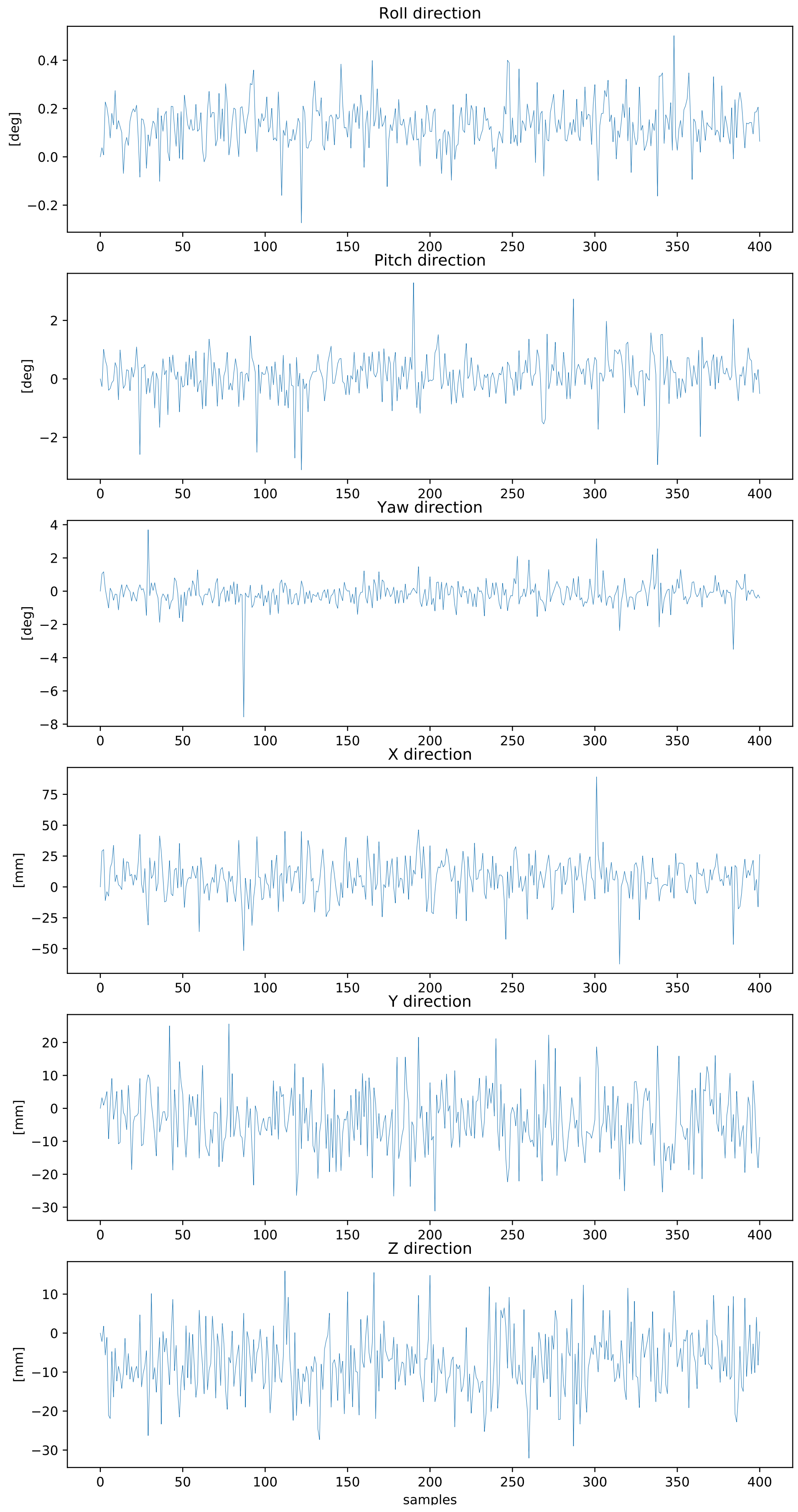


Figure 8.5: Case A: Pose errors in each DoF over samples in test set

Table 8.4 lists the amounts of the test data satisfying the 10-percent requirements. The 10-percent requirements are well satisfied in this case.

Measurement	**Amount**
Translation errors less than 10 % of distance, $T_{\%}$	98 %
Rotation errors less than 10 % of rotation, $A_{\%}$	97 %

Table 8.4: Case A: Amount of test data satisfying the 10-percent requirements

8.2.2 Case B: Real-World Data with AruCo

This section presents one of the results obtained from training and testing the model on real-world data with images of the AruCo marker. Datasets from the second lab session were used in this case, with images of the AruCo marker mounted on the metal stick. Table 8.5 lists the details of the train and test datasets used in this particular case, and Figure 8.6 illustrates a sample image from the datasets.

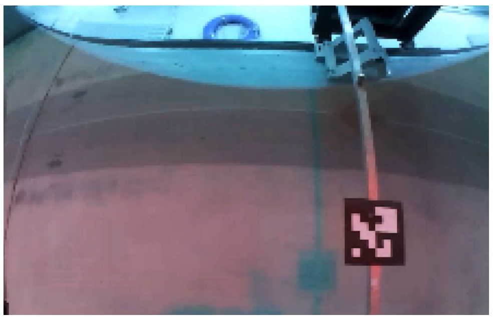


Figure 8.6: Sample image from datasets utilized in case B

Data ID	**Description**	**# Train images**	**# Test images**	**Splitting**
2,4	AruCo marker on metal stick	10 500	2 798	Random

Table 8.5: Case B: Details of train and test datasets

Figure 8.7 shows histograms of the relative angler error and the Euclidean translation error, presenting the model's overall performance for estimating attitude and position, respectively. The median, mean and standard deviation for these measurements are listed in Table 8.6. The errors are larger than what was seen in case A, but the median values for both errors are still considered small. The differences between the median and mean values are larger than what was seen in case A, as is the standard deviations. This relates to the outliers of significant large values for both the relative angle errors and the Euclidean translation errors, seen in the histograms in Figure 8.7. The majority of the errors are however in the small end of the scale.

Measurement	**Median**	**Mean**	**Standard deviation**
Relative angle error (from quaternions) [deg]	0.578	1.209	4.058
Euclidean translation error [mm]	26.52	38.33	68.64

Table 8.6: Case B: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

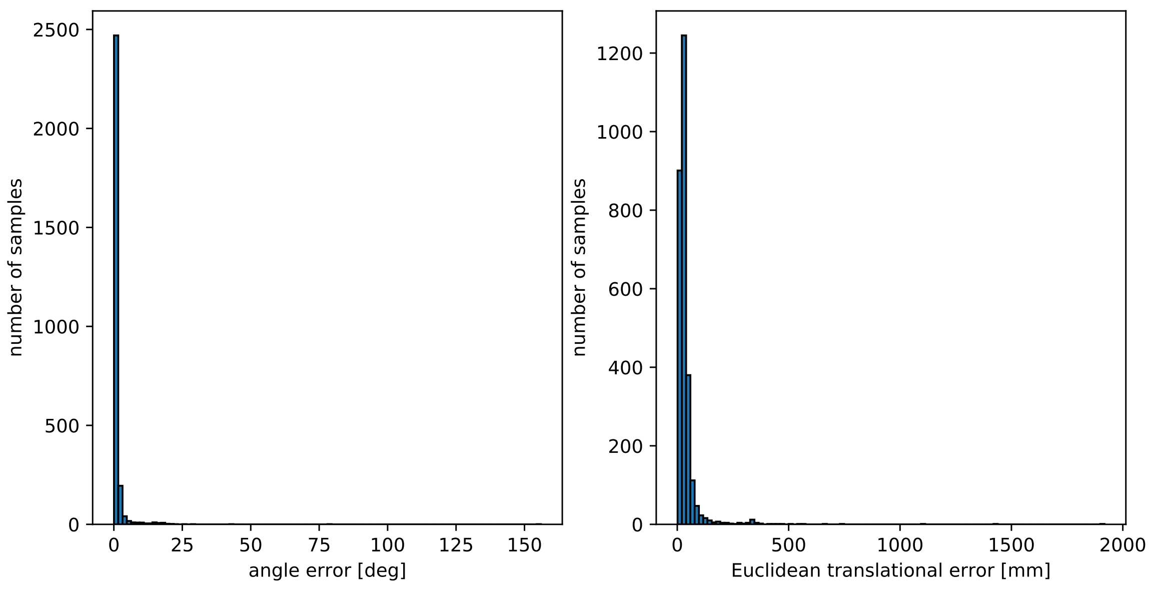


Figure 8.7: Case B: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

Figure 8.8 and Figure 8.9 show histograms and plots, respectively, for the pose errors in all 6-DoF. The corresponding mean and standard deviation values are listed in Table 8.7. It can be seen from the histograms in Figure 8.8 that the errors in all 6-DoF are approximately normally distributed around zero. Outliers of large sizes are observed in all DoFs except pitch direction. It is also observed from Figure 8.9 that the large outliers occur in multiple DoFs for the same samples. For the attitude errors, the largest standard deviation occurs in yaw direction. The largest translational errors are seen in sway direction, where both the standard deviation and the mean values are largest. The mean values are however small for all 6-DoF, and the majority of the errors are close to the mean values. When disregarding the outliers, the errors lie in the approximate same value range as the errors in case A seen in Figure 8.4.

DoF	**Mean**	**Standard deviation**
Roll [deg]	-0.040	2.326
Pitch [deg]	0.262	1.605
Yaw [deg]	0.122	3.859
X [mm]	-3.72	46.52
Y [mm]	10.42	50.19
Z [mm]	4.47	36.82

Table 8.7: Case B: Summary statistics of pose errors in each DoF

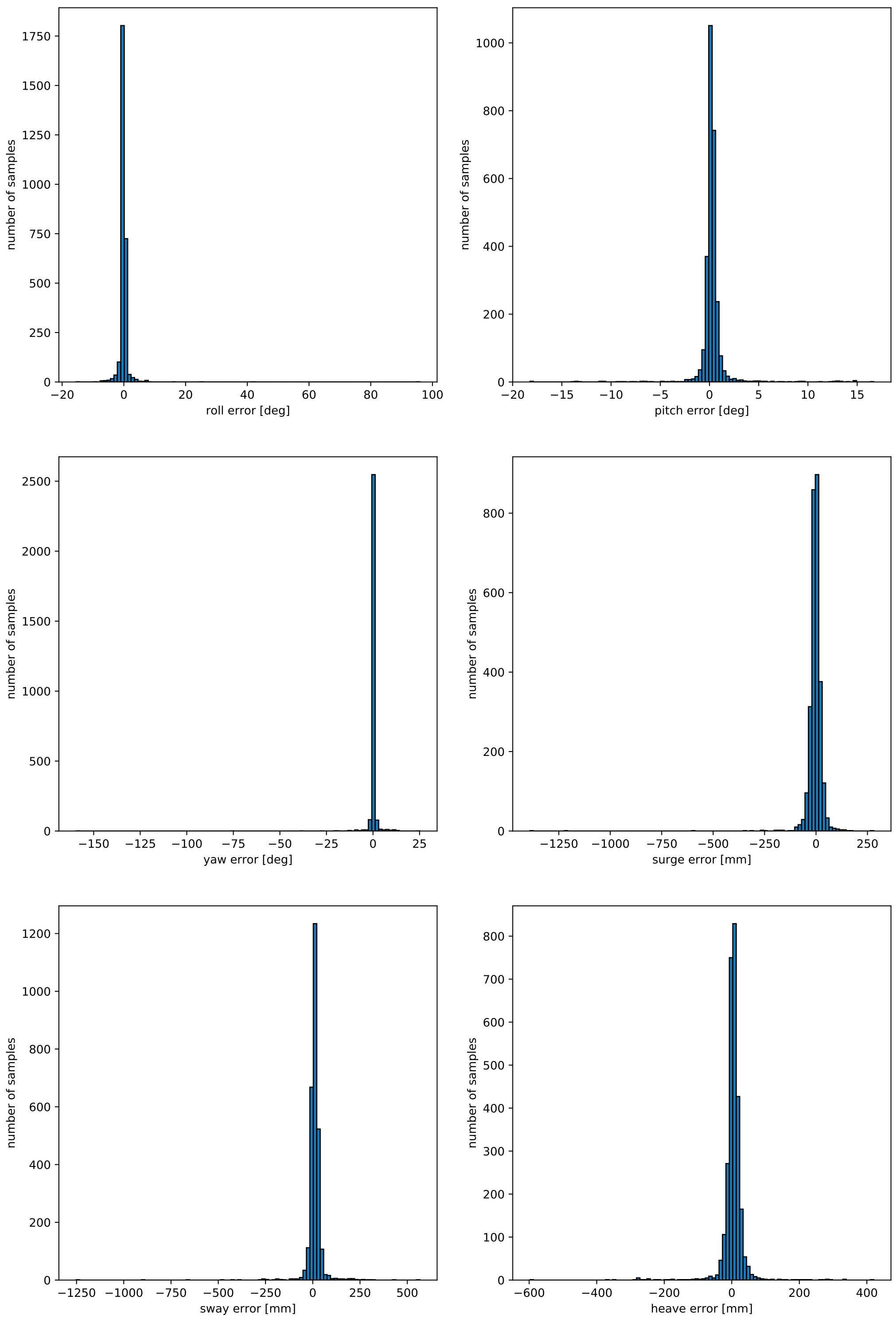


Figure 8.8: Case B: Pose errors in each DoF

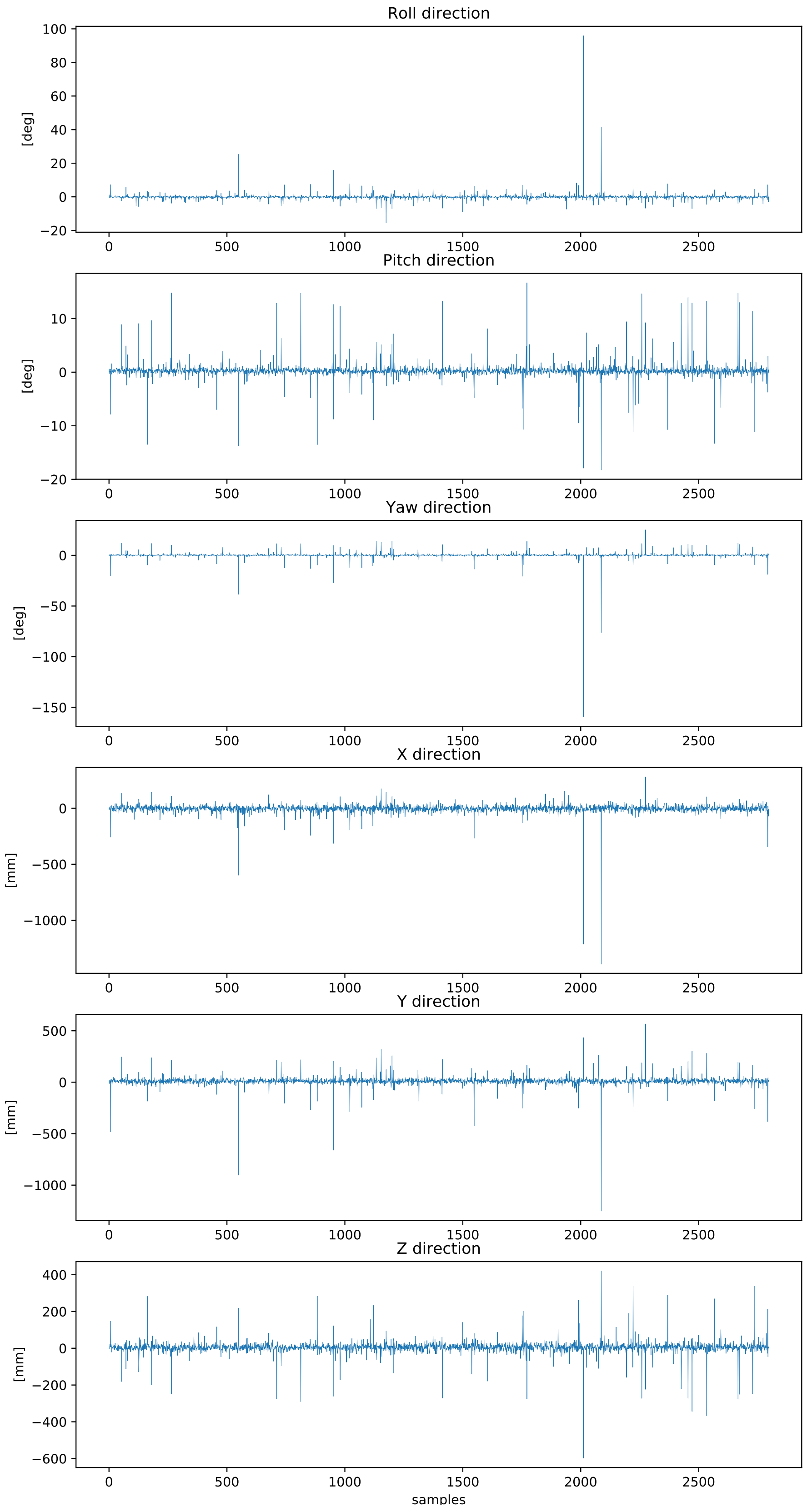


Figure 8.9: Case B: Pose errors in each DoF over samples in test set

Table 8.8 lists the amounts of the test data satisfying the 10-percent requirements, revealing that the requirements are satisfied well. The amounts of test data satisfying the requirements are however reduced from case A, especially regarding the rotation requirement.

Measurement	**Amount**
Translation errors less than 10 % of distance, $T_{\%}$	94 %
Rotation errors less than 10 % of rotation, $A_{\%}$	81 %

Table 8.8: Case B: Amount of test data satisfying the 10-percent requirements

8.2.3 Case C: Real-World Data with Ring Object (Random)

This section presents one of the results obtained from training and testing the model on real-world data with images of the ring object. One of the datasets from the second lab session was used in this case, with images of the ring object mounted on the metal stick. Table 8.9 lists the details of the train and test datasets used in this particular case, and Figure 8.10 illustrates a sample image from the dataset.

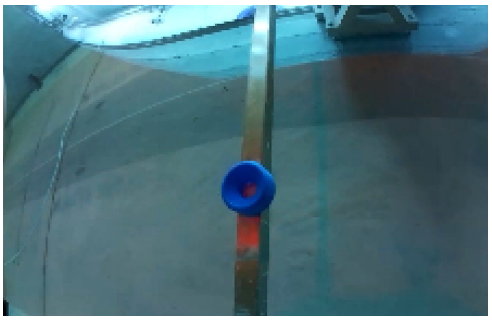


Figure 8.10: Sample image from dataset utilized in case C

Data ID	**Description**	**# Train images**	**# Test images**	**Splitting**
5	Ring object on metal stick	11 000	2 888	Random

Table 8.9: Case C: Details of train and test datasets

Figure 8.11 shows histograms of the relative angle error and the Euclidean translation error, presenting the model's overall performance for estimating attitude and position, respectively. The median, mean and standard deviation for these measurements are listed in Table 8.10. The median values are considered small, and are smaller than the corresponding errors in case B. The median of the rotation error is even lower than what was obtained in the simulation case (case A), and the translation error is approximately similar. The standard deviations of the errors are reduced from case B, while they are still larger than in case A. Correspondingly, the errors exhibit some outliers, seen in Figure 8.11, also in this case.

Measurement	**Median**	**Mean**	**Standard deviation**
Relative angle error (from quaternions) [deg]	0.381	0.721	3.017
Euclidean translation error [mm]	19.16	23.88	33.86

Table 8.10: Case C: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

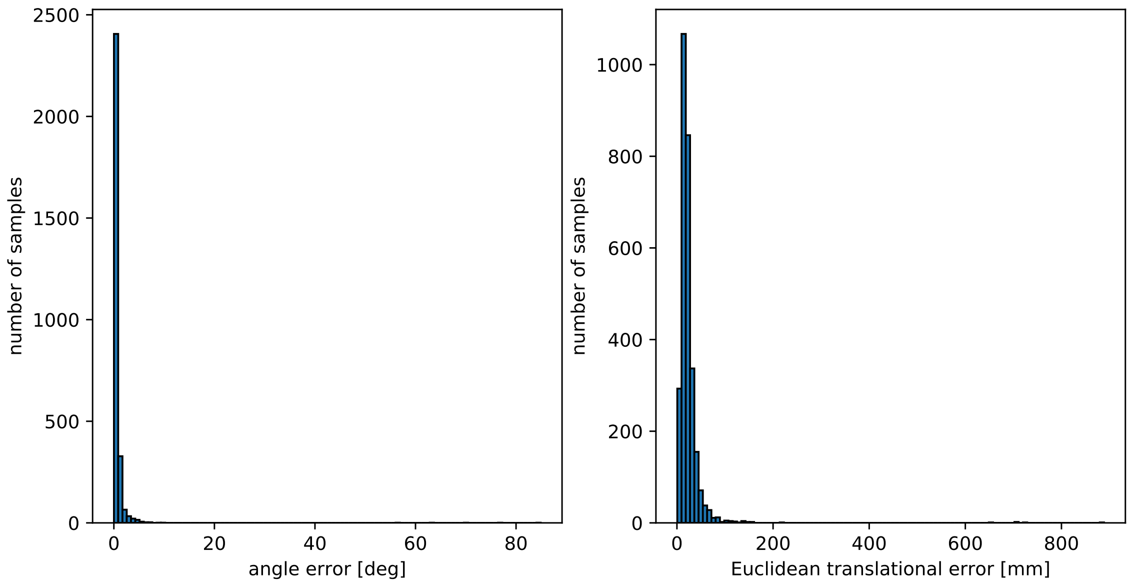


Figure 8.11: Case C: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

Figure 8.12 and Figure 8.13 show histograms and plots, respectively, of the pose errors in all 6-DoF, and the corresponding values for mean and standard deviation can be seen in Table 8.11. Also in this case, the errors are approximately normally distributed around zero. For the attitude directions, large outliers are only observed in roll direction, also being the direction with the highest standard deviation. The mean valued errors in roll, pitch and yaw are closer to zero in this case, compared to cases A and B. A few large outliers can be seen in all the translational directions, but both the mean and the standard deviations are pretty similar in these three directions. The outliers thus occur to a smaller extent in this case compared with case B. Again, it is seen in Figure 8.13 that the large outliers trend to occur in multiple DoFs at the same time.

DoF	**Mean**	**Standard deviation**
Roll [deg]	0.002	3.217
Pitch [deg]	0.003	0.429
Yaw [deg]	-0.016	0.414
X [mm]	-8.21	21.26
Y [mm]	5.55	24.17
Z [mm]	-4.58	23.69

Table 8.11: Case C: Summary statistics of pose errors in each DoF

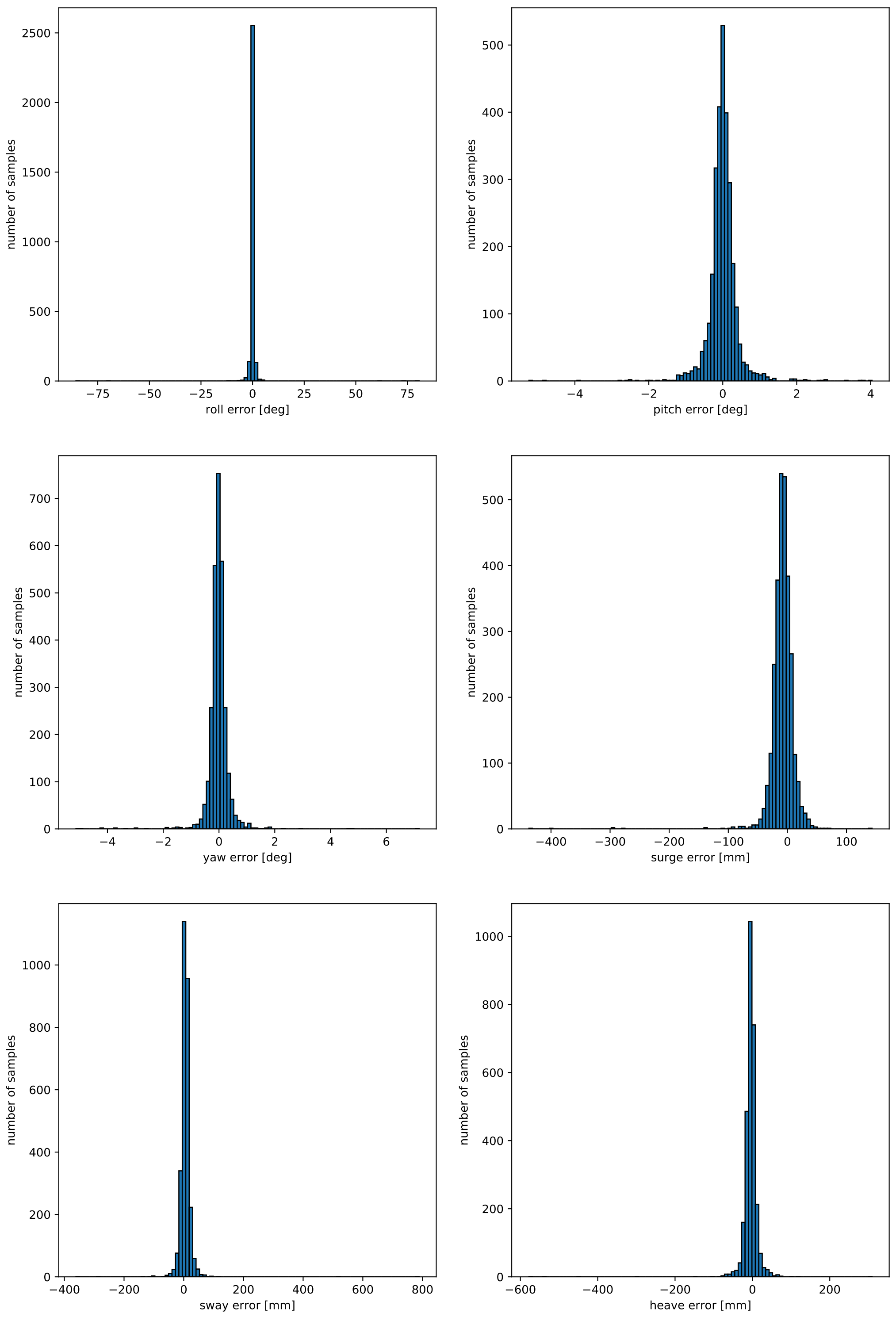


Figure 8.12: Case C: Pose errors in each DoF

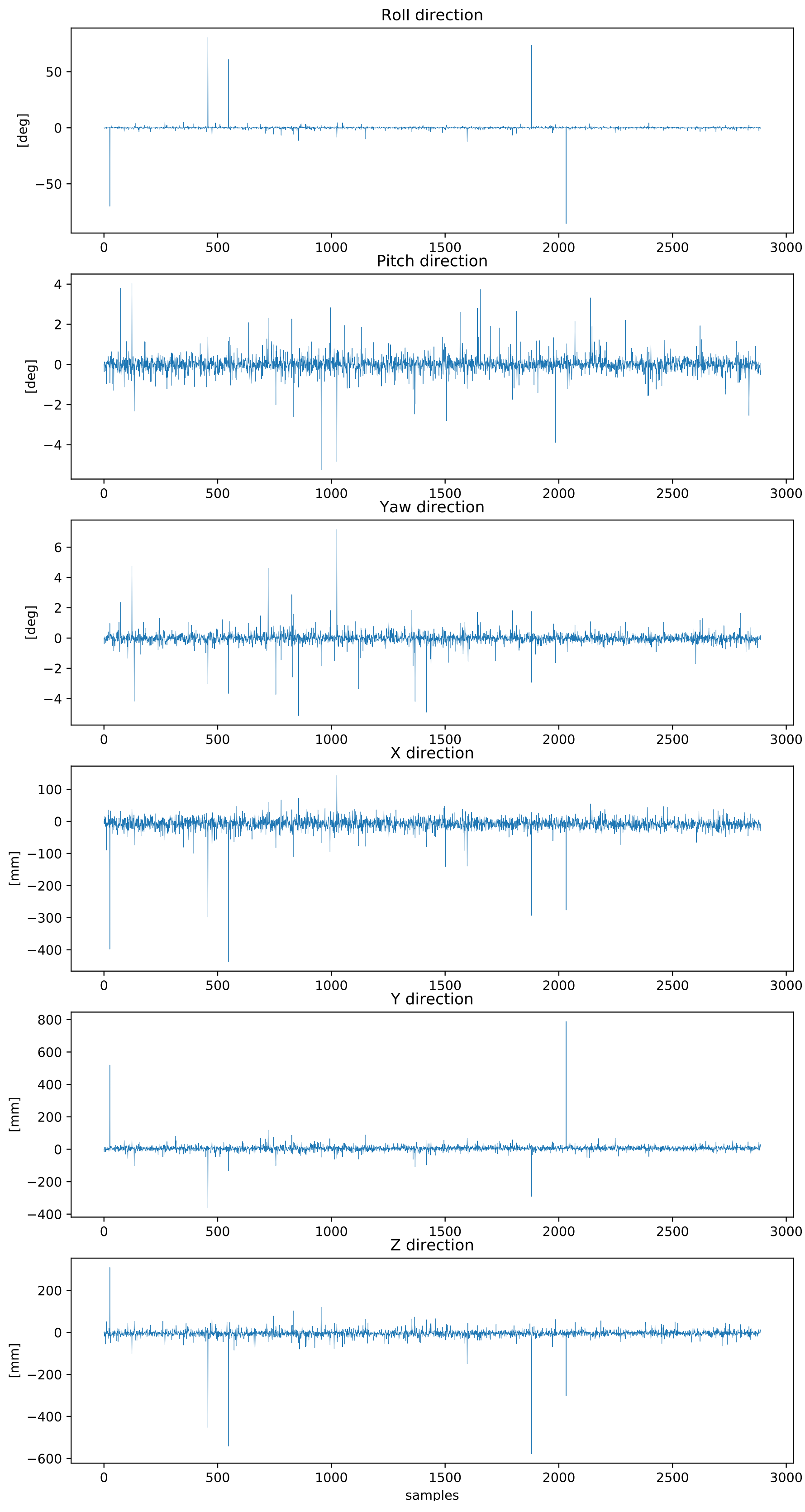


Figure 8.13: Case C: Pose errors in each DoF over samples in test set

Table 8.12 lists the amounts of the test data satisfying the 10-percent requirements. The amounts are increased from case B, but the rotation error requirement is still achieved less satisfactory than the translation error requirement.

Measurement	**Amount**
Translation error less than 10 % of distance, $T_{\%}$	99 %
Rotation error less than 10 % rotation, $A_{\%}$	85 %

Table 8.12: Case C: Amount of test data satisfying the 10-percent requirements

8.2.4 Case D: Real-World Data with Ring Object (Sequential)

This section presents results on one additional case where the network was trained and tested on real-world data with images of the ring object. The dataset used in case C was also used in this case, with images of the ring object mounted on the metal stick. This dataset was sequentially split into train and test sets in this case, in difference from the previous case C, where the dataset was randomly split into these sets. Since this case is based on the same data as case C, the results presented here should be seen in conjunction with the results of case C. Table 8.13 lists the details of the train and test datasets used in this particular case, and Figure 8.14 illustrates a sample image from the dataset.

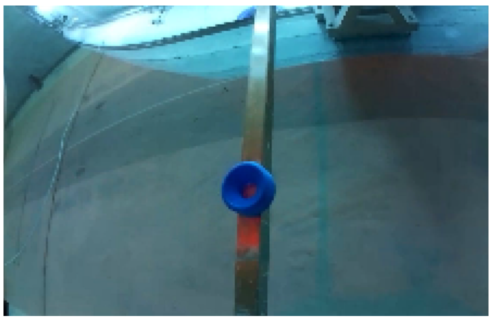


Figure 8.14: Sample image from dataset utilized in case D

Data ID	**Description**	**# Train images**	**#Test images**	**Splitting**
5	Ring object on metal stick	11 000	2 888	Sequential

Table 8.13: Case D: Details of train and test datasets

Figure 8.15 shows histograms of the relative angle error and the Euclidean translation error, presenting the model's overall performance for estimating attitude and orientation, respectively. The median, mean and standard deviation for these measurements are listed in Table 8.14. These errors are larger than what was seen in case C, and the standard deviations, in particular, are significantly larger. The outliers in the error histograms in Figure 8.15 are also larger than what was seen in case C.

Measurement	**Median**	**Mean**	**Standard deviation**
Relative angle error (from quaternions) [deg]	0.856	1.853	11.274
Euclidean translation error [mm]	20.22	27.76	52.29

Table 8.14: Case D: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

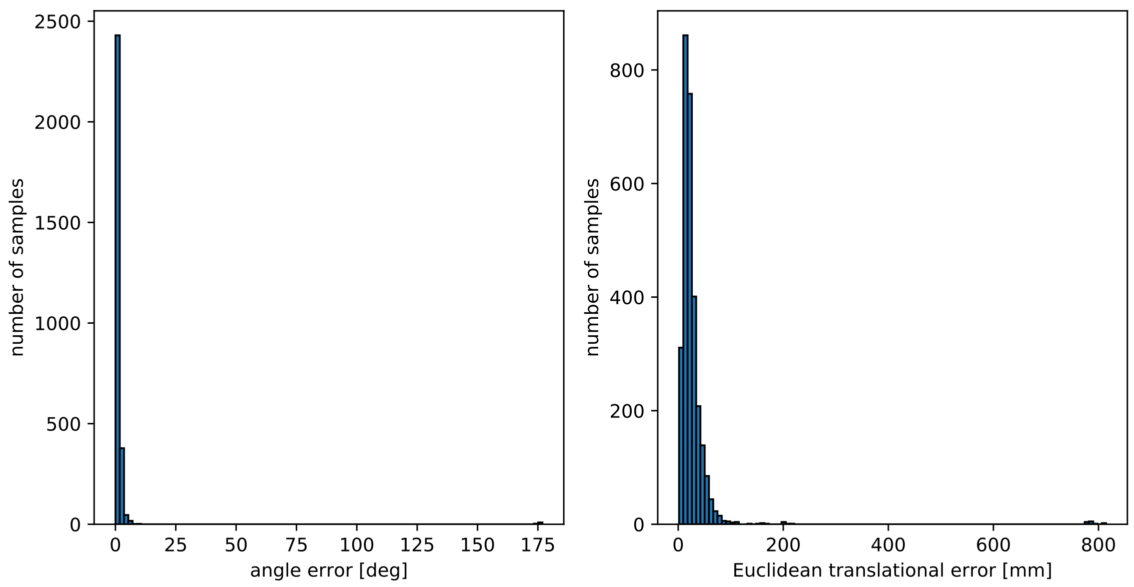


Figure 8.15: Case D: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

Figure 8.16 and Figure 8.17 shows histograms and plots, respectively, for the pose errors in all 6-DoF, and the corresponding values for mean and standard deviation are listed in Table 8.15. These errors are approximately normally distributed around zero. For the attitude directions, large outliers are only observed in yaw direction, also being the direction in which the standard deviation of the error is largest. Among the translation directions, large outliers occur in all directions. It is seen in Figure 8.17 that the largest errors occur in all DoFs for the same samples, similar to the trends observed in cases B and C.

DoF	**Mean**	**Standard deviation**
Roll [deg]	-0.340	1.762
Pitch [deg]	-0.206	1.216
Yaw [deg]	0.636	11.724
X [mm]	7.38	41.27
Y [mm]	7.80	35.44
Z [mm]	3.61	20.43

Table 8.15: Case D: Summary statistics of pose errors in each DoF

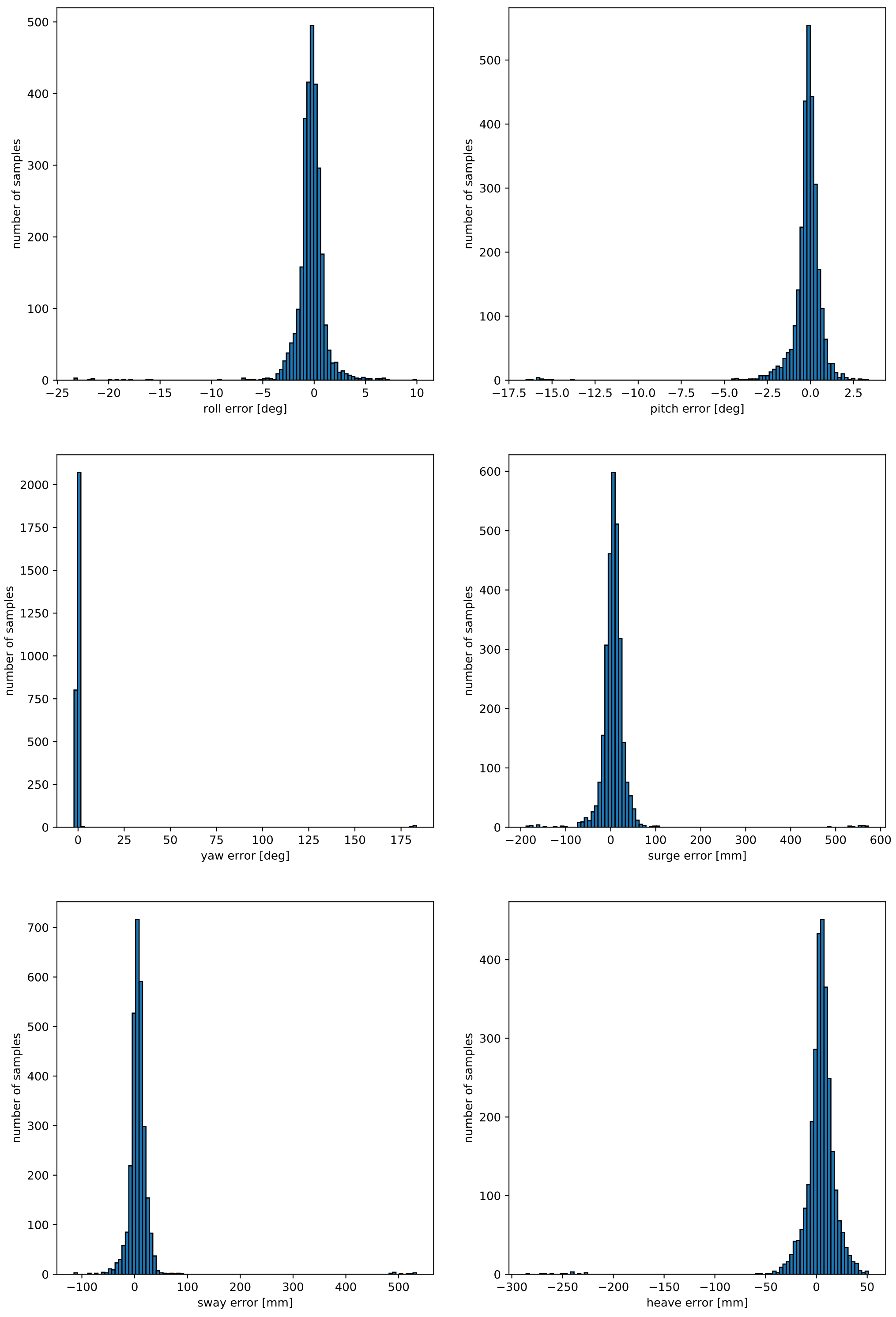


Figure 8.16: Histograms of errors in 6-DoF case D

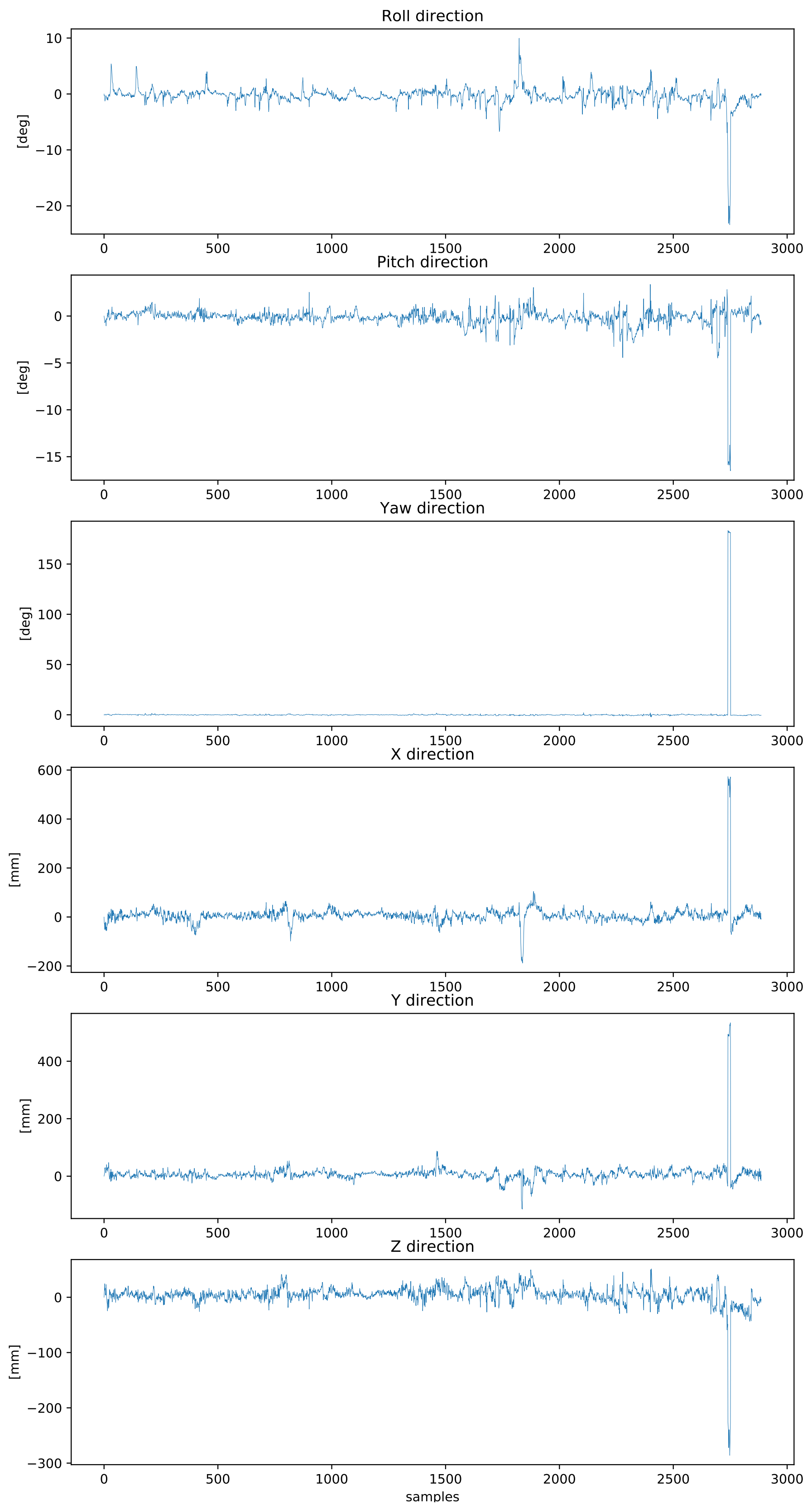


Figure 8.17: Case D: Pose errors in each DoF

Table 8.16 lists the amounts of the test data satisfying the 10-percent requirements. The amounts of test data satisfying these requirements are reduced from case C.

Measurement	**Amount**
Translation error less than 10 % of distance, $T_{\%}$	88 %
Rotation error less than 10 % rotation, $A_{\%}$	61 %

Table 8.16: Case D: Amount of test data satisfying the 10-percent requirements

Since the dataset in this case was sequentially split into train and test sets, there is a sequenced order in the test data. The true and predicted pose is therefore plotted for the samples in the test data, representing a time series. Figure 8.18 shows true and predicted orientation in roll, pitch and yaw directions. The blue and orange lines represent true and predicted values, respectively. The predicted orientations follow the true values well. The predicted orientations in pitch direction are slightly less satisfactory, with more frequent gaps between the true and predicted value than what is seen in the remaining directions. However, due to the smaller scale in this particular plot, more details are also revealed in the pitch plot compared to the roll and yaw plots. The network fails to predict orientation in the small interval seen to the right in the plots, according to what was seen in Figure 8.17. This mainly regards the yaw predictions, yielding errors of almost 180 degrees.

Similarly, Figure 8.19 shows true and predicted position in surge, sway and heave directions. Again, the blue line represents the true values, and the orange line represents the predicted values. These plots reveal that the predicted positions follow the true positions very well. Similar to what was seen for the orientation plots, large errors are observed in the small area to the right in the plots. The true position in both roll, sway and heave is suddenly smaller in this area, which the network fails to predict. At the same time, the network overshoots its predictions in the remaining DoFs, yielding large errors in all 6-DoF. Hence, it appears that the network understands that these images represent a larger relative pose, but fails to sort out in which degrees of freedom.

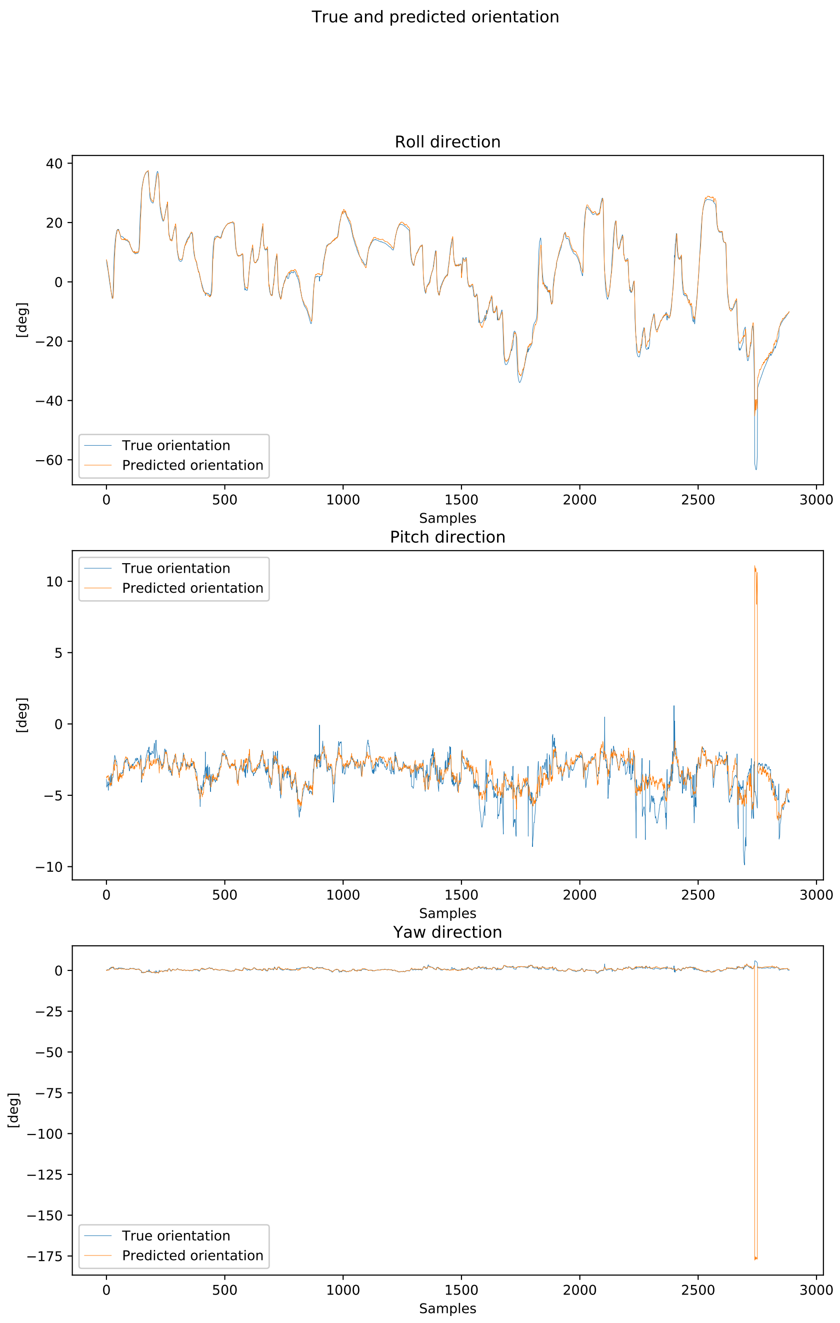


Figure 8.18: Case D: True and predicted orientation in roll, pitch and yaw

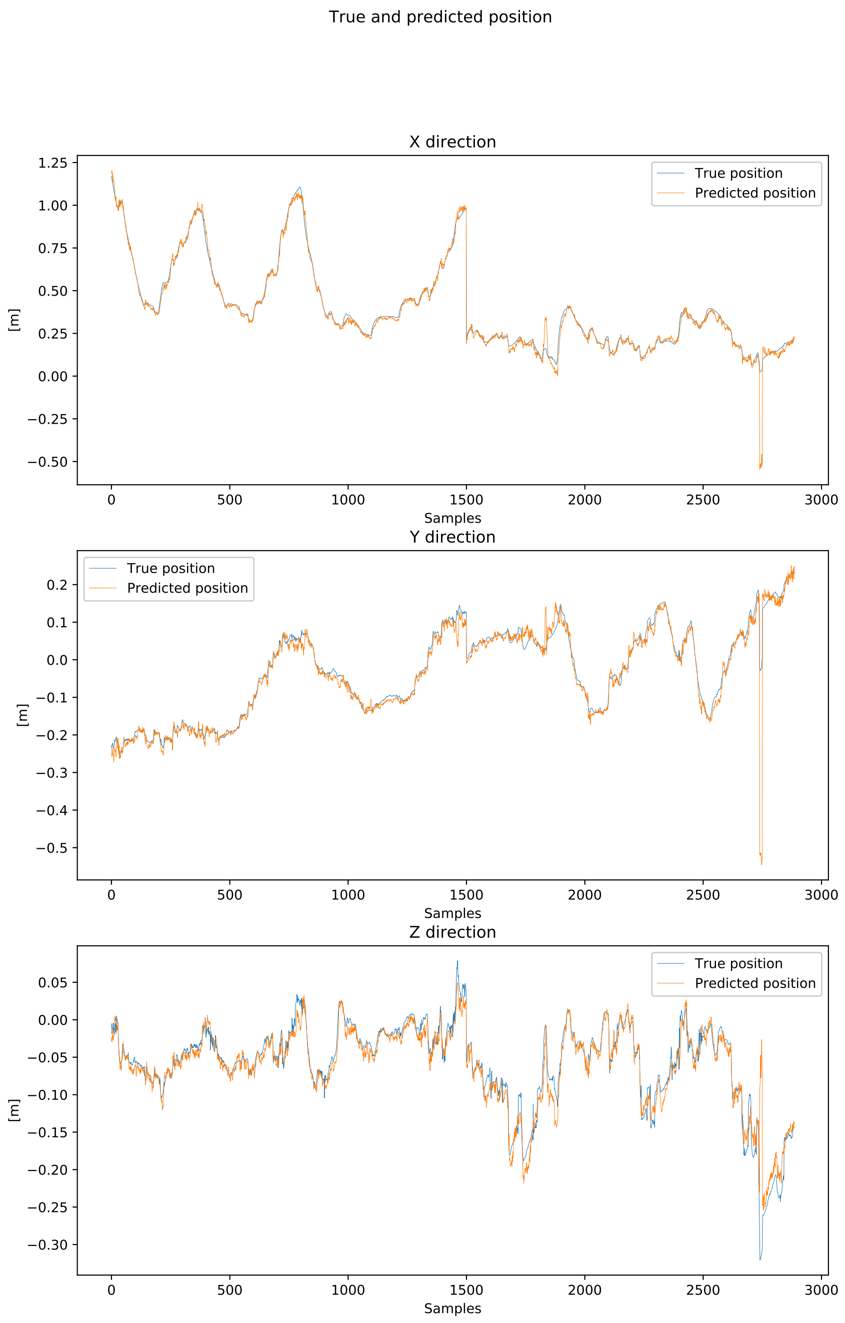


Figure 8.19: Case D: True and predicted position in surge, sway and heave

8.2.5 Additional Results

This section presents additional results on experiment cases that were conducted to investigate particular topics, as described in the process outline in Section 8.1. Only a statistical summary of these cases are presented here, and detailed results can be seen in Appendix D. The cases are grouped in tables organized by the topic their results highlight. In the following tables, the dataset IDs (referring to Table 6.1) in which the train and test sets were sampled from and other relevant properties of each case is listed. The median, mean and standard deviation of the Euclidean translation error and the relative angle error are included, to give an overall impression of the model performance in each particular case. In addition, a reference to the appendix of the detailed results for each case is included.

Filter by Distance

The distances between the camera and the target object in dataset 1 vary between 0 and 7 meters, which was exploited to investigate the influence of the distances on the performance of the network. The dataset was filtered by distance, and the network was trained and tested on data with relative distances in specific value ranges. For comparison, results from one case using images with relative distance less than 1 meter, and one case using the entire dataset (distances less than 7 meters) are listed in Table 8.17.

Case ID	Training dataset	Testing dataset	Distance	Position error [mm]			Orientation error [deg]			Details
				Med	Avg	Std. dev	Med	Avg	Std. dev	
1	1	1	Dist≤1m	38.95	47.45	65.96	1.112	1.793	4.709	D.1
2	1	1	Dist≤7m	89.98	106.97	84.63	1.154	1.633	3.493	D.2

Table 8.17: Summary statistics of results from cases with different relative distances between the camera and target object

Table 8.17 reveals that the translation errors are significantly smaller in the case where the distances are less than 1 meter (case 1), than the case with distances up to 7 meters (case 2). However, the 10-percent requirement for translation is approximately equally satisfied in the two cases, seen in Table D.4 and Table D.8. The translation errors are less than 10 % of the relative distance in 86 % and 82 % of the test data in the close-up and distanced case, respectively. The size of the translation errors of the pose estimates is thus proportional to the relative distances between the camera and the target object. The rotation errors remain approximately the same for both cases.

Light Conditions

In the second lab session, datasets with varying light conditions were produced. This was done to investigate how robust the pose estimation method is in challenging conditions. Recall the three different light conditions from Section 6.4:

1. *Full lights:* all lights in the MC lab is turned on
2. *Reduced light A:* the lights on the row furthest away from the target object is turned off
3. *Reduced light B:* the lights on the row closest to the target object is turned off

The results obtained with the datasets of varying light conditions are listed in Table 8.18.

Case ID	**Training dataset**	**Testing dataset**	**Light**	**Position error [mm]**			**Orientation error [deg]**			**Details**
				Med	Avg	Std. dev	Med	Avg	Std. dev	
3	2,3,4	2	Full lights	81.96	246.90	343.32	2.481	12.630	20.973	D.3
4	2,3,4	3	Reduced lights A	125.35	285.61	464.52	1.887	18.586	50.296	D.4
5	2,3,4	4	Reduced lights B	75.52	132.97	216.15	2.024	6.300	21.021	D.5

Table 8.18: Summary statistics of results from cases with different light conditions in the MC-lab

From Table 8.18 it is seen that the largest errors occur under the reduced lights A condition (case 4). The smallest errors occur under the reduced lights B, (case 5), which are even smaller than the case with all lights turned on (case 3).

Backgrounds and Target Object

Recall from Section 6.4 that the datasets produced in the MC-lab contain images of two different objects, the AruCo marker and the ring object, on two different backgrounds, the yellow box and the metal stick. Results from four cases combining these objects and backgrounds differently are listed in Table 8.19.

Case ID	**Training dataset**	**Testing dataset**	**Backgr./ object**	**Position error [mm]**			**Orientation error [deg]**			**Details**
				Med	Avg	Std. dev	Med	Avg	Std. dev	
2	1	1	Box/AruCo	89.98	106.97	84.63	1.154	1.633	3.493	D.2
B	2,4	2,4	Stick/AruCo	26.52	38.33	68.64	0.578	1.209	4.058	8.2.2
6	6	6	Box/ring	21.46	22.54	8.98	0.503	0.670	0.587	D.6
C	5	5	Stick/ring	19.16	23.88	33.86	0.381	0.721	3.017	8.2.3

Table 8.19: Summary statistics of results from cases with different background and different target object

For the AruCo marker cases, the errors are larger in the case with the box being the background (case 2) than the case with the stick (case B), seen in Table 8.19. Conversely, in the cases with the ring being the target object, the case with the box as background (case 6) yields lower errors than the case with the stick being the background (case C).

However, the errors are significantly lower in the cases with the ring object being the target object, compared with the AruCo marker. This was first observed in the difference of the results of case B and case C, and is also confirmed in Table 8.19, where the errors of the cases with the ring object (case 6 and C) are smaller than the cases with the AruCo marker (case 2 and B).

Random vs. Sequential Split of Datasets

Recall from Section 8.1 that the datasets were split into train and test sets in two different ways, either random or sequential. Results obtained with the two different approaches are listed in Table 8.20.

Case ID	**Training dataset**	**Testing dataset**	**Splitting**	**Position error [mm]**			**Orientation error [deg]**			**Details**
				Med	Avg	Std. dev	Med	Avg	Std. dev	
2	1	1	Random	89.98	106.97	84.63	1.154	1.633	3.493	D.2
7	1	1	Sequential	205.18	236.19	153.16	3.5127	5.513	5.498	D.7
B	2,4	2,4	Random	26.52	38.33	68.64	0.578	1.209	4.058	8.2.2
8	2,4	2,4	Sequential	77.85	181.39	269.41	2.376	8.825	18.668	D.8
6	6	6	Random	21.46	22.54	8.98	0.503	0.670	0.587	D.6
9	6	6	Sequential	50.88	63.15	47.66	1.316	1.757	1.418	D.9
C	5	5	Random	19.16	23.88	33.86	0.381	0.721	3.017	8.2.3
D	5	5	Sequential	27.95	38.60	81.15	0.929	2.029	11.162	8.2.4

Table 8.20: Summary statistics of results with both random and sequential split of dataset

The cases are listed pairwise in Table 8.20, where the same dataset is split into train and test sets both randomly and sequentially. All the cases with random split yield smaller errors than the cases with sequential split, which was also observed in the difference of the results of cases C and D.

Transfer Knowledge Between Domains

It was conducted train and test cases to assess how well the network generalizes its knowledge between different domains. In these cases, the network was tested on data from another dataset than it was trained on. The results obtained from this are listed in Table 8.21. Detailed results are omitted from Appendix D for the cases yielding extremely large errors, as further details provide no additional insight.

Case ID	**Training dataset**	**Testing dataset**	**Object**	**Position error [mm]**			**Orientation error [deg]**			**Details**
				Med	Avg	Std. dev	Med	Avg	Std. dev	
16	0	2	AruCo	1047.3	1051.5	230.32	17.239	19.921	13.119	—
10	1,2,4	1	AruCo	232.98	269.62	156.54	3.910	6.377	6.889	D.10
17	2,4	1	AruCo	1007.5	1075.0	531.59	37.23	38.43	15.59	—
7	1	1	AruCo	205.18	236.19	153.16	3.5127	5.513	5.498	D.7
11	1,2,4	2,4	AruCo	92.27	168.63	227.13	1.955	6.830	15.796	D.11
18	1	2,4	AruCo	1023.5	1019.3	415.25	20.22	23.22	17.36	—
8	2,4	2,4	AruCo	77.85	181.39	269.41	2.376	8.825	18.668	D.8
12	5,6	6	Ring	41.54	54.07	46.03	1.182	1.537	1.323	D.12
13	5	6	Ring	196.91	201.21	74.27	4.962	5.112	2.449	D.13
9	6	6	Ring	50.88	63.15	47.66	1.316	1.757	1.418	D.9
14	5,6	5	Ring	26.59	38.09	97.41	0.965	2.042	11.349	D.14
15	6	5	Ring	203.21	215.21	102.28	5.962	7.194	6.418	D.15
D	5	5	Ring	20.22	27.76	52.29	0.856	1.853	11.274	8.2.4

Table 8.21: Summary statistics of results from cases transferring knowledge between different domains

The network trained on the simulated data was tested on real-world data of the AruCo marker, corresponding to case 16 in Table 8.21. All measurements of both the position errors and the orientation errors are extremely large in this case.

Continuing, it was experimented with the different domains of the real-world data. Recall from Section 6.4 that for each object, being the AruCo marker and the ring object, two different datasets with different backgrounds were recorded. The network was trained on data with one background, and tested on data with the other. This was done both ways, with both objects. Besides, the network was trained on a concatenated dataset from both domains, and tested on each of the domains individually. The related results can be seen in Table 8.21. For comparison, the cases where the network is trained and tested on data from the same domain are also included in this table.

The relevant results from transferring knowledge between the two different background domains with the AruCo marker are seen from cases 10, 17, 7, 11, 18 and 8 listed in Table 8.21. For both backgrounds, the lowest errors are achieved when the network was trained only on the data from the same dataset as the test set (case 7 and case 8). When trained on a concatenated dataset representing both domains, (case 10 and case 11), the errors were slightly larger. In the cases where training and testing data represent different domains (case 17 and case 18), the errors are enormous, with position errors larger than 1 meter and orientation errors in the range of 20-40 cm.

The relevant results from transferring knowledge between the two different background domains with the ring object are seen from cases 12, 13, 9, 14, 15, and D listed in Table 8.21. Similar to the corresponding discussion on the AruCo marker cases, the smallest errors are obtained when the network is trained and tested on data from the same domain (case 9 and case D). Again, the errors obtained when the network is trained on a concatenated dataset representing both domains are slightly larger (case 12 and case 14). For the cases where the training and test data represent different domains (case 13 and case 15), the errors are significantly reduced from what was seen for the corresponding cases with the AruCo marker. Although the magnitudes of the errors are still large in these cases, being approximately 20 cm and 5-6 degrees in translation and orientation, respectively, the network still manages to predict the poses to some extent. It can be seen from the plots of the true and predicted poses for case 13 and 15, in Figure D.45, Figure D.46, Figure D.53 and Figure D.54, respectively, that the poses predicted by the network follow the contour of the true poses (to varying extents in the different DoF).

Chapter 9

Discussion

This chapter presents a discussion of the work of this thesis. Some overall remarks on the results are presented in Section 9.1, before the datasets and model will be discussed in conjunction with the results in Section 9.2 and Section 9.3, respectively. Finally, the feasibility of the suggested model is discussed in Section 9.4.

9.1 Overall Remarks

Firstly, the errors in all the four cases A, B, C and D presented in Section 8.2 are considered satisfactorily small. The median of the Euclidean translation error varies between 19 and 28 mm, and the median of the relative angle error varies between 0.4 and 0.9 degrees in these cases.

The results further revealed that the model performs better on the simulation data (case A) than the real-world data (case B, C, and D). This was expected, as the underwater environment features more challenging image conditions than a noise-free simulation environment. This difference primarily regards the occurrence of large outliers in the errors. Such outliers occur to varying extents in the real-world experiments, while they do not occur at all for the simulation case.

Moreover, it is observed from the results that the network performs more satisfactory in the cases with the ring object being the target object, compared with the ArUco marker. This may relate to the symmetry of the ring object, and it is possible that the network manages to learn the radius of the ring and from this reason about the relative pose. Regardless, this result suggests that the PoseNet model does not favor the presence of artificial markers in the images. Since this work was motivated by the need for an underwater pose estimation method without the need for artificial markers, this is considered a significant result.

9.2 Datasets

For all data-driven approaches, the datasets utilized introduce uncertainty to the results. Some observations in the results address challenges related to the datasets, which are discussed in the following. Recall that plots of the pose labels of the datasets are seen in Appendix C, and sample images are seen in Figure 6.6 and Figure 5.2.

9.2.1 Incorrect Pose Labels

The results presented in Section 8.2 revealed that the pose errors exhibit significant outliers to varying extents in all the real-world experiments (cases B, C and D). One reason for the occurrence of these large outliers may be the process at which labeling of the datasets took place. Recall from Section 6.3 that the process of labeling the datasets was automated and based on the raw Qualisys measurements mapped by a time-stamp to the camera of the ROV. The Qualisys system sometimes lost tracking of the ROV, and when re-tracking occurred, the estimated axes applied to the vehicle often exhibited wrong rotations for a couple of samples. This resulted in incorrect, outlying values for a few of the pose labels, seen in the pose plots of the datasets in Appendix C. It is uncertain how these incorrect pose labels affect the performance of the network. When samples with these incorrect, outlying pose labels occur in the test set, it necessarily affects the results as the network will not predict the corresponding incorrect values. It is however not obvious how the outliers affect the training process. On the one hand, they only occur in a few samples out of thousands in the training sets, and should therefore not have a too big impact. However, the loss value calculated by the network when processing these samples will indeed be large, and from the update rule in gradient descent optimization (see Section 3.2) this can lead to relatively large adjustments on the network weights. Due to the complex network architecture featuring 23 layers, the network has a large capacity in terms of the amount of information from the training set that can be stored in the network weights. It is therefore considered possible that the network "remembers" the particular training samples with outlying, incorrect pose labels, and in a way hard-codes these pose values into the weights during the training process. The latter is considered of importance, and the hypothesis is thus that the erroneous labeling influence the training process negatively.

The outlying, incorrect pose labels appear in all the real-world datasets except from dataset 6 (see Table 6.1), whose pose plots are seen in Appendix C.7. The results obtained when training and testing the network on dataset 6 are seen in Appendix D.6, where no outliers of significant magnitude occur among the errors. This demonstrates the negative impact of the incorrect pose labels, and the occurrence of outlying errors could arguably have been reduced by filtering them out of the datasets.

Further, the issue of incorrect labeling does not apply for the simulated dataset, seen in Appendix C.1. This, in addition to the different difficulty levels of image conditions, explain the performance gap between the simulation experiment (case A) and the real-world experiments (case B, C and D).

9.2.2 Value Range of Pose Labels

When inspecting the values of the 6-DoF pose labels of the datasets in Appendix C, it is clear that the pose values are not evenly represented in the datasets. The range of the pose values for the data points varies between each DoF, and between the different datasets. In addition, some values are represented by a greater amount of data points than others. These inconsistencies in the pose labels relate to some of the observations in the results, elaborated in the following.

Errors of Pose Estimates

There are two particular trends in the occurrence of the 6-DoF pose errors that relate to the values of the pose labels in the datasets.

Firstly, in each of the four cases presented in Section 8.2, the magnitudes of the pose estimation errors vary between the different degrees of freedom. This is seen in conjunction with the values of the pose labels, and it appears that in each case, the largest errors of the pose estimates occur in the DoFs in which the pose values of the relevant dataset varies within the largest size range. Similar, the smallest errors tend to occur in the DoFs in which the pose values feature small ranges. Some specific observations illustrating this relation are listed below.

- **Case A**: Recall from Section 8.2.1 that the smallest attitude errors is seen in roll direction. This corresponds to the direction with the smallest value range of the pose labels, seen in Appendix C.1, where the rotation in roll is more or less zero for all the data points.
- **Case B**: Recall from Section 8.2.2 that the largest pose errors occur in yaw and sway directions. This corresponds to the directions with the largest value range (and

most incorrect pose labels according to the discussion in Section 9.2.1) of the pose labels, seen in Appendix C.3 and Appendix C.5. Similar, the smallest orientation errors are observed in pitch direction, which is the direction with the smallest value range and no outliers in the pose labels.

- **Case C**: Recall from Section 8.2.3 that the largest pose errors occur in roll direction. Again, this coincides with the plots of the pose labels of the dataset used in this case seen in Appendix C.6.

A second relation between the errors and the pose labels is seen in the results for case D, particularly in Figure 8.18 and Figure 8.19. The network fails to predict the pose for a selection of samples seen to the right in these plots. It is observed that the true pose values for roll, sway and heave suddenly are significantly smaller for these particular samples compared with the rest of the dataset. Thus, the network fails to predict pose for samples with extreme valued poses. Similar trends are also seen in cases presented in appendices D.11, D.14 and D.12.

These two relations between the errors and the pose labels are considered reasonable. Recall from Section 2.3.3, that a great challenge with regression networks is namely the datasets. A neural network learns from the data it is exposed to during training, and traditional classification networks normally need several training examples for every possible category. For regression problems, the output domain is infinite and continuous, and it is therefore impossible to have training examples for every possible outcome. It is however desired to cover most of the possible output domain during training, which relates to two properties of the pose labels in the dataset:

1. **Size of value range**: The value range for the pose labels in a particular DoF and/or particular datasets defines the size of the possible output pose domain. A larger output domain requires larger training sets to cover a sufficient amount of the possible pose values. Larger output domains thus feature more complex regression problems. It is therefore reasonable that the smallest/largest errors occur in the directions in which the true pose values vary within the smallest/largest value ranges in the datasets.

2. **Distribution within value range**: To cover a sufficient amount of the pose domain, each value should be represented by approximately the same number of data points in the training set. Ideally, the images should have been sampled from a uniform distribution in all 6-DoF. Since this is not the case for the datasets collected in this work, the network is likely not exposed to the extreme pose values for a sufficient number of times during training. It is thus reasonable that the network struggles to estimate extreme pose values, as seen for case D.

It can be seen from the plots in Appendix A.1 that the translations between the camera and the AruCo board lie in the range between -0.15 m and 0.15 m in sway and heave directions, and the roll rotations are more or less zero for all samples. This represents smaller value intervals than what is seen for the real-world datasets. The pose domain for the simulated dataset is thus significantly smaller compared to the real-world data, which reduces the complexity of the pose regression problem in case A. This further explains the performance gap between the simulation experiment (case A) and the real-world experiments (case B, C and D).

Transfer Knowledge Between Domains

The results presented in Table 8.21 revealed that transferring of knowledge between different domains was difficult. This particularly yielded the case when the network was trained on simulation and tested on real-world data. The difference between the range of pose values of the simulated and the real-world data is however significant, seen in appendices C.1 and C.3. Thus the network has not seen poses similar to the correct test poses during training, which explains why the network fails to apply its knowledge from the simulation data to the real-world data.

The results further revealed that transferring of knowledge between the two domains with the ring object was more successful than the corresponding experiments with the AruCo marker. Similar to previous discussion, it can be observed a large difference between the range of the pose values between the two datasets representing the AruCo marker in different domains, seen in appendices C.2 and C.3. Conversely, the pose values of the two different datasets with the ring object, seen in appendices C.6 and C.7, varies within a more similar value range than what was seen for the AruCo marker. This is likely to be part of the reason why the network transfers knowledge more successfully between the domains with the ring than the AruCo marker.

Random vs. Sequential Split

Whether the datasets were split into train and test sets randomly or sequentially turned out to make a great impact on the magnitude of the errors at test time. In particular, the cases where the train and test sets were obtained randomly yielded better results than the cases where the dataset was sequentially split, and the order in which the data points were recorded were kept in the train and test sets. This is observed in Table 8.20, and from the performance difference between case C and case D.

It is however reasonable that the errors are smaller when the network is trained and tested on datasets that are randomly obtained. Recall that the datasets collected in the

MC-lab, described in Chapter 6, are sampled at a relatively high frequency of 15 Hz. Consequently, images taken in a row have relatively similar values for the pose label, and image contents. When the train and test sets are generated completely randomly from such a dataset, there are high chances that there exist data points of similarity in the two sets. This way, the network is exposed to data points during training that are relatively similar to data points in the test set, naturally making it easier to predict the images in the test set at test time. This does not apply in the cases of sequential split, as the X first images of the dataset make up the training data, and the Y following images make up the test data. This difference relates to the uneven representation of pose values in the datasets.

It is still considered valid to split the datasets randomly into train and test sets, as one can argue that each data point is unique regardless of the sequence it is part of. Regardless of the fact that random split yielded the lowest errors, it is interesting to assess the results obtained with sequential split, since true and predicted pose can be illustrated for a time series. This is seen in Figure 8.18 and Figure 8.19 for case D, and can be seen for several results in Appendix D.

9.2.3 Image Contents

This section discusses properties and flaws of the images in the datasets.

Backgrounds

In the simulated dataset, described in Chapter 5, the backgrounds behind the ArUco marker were randomly generated, being unique in each image. This leaves the ArUco marker the only constant feature in the images across all the samples in the datasets. The fact that the network successfully learns how to regress the pose from these images, reveals that it learns to identify and extract information from only the ArUco marker, and disregard the background.

Conversely, in each of the real-world datasets, the backgrounds behind the target object are constant for all the images. The backgrounds are also relatively feature-rich, seen in Figure 6.6, and it is therefore possible that the CNN learns to regress pose based on additional features than the objects in the images. This further makes it difficult to apply knowledge from one domain on another, complementing the previous discussion regarding transferring of knowledge between domains.

Light Contamination

The results governing the investigation of light influence, presented in Table 8.18, revealed that the largest errors appear under the reduced light A condition, while the reduced light B condition featured even lower errors than what was obtained with all the lights on. It is remarkable that the network struggles in only one of the reduced light cases. When inspecting sample images from the different datasets, such as the ones seen in Figure 6.6, one can see that the images taken under condition A (corresponding to dataset 3) contain red lights from the cameras in the Qualisys system. This light contamination is not present in the images of condition B, nor in the datasets with full lights. It is not clear why the light contamination only appears in the dataset with reduced light A, but it does explain why the network performance is less satisfactory in this case. The images in the dataset of condition B (dataset 4) however, appear even more clear than the images taken with full lights. Although the different datasets used to investigate the influence of light conditions represents different image quality, they do not necessarily represent the influence of light conditions very well due to the noise contamination from the deep blue LEDs in the laboratory. This investigation of the model's robustness to difficult light conditions is therefore not considered sufficient. However, it is shown that the model performance is indeed related to the image quality, which was expected.

For future work, varying light conditions should be further investigated. It would be more interesting to turn all the lights in the MC-lab off, and equip the ROV with light sources as this would be a more realistic representation of the light conditions during a subsea operation.

Incorrect Colors

Another issue with the dataset produced in this work is the loss of some colors in the images. As seen in the example images in Figure 6.6, the orange ring object and the yellow template box used in the experiments both appear blue in the images. This is likely due to some issues when converting the laboratory data from bag files. It is uncertain how this flaw affects the performance of the network. Since the ring object is blue in the images, it is not represented in a separate channel (RGB-channels) from the background in the image matrix. The channels of the input images are however mixed throughout the network architecture, and do not necessarily affect the performance that much. One can also argue that the fact that the network has to identify the object in the same channel as the background forces the network to investigate the shades of the images more, making it more robust for pose estimation.

9.3 Model

This section discusses the model implemented in this work, described in Chapter 4.

9.3.1 Architecture

The architecture of the implemented model is described in Section 4.2. While the PoseNet architecture is the state-of-the-art CNN for 6-DoF pose regression, it was developed for a different problem than the one addressed in this thesis. PoseNet was designed for a more coarse pose estimation purpose, namely to decide the relative pose of a camera relative to a scenery of spatial extents in ranges up to 500 x 100 meters. The associated errors obtained for this purpose lie in the range of 2 meters and 8 degrees. It is therefore not comparable to this work, operating over small spatial extents yielding errors in the range of 20 mm and 0.5 degrees. The PoseNet model has thus not been shown to be successful for high-precision localization systems for robotic applications. The architecture of the implemented model is discussed in conjunction with the results in the following.

The original PoseNet model was chosen in this thesis. For most of the extended models presented in Section 2.3.4, it is not clear whether the models actually deliver better performance than the original PoseNet model. However, The Bayesian PoseNet model [29] is developed by the same authors as the PoseNet, and is tested with the same evaluation metrics and the same datasets, yielding a valid comparison to the PoseNet performance. This model is shown to outperform the PoseNet in terms of accuracy, and could therefore have been considered implemented in addition to, or instead of, the original PoseNet in this work.

It was observed from the results that the pose errors exhibit some significant outliers, to varying extents, in the real-world experiments (case B, C, and D). According to previous discussion, these outliers typically occur for samples with extreme valued pose labels. Further, these outliers tend to occur in multiple DoFs at the same time, seen in Figure 8.9, Figure 8.13 and Figure 8.17. This illustrates how it might not be beneficial having a model regressing all 6-DoF simultaneously, such as the PoseNet. The outlying errors are highly coupled, and could possibly have been reduced by separating the regressors, with an architecture similar to BranchNet [35] introduced in Section 2.3.4.

As previously discussed regarding the datasets, the regression model is problematic in terms of the pose label domain problem. When solving a problem with a continuous output domain with supervised learning, there exists an upper limit in terms of possible accuracy that can be achieved. This is related to the fact that a continuous pose domain cannot be perfectly covered by a finite training data set. The regression model itself is therefore challenging, since there exists no perfect dataset to solve the problem.

9.3.2 Hyperparameters

Common for all neural networks is that the hyperparameter configuration greatly impact the performance. Identification of the optimal hyperparameters is a complex problem, requiring expensive grid search and/or advanced strategies, making up a whole research field in itself. The values for the hyperparameters were therefore mainly based on previous work and some trial and error, described in Section 4.4. Hence, a more optimal configuration of the hyperparameters of the model for the purpose of this work is likely to exist.

The scaling factor, β, in the loss function seen in Equation 4.2 is in particular a crucial hyperparameter in this model. Recall from Section 4.1 that this factor weighs the relative importance between the translation error and the orientation error for the pose estimates delivered by the network. The value of this scaling factor was identified by a coarse grid search (see Section 4.4) for one of the datasets. Tuning of this factor is expensive in terms of time, as it requires training of the network in several iterations. This grid search was therefore only conducted once, although it could have been beneficial to tune it individually for each dataset. Another thing to notice regarding this tuning, was that no obvious optimal value for β was suggested by the plots of the grid search, seen in Figure 4.2. For comparison, the grid search carried out during the development of the original PoseNet model, seen in Figure 2.2, suggest a more obvious optimal value for β. It is therefore not certain that the chosen value in this work, of $\beta = 75$, is optimal. The PoseNet model is criticized namely for this challenging scaling factor. Several alternatives presented in Section 2.3.4 are therefore suggested, and could also have been explored in this work.

The relation between the size of the position errors and the rotation errors is observed from the results. This relation varies slightly from case to case, but it can be seen from the results that a translation error around 20 mm corresponds to a relative angle error around 0.5 degrees, approximately. Rotation errors are considered more critical since a wrong estimation of the rotation eventually leads to an increased translation error. The relation between the two errors observed in the results therefore seems reasonable. If however one of the two errors is considered of higher importance, the scaling factor β in the loss function of the network, seen in Equation 4.2, can be adjusted accordingly.

In addition to the hyperparameters listed in Table 4.1, different approaches for image preprocessing and weight initialization could have been explored more. The weights could for instance have been initialized with the GoogLeNet weights trained on a dataset more relevant for this case than the Places [55] set.

9.4 Feasibility

This section discusses to what extent the suggested model is applicable to serve as a complementary localization system for an underwater vehicle performing manipulation work. As stated in the introduction, a real-time algorithm for frequent 6-DoF localization with high accuracy is needed to increase the level of autonomy in ROV operations.

The discussion of feasibility is based on case C, where the model estimates pose relative to the ring object representing a subsea valve. According to previous discussion, the results of this case are better than the cases covering the AruCo marker. It is thus shown that the model suggested in this work does not benefit from artificial markers. This strengthens the feasibility of the method, as no additional infrastructure for insertion of artificial markers on subsea structures is needed.

The suggested algorithm is not tested real-time in this work, but according to the authors of the PoseNet paper [9], each pose can be calculated in only 5 ms with this architecture, which is considered sufficient. The network weights are stored in only 50 MB, which is feasible for implementation on an underwater vehicle.

In terms of accuracy, it is difficult to determine the necessary level of accuracy required for different subsea operations. The medians of the total errors in orientation and translation obtained with the method are 0.4 degrees and 19 mm, respectively. The means and standard deviations of the translation and orientation errors obtained in this case are 24 ± 34 mm and 0.7 ± 3.0 degrees, respectively. For comparison, the artificial marker-based method presented in [7] featured corresponding errors of 118 ± 126 mm and 4.2 ± 4.6 degrees, respectively, which is outperformed by the method suggested in this work. As seen in the results, there are a few large errors in the pose predictions delivered by the method, decreasing its feasibility. This can possibly be solved by having multiple networks regressing pose in parallel, and based on the multiple output poses determine the final output pose. Alternatively, the Bayesian PoseNet model [29] discussed in Section 9.3.1 can be considered to reduce the error outliers, as it quantifies the uncertainty of each pose label predicted by the network.

The achieved accuracy obtained in this work was discussed with Dr. Nuno Gracias, a postdoctoral researcher of the Computer Vision and Robotics Research Group of the University of Girona. His team has conducted several studies regarding localization of underwater vehicles, such as [3]. He commented that the accuracy of this work seemed reasonable, but emphasized the difficulty of quantifying the degree of accuracy required for manipulation work without testing in a closed-loop. The feasibility of the accuracy obtained with this method should therefore be assessed in a closed-loop implementation with a controller in further work.

It can be considered to use this model in cooperation with other computer vision algorithms. For instance, the model suggested in this work can provide localization assisting the ROV to navigate to a certain area where one or more artificial markers are visible, and from there on continue with traditional CV algorithms for pose estimation with artificial markers. It can further be considered to use the measurements from this model as input to a sensor fusion system, allowing for measurements from different sensors to correct and improve each other. This can be achieved with an extended Kalman filter (EKF), which can incorporate the different measurement inputs based on their associated uncertainties [7].

A drawback with the suggested CNN model is the need for labeled training data, which is not available from the environment of subsea installments. Transferring knowledge from simulation to real-world data is not shown successful from the results. Transferring knowledge between different underwater domains is however shown partially satisfactory, demonstrating the network's potential to transfer knowledge between domains in general. In order to be able to apply a network trained exclusively on simulated data in real-world applications, more work should have been invested in the simulation, to make it more similar to real-world data. Based on the results discussed above, it is considered crucial that the values of the pose labels in the training set lie within the same range as the relative poses one expect during the actual subsea operation (corresponding to the test sets). It would necessarily be required simulated images of the subsea valve as well. It can also be considered an option to generate training data for a subsea operation in the MC-lab, by 3D-printing the subsea component(s) of interests, and produce datasets with labeled images according to the procedure of Chapter 6. Regardless, further improvements for transferring the network knowledge between different domains are necessary for the model to be feasible in industrial applications.

Chapter 10

Conclusion

This thesis has investigated the application of a CNN method to estimate 6-DoF pose in an underwater environment, as an alternative to existing artificial marker-based methods. This was intended to serve as a complementary, high-precision localization system for an underwater vehicle performing subsea manipulation work. The CNN architecture PoseNet was chosen for this purpose.

The thesis presented related work on underwater localization, including state-of-the-art methods for camera-based underwater pose estimation with artificial markers. Recent advances within the field of image processing with CNNs have resulted in promising methods for pose regression, and related work on this topic was reviewed and presented in this report. The CNN architecture PoseNet was implemented and assessed. Real-world datasets were produced in the MC-laboratory at NTNU, containing underwater images of objects labeled with the 6-DoF pose. The network was then trained and tested in several iterations, beginning with images of an AruCo marker before moving on to images of a mock-up subsea valve.

The results showed that the implemented model regresses 6-DoF underwater pose successfully, based on imagery input of the mock-up valve. The model delivers accuracy of 19 mm and 0.4 degrees for position and rotation, respectively. The results revealed that the implemented model, in fact, performs better on images of the valve model, than images of the AruCo marker. The undesired need for artificial markers is thus removed with this method.

Suggestions for further work are presented in the next chapter.

Chapter 11

Further Work

This thesis has investigated the application of a CNN to estimate relative 6-DoF pose between an underwater vehicle and a fixed object, in order to remove the need for artificial markers. Suggestions for further work are presented in the following.

Firstly, the suggested solution should be applied in a closed-loop for station keeping of an ROV performing manipulation work, in order to assess the feasibility of the accuracy achieved with the method.

It was seen from the results that the errors of the pose estimates exhibit some significant outliers. Further work should investigate methods for reducing these, e.g. by exploring the Bayesian PoseNet model [29] according to the discussion in Section 9.4.

Another possible approach for reducing the outliers of the errors is to incorporate temporal abilities to the model. The model implemented in this thesis estimates the 6-DoF relative pose based on one single input image, without taking previous estimates into account. That is, the algorithm estimates current pose independent of time, without exploiting knowledge about previous poses. Since the images in the collected datasets are sampled at a frequency of 15 Hz, the sequenced order of the datasets represents a temporal encoding. This could be exploited by adding temporal abilities to the model. Recurrent neural networks, introduced in Section 3.4, was investigated for this purpose. Such networks introduce temporal abilities to neural networks, and it was considered to add recurrent blocks to the architecture of the implemented model described in Section 4.2. However, this approach was discussed with Alex Kendall, one of the authors of PoseNet [9], suggesting that such temporal models often acts simply as a low pass filter in practice. Further investigation of how temporal abilities can be included in the suggested model, either with filters or recurrent neural networks, is recommended for further work.

Moreover, further investigation of the network's possibility to transfer knowledge between different domains should be performed. Training data is not available from the environment of subsea installments, and improved methods for transferring knowledge from simulation to real-world environments are necessary for the model to be feasible in industrial applications. Improvements of the simulation environment are suggested, in order to reduce the gap between the simulated and real-world domains. It can also be considered to explore generative adversarial networks (GAN) [24] for improved transferring of knowledge between domains.

Additional topics suggested for further works are listed below.

- Further assess the method's robustness to challenging lighting conditions.
- Implement the measurements from the suggested model as input to a sensor fusion system, allowing for measurements from different sensors on the ROV to correct and improve each other.
- Approaches for preprocessing of the images can be explored. It can be considered to implement an Autoencoder [24] in order to remove noise from the input images.
- CNN architectures such as the ones described in Section 2.3.4 can be explored as alternatives to the PoseNet model.

Bibliography

[1] I. Schjølberg, T. B. Gjersvik, A. A. Transeth, and I. B. Utne, "Next generation subsea inspection, maintenance and repair operations," *IFAC PapersOnLine*, vol. 49, no. 23, pp. 434–439, 2016.

[2] E. I. Grøtli, J. Tjønnås, J. Azpiazu, A. A. Transeth, and M. Ludvigsen, "Towards more autonomous rov operations: Scalable and modular localization with experiment data," *IFAC PapersOnLine*, vol. 49, no. 23, pp. 173–180, 2016.

[3] N. Palomeras, S. Nagappa, D. Ribas, N. Gracias, and M. Carreras, "Vision-based localization and mapping system for auv intervention," in *2013 MTS/IEEE OCEANS - Bergen*, pp. 1–7, June 2013.

[4] E. G. News, "Oceaneering awarded equinor contract for e-rov services." `http://www.energyglobalnews.com/oceaneering-awarded-equinor-contract-for-e-rov-services/`. Accessed: 2019-05-26.

[5] E. H. Henriksen, I. Schjølberg, and T. Gjersvik, "Vision based localization for subsea intervention," 2017.

[6] E. H. Henriksen, "System design for the next generation subsea imr vehicles," 2018.

[7] A. Gomez Chavez, C. Mueller, T. Doernbach, and A. Birk, "Underwater navigation using visual markers in the context of intervention missions," *International Journal of Advanced Robotic Systems*, vol. 16, 03 2019.

[8] T. Palmer, D. Ribas, P. Ridao, and A. Mallios, "Vision based localization system for auv docking on subsea intervention panels," in *OCEANS 2009-EUROPE*, pp. 1–10, IEEE, 2009.

[9] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," 2015.

[10] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes," 2017.

[11] T.-T. Do, M. Cai, T. Pham, and I. Reid, "Deep-6dpose: Recovering 6d object pose from a single rgb image," 2018.

[12] M. H. Leonhardsen, "Deep learning for underwater pose estimation: Preliminary studies." Project thesis, fall semester 2018.

[13] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," pp. 85–94, IEEE Publishing, 1999.

[14] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3400–3407, May 2011.

[15] S. Garrido-Jurado, R. Muñoz-Salinas, F. Madrid-Cuevas, and M. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280 – 2292, 2014.

[16] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," pp. 2564–2571, IEEE Publishing, 2011.

[18] V. Mondéjar-Guerra, S. Garrido-Jurado, R. Muñoz-Salinas, M. J. Marín-Jiménez, and R. Medina-Carnicer, "Robust identification of fiducial markers in challenging conditions," *Expert Systems with Applications*, vol. 93, pp. 336 – 345, 2018.

[19] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes," vol. 7724, pp. 548–562, 2013.

[20] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, pp. 858–865, IEEE, 2011.

[21] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, "Scene coordinate regression forests for camera relocalization in rgb-d images," in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2930–2937, IEEE, 2013.

[22] C. Choi and H. I. Christensen, "Rgb-d object pose estimation in unstructured environments," *Robotics and Autonomous Systems*, vol. 75, no. PB, pp. 595–613, 2016.

[23] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6d object pose estimation using 3d object coordinates," vol. 8690, pp. 536–551, Springer Verlag, 2014.

[24] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2017.

[25] H. Su, C. R. Qi, Y. Li, and L. Guibas, "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views," 2015.

[26] S. Tulsiani and J. Malik, "Viewpoints and keypoints," 2014.

[27] A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," 2017.

[28] F. Walch, C. Hazirbas, L. Leal-Taixé, T. Sattler, S. Hilsenbeck, and D. Cremers, "Image-based localization using lstms for structured feature correlation," 2016.

[29] A. Kendall and R. Cipolla, "Modelling uncertainty in deep learning for camera relocalization," *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

[30] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6d object pose prediction," 2017.

[31] M. Rad and V. Lepetit, "Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth," 2017.

[32] C. Szegedy, P. Wei Liu, S. Yangqing Jia, D. Sermanet, D. Reed, V. Anguelov, A. Erhan, A. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," vol. 07-12-, pp. 1–9, IEEE, 2015.

[33] X. Li, J. Ylioinas, J. Verbeek, and J. Kannala, "Scene coordinate regression with angle-based reprojection loss for camera relocalization," 2018.

[34] Q. Fang and T. Hu, "Euler angles based loss function for camera relocalization with deep learning," *CoRR*, vol. abs/1802.08851, 2018.

[35] J. Wu, L. Ma, and X. Hu, "Delving deeper into convolutional neural networks for camera relocalization," pp. 5644–5651, Institute of Electrical and Electronics Engineers Inc., 2017.

[36] J. P. Adam Gibson, "Getting started with deep learning," 2013.

[37] S. Rao, "Lecture notes, cs188: Introduction to artificial intelligence, UC Berkeley, fall 2017." https://drive.google.com/file/d/15EHCsuNc4vILsWg6HK8ML0F10W9E6Qrl/view. Accessed: 2018-11-29.

[38] A. Krizhevsky, I. Sutskever, and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, pp. 84–90, 2017.

[39] M. Agarwal, "Back propagation in convolutional neural networks - intuition and code." https://becominghuman.ai/back-propagation-in-convolutional-neural-networks-intuition-and-code-714ef1c38199. Accessed: 2018-11-14.

[40] S. K. Raghuwanshi and R. K. Pateriya, "Accelerated singular value decomposition (asvd) using momentum based gradient descent optimization," *Journal of King Saud University - Computer and Information Sciences*, 2018.

[41] G. Hinton, "Lecture notes: Neural networks for machine learning, coursera." https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed: 2018-11-29.

[42] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[43] M. D. Zeiler, "Adadelta: An adaptive learning rate method," 2012.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[45] S. Ruder, "An overview of gradient descent optimization algorithms," *CoRR*, vol. abs/1609.04747, 2016.

[46] A. Ng, "Online course, deep learning - convolutional neural networks, coursera." https://www.coursera.org/learn/convolutional-neural-networks?action=enroll&authMode=login&specialization=deep-learning. Accessed: 2018-09-10.

[47] W. W. A. Y. Soroush Nasiriany, Garrett Thomas, "Lecture notes, cs189: Introduction to machine learning, uc berkeley." http://snasiriany.me/files/ml-book.pdf. Accessed: 2018-11-29.

[48] A. Karpathy, "Lecture notes, cs231n: Convolutional neural networks for visual recognition, stanford." http://cs231n.github.io/convolutional-networks/. Accessed: 2018-11-05.

[49] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[50] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[51] A. Ng, "Lecture notes, deep learning - convolutional neural networks, coursera." `https://nhannguyen95.github.io/coursera-deep-learning-course-4-week-1/#edge-detection-example`. Accessed: 2018-11-29.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[53] J. J. Fei-Fei Li and S. Yeung, "Lecture notes, cs231n: Convolutional neural networks for visual recognition, stanford." `http://cs231n.stanford.edu/slides/2019/cs231n_2019_lecture10.pdf`. Accessed: 2019-05-07.

[54] D. Masters and C. Luschi, "Revisiting small batch training for deep neural networks," 2018.

[55] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," vol. 1, pp. 487–495, Neural information processing systems foundation, 2014.

[56] TensorFlow, "Tensorflow documentation." `https://www.tensorflow.org/`. Accessed: 10-09-2018.

[57] K. Sommer, "Tensorflow implementation of posenet." `https://github.com/kentsommer/tensorflow-posenet`. Accessed: 15-02-2019.

[58] The HDF Group, "Hierarchical data format, version 5." https://www.hdfgroup.org/HDF5/, 1997-NNNN. Accessed: 2019-02-05.

[59] Blender, "Blender documentation." `https://www.blender.org/`. Accessed: 10-09-2018.

[60] G. AI, "Open images challenge 2018." `https://storage.googleapis.com/openimages/web/index.html`. Accessed: 2019-04-25.

[61] B. Robotics, "Bluerov2." `http://docs.bluerobotics.com/brov2/`. Accessed: 2019-04-06.

[62] B. Robotics, "Raspberry pi camera module." https://www.bluerobotics.com/store/sensors-sonars-cameras/cameras/cam-rpi-wide-r1/. Accessed: 2019-05-16.

[63] Qualisys, "Qualisys." https://www.qualisys.com/. Accessed: 2019-04-08.

[64] ROS, "Robotic operating system (ros)." http://www.ros.org/. Accessed: 2019-02-05.

[65] T. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control.* 05 2011.

[66] J. P. Morais, S. Georgiev, and W. Sprößig, *An Introduction to Quaternions*, pp. 1–34. Basel: Springer Basel, 2014.

Appendix A

Abstract Paper

Attached below is the abstract submitted for the OCEANS 2019 Seattle conference.

Underwater 6-DoF Pose Estimation with Deep Learning

1st Mari Hovem Leonhardsen
Dept. of Marine Technology
NTNU
Trondheim, Norway
marihl@stud.ntnu.no

2nd Mikkel Cornelius Nielsen
Dept. of Marine Technology
NTNU
Trondheim, Norway
mikkel.cornelius.nielsen@ntnu.no

3rd Ingrid Schjølberg
Dept. of Marine Technology
NTNU
Trondheim, Norway
ingrid.schjolberg@ntnu.no

***Abstract*—Autonomy in underwater intervention operations requires localization systems of high accuracy. State-of-the-art methods rely on computer vision to provide the necessary localization accuracy. However, traditional computer vision solutions require hand-crafted feature extraction, which often necessitates reliance on artificial markers, which is undesired. Recent advances within deep learning, in particular, convolutional neural networks (CNNs), have resulted in promising methods for pose estimation based on imagery input. This paper investigates the opportunity of applying CNNs to estimate the 6-DoF pose between an underwater vehicle and a fixed object, without the need for artificial markers.**

***Index Terms*—Underwater robotics, intervention, autonomy, CNN, pose estimation, localization**

I. Introduction

Increased level of autonomy in underwater vehicle operations is of high interest to reduce the cost of intervention missions and increase the frequency of inspections. Performing manipulation tasks on subsea installments require extremely precise maneuvering of the vehicle, addressing the need for a high-precision localization system. A method for predicting the vehicle position and rotation together referred to as the 6-DoF pose, relative to the object subject to manipulation is therefore of high interest. Multiple computer vision (CV) based approaches are suggested for this purpose. However, the CV methods rely on having pre-installed artificial markers available on the subsea structures [1] [2]. The current generation of deployed subsea infrastructure does not have such markers available, rendering the CV methods infeasible to use in practice. Recent advances within deep learning, particularly convolutional neural networks (CNNs) shows promise with respect to pose estimation based on imagery input. This paper investigates the application for such CNN methods to estimate 6-DoF pose in an underwater environment as an alternative to existing artificial marker-based methods.

II. Method

A. Model

CNNs are a particular type of neural networks operating on multidimensional data, shown to be extremely successful in computer vision applications [3]. The success of CNNs came from the increased accuracy to image classification competitions. However, recent work shows promising results within regression problems as well. The PoseNet architecture is an example of a regression network, and estimates a 6-DoF pose from one single image [4]. This network is shown to be successful in terrestrial domains. This paper investigates the usage of PoseNet for subsea applications.

The PoseNet is a 23 layer deep CNN architecture, regressing relative position and orientation simultaneously. It takes a 224x224 image as input and outputs a pose vector $\boldsymbol{\eta}$ of 7 entries, composed of the 3D camera position $\boldsymbol{p} \in \mathbb{R}^3$ and orientation represented by the quaternion $\boldsymbol{q} \in \mathcal{S}^3$, such that,

$$\boldsymbol{\eta} = [\boldsymbol{p}^T, \boldsymbol{q}^T]^T \in \mathbb{R}^3 \times \mathcal{S}^3 \tag{1}$$

Notational $\hat{(\cdot)}$ denotes an estimate of a variable, such that $\boldsymbol{p}$, $\hat{\boldsymbol{p}}$, $\boldsymbol{q}$ and $\hat{\boldsymbol{q}}$ represents the actual and predicted position and orientation, respectively. The network is trained with stochastic gradient descent, minimizing the loss function seen in Eq. (2).

$$Loss(\boldsymbol{\eta}, \hat{\boldsymbol{\eta}}) = ||\hat{\boldsymbol{p}} - \boldsymbol{p}||_2 + \beta \left|\left| \hat{\boldsymbol{q}} - \frac{\boldsymbol{q}}{||\boldsymbol{q}||} \right|\right|_2 \tag{2}$$

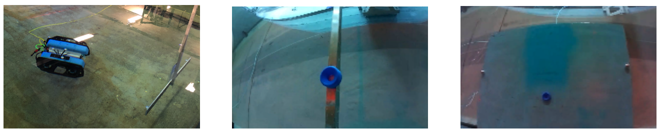
where β is a scaling factor used to balance the expected value of the position error and the orientation error.

B. Dataset

The datasets $\mathcal{D}$ contains n datapoints $\boldsymbol{d}_i, i = 1 \dots n$, where a data point $\boldsymbol{d}_i$ contains an image from the ROV camera as $\boldsymbol{X}_i$ and a relative pose as $\boldsymbol{\eta}_i$ such that,

$$\boldsymbol{d}_i = \{\boldsymbol{X}_i, \boldsymbol{\eta}_i\} \tag{3}$$

The images in the datasets used for training and testing the network contain underwater images of a plastic ring modeling a subsea valve. These datasets were produced in the marine cybernetics laboratory (MCLab) at NTNU in Trondheim, using the research ROV BlueROV2 equipped with a camera. Fig. 1a. illustrates the laboratory setup. The pose vector $\boldsymbol{\eta}^c_{o/c}$ denotes the pose of the object, denoted $\{o\}$, relative to the camera, denoted $\{c\}$, in the camera frame and consists of the position $\boldsymbol{p}^c_{o/c} \in \mathbb{R}^3$ and a unit quaternion $\boldsymbol{q} \in \mathcal{S}^3$. MCLab enables localization of the ROV and object using an underwater camera system from Qualisys. The Qualisys measurements provided the pose labels for training the network. Fig. 1b and Fig. 1c. shows sample images from the gathered datasets.



(a) (b) (c)

Fig. 1: Laboratory setting with BlueROV2 in the MCLab pool with the mock-up valve attached to two different objects to create variation in the background.

TABLE I: Summary statistics of the pose errors separated into the positional and orientation errors respectively.

	Median	**Mean**	**Std. dev**
Position error [mm]	19.16	23.88	33.86
Orientation error [deg]	0.38	0.72	3.02

III. Results and Discussion

The collected data used for training contained 11K images and the network trained on the dataset in 150 epochs. The performance assessment uses the total position as the Euclidean norm of the 3D position, and the total orientation error measured as the relative angle between the actual and predicted quaternion. Tab. I shows the median, mean, and standard deviation of these errors.

The results shows promise and illustrate that the PoseNet architecture is applicable to the underwater 6-DoF pose estimation problem. The accuracy achieved with this method outperforms the accuracy obtained with CV methods and artificial markers in [2]. However, the results exhibit some outliers in the obtained errors of the pose estimates. Fig. 2, contains plots of the errors in each DoF, for all the samples in the test validation set. The outliers occur to varying extents in the different DoFs and tend to occur in multiple DoFs simultaneously. The latter can relate to the network architecture, as all the DoFs are regressed simultaneously. One reason for the large outliers is the process at which the labeling took place. The process of labeling the data was automated and based on the raw Qualisys data mapped by a time-stamp to the camera of the ROV. The Qualisys system sometimes lost tracking of the ROV and when re-tracking occurred the estimated axes applied to the vehicle often exhibited wrong rotations for a couple of samples. Erroneous labels causes detrimental effects to the training and could potentially explain the outliers.

IV. Conclusion

This paper presents a method for estimating 6-DoF pose between an underwater vehicle and a fixed object, without the need for artificial markers. The CNN architecture PoseNet is trained and tested on underwater datasets, and accuracy in the range of 19 mm and 0.4 degrees for position and orientation, respectively, is achieved. The solution exhibited outliers and future work includes reducing the number of outliers, as well as, applying the solution to closed-loop station keeping of a ROV.

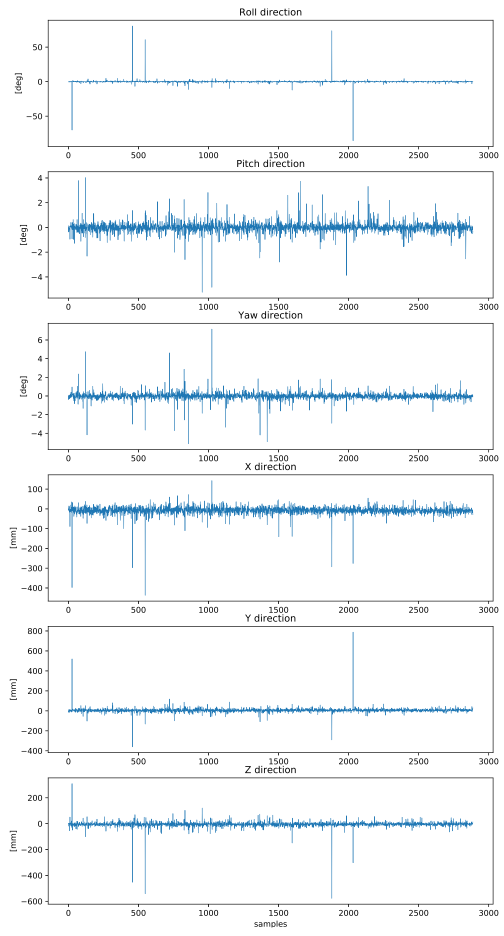


Fig. 2: Error variables in six degrees of freedom. The quaternion $\boldsymbol{q}$ is transformed into the Euler Angles around the primary axes of rotations.

References

[1] E. Henriksen, I. Schjlberg, and T. Gjersvik, "Vision based localization for subsea intervention," 2017.

[2] A. Gomez Chavez, C. Mueller, T. Doernbach, and A. Birk, "Underwater navigation using visual markers in the context of intervention missions," *International Journal of Advanced Robotic Systems*, vol. 16, 03 2019.

[3] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," Cambridge, Mass, 2017.

[4] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," 2015.

Appendix B

Complementary Theory

Activation Functions

Step Function

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & \text{otherwise} \end{cases} \tag{B.1}$$

The step function has a binary output. Neurons with this activation function are called *perceptrons*, and they serve as binary classifiers.

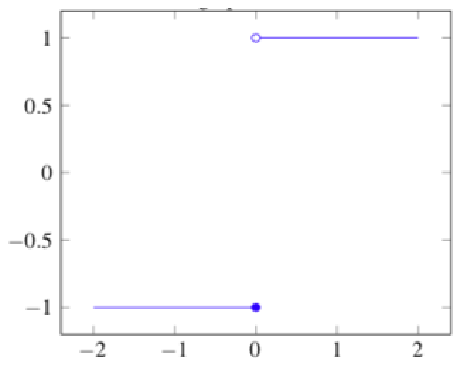


Figure B.1: Step function

Linear Function

$$f(x) = ax \tag{B.2}$$

The linear function simply scales the input. This function will therefore not introduce nonlinearity to the network.

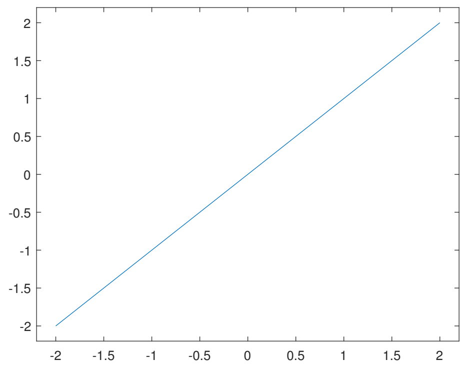


Figure B.2: Linear function

Sigmoid Function

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{B.3}$$

The sigmoid function squeezes the input value from an infinite range into a value between 0 and 1. The output value can be considered to be a probability, very close to either 0 or 1 for most input values.

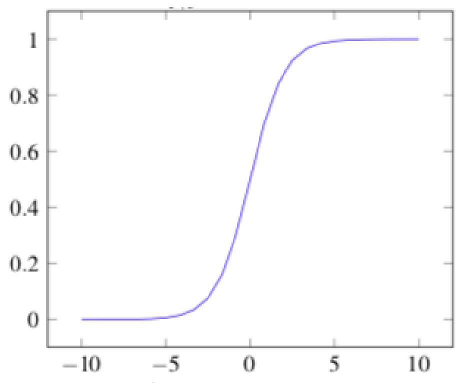


Figure B.3: Sigmoid function

Tanh

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{B.4}$$

Tanh is pretty similar to the sigmoid function, except that it returns a value between -1 and 1. Because of this, tanh handles negative values well.

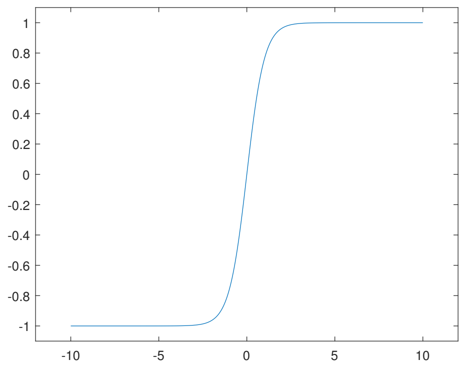


Figure B.4: Tanh function

Softmax

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{k=n} e^{x_k}} \tag{B.5}$$

Softmax is a generalization of sigmoid, as it can have multiple boundaries and outputs probabilities between 0 and 1 for several categories. Softmax is therefore well suited for the output layer of a network representing a classifier with more than two classes. In Equation B.5, k is the number of output classes.

ReLU

$$f(x) = \begin{cases} x & x \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{B.6}$$

The Rectified Linear Unit, also known as ReLU, activates a node if the input is above some threshold, that is implicitly defined by the weights of previous layers. Whenever the input to ReLU is above zero, it represents a linear relationship, while it returns zero for all values below zero.

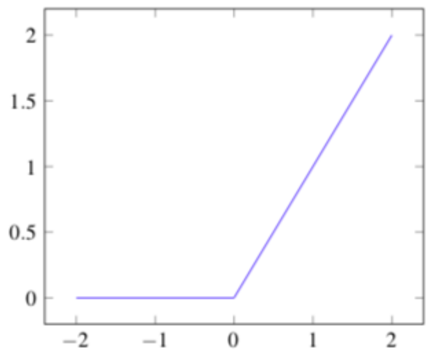


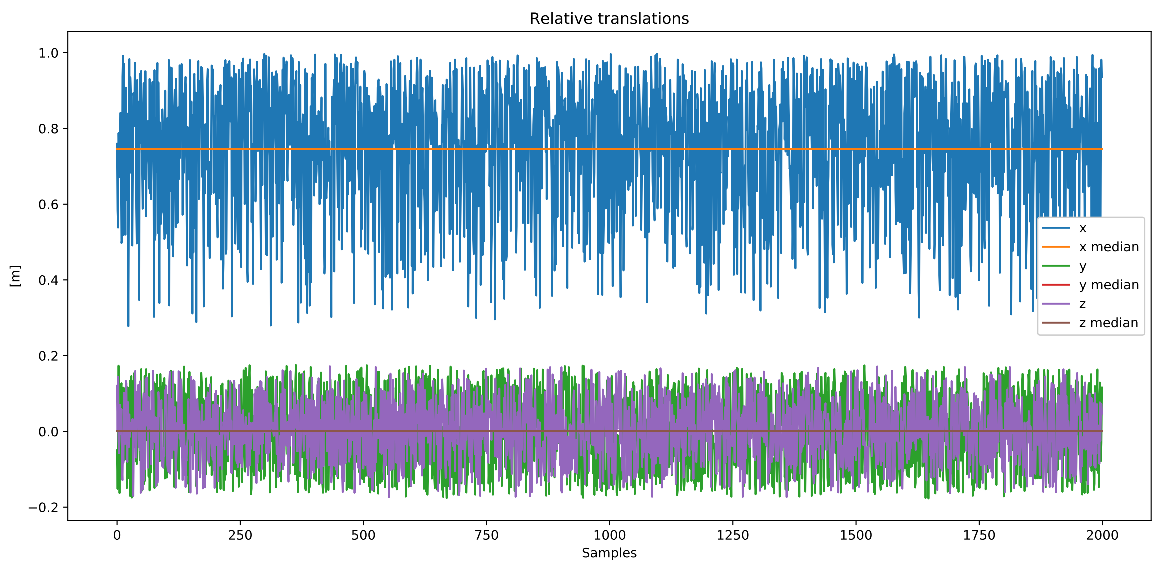
Figure B.5: ReLU function

Appendix C

Details of Datasets

This appendix contains plots of the 6-DoF relative translation and orientation between the camera and the target object, in the different datasets presented in Chapter 5 and Chapter 6. The median of the relative pose is also plotted for each DoF.

C.1 Dataset 0



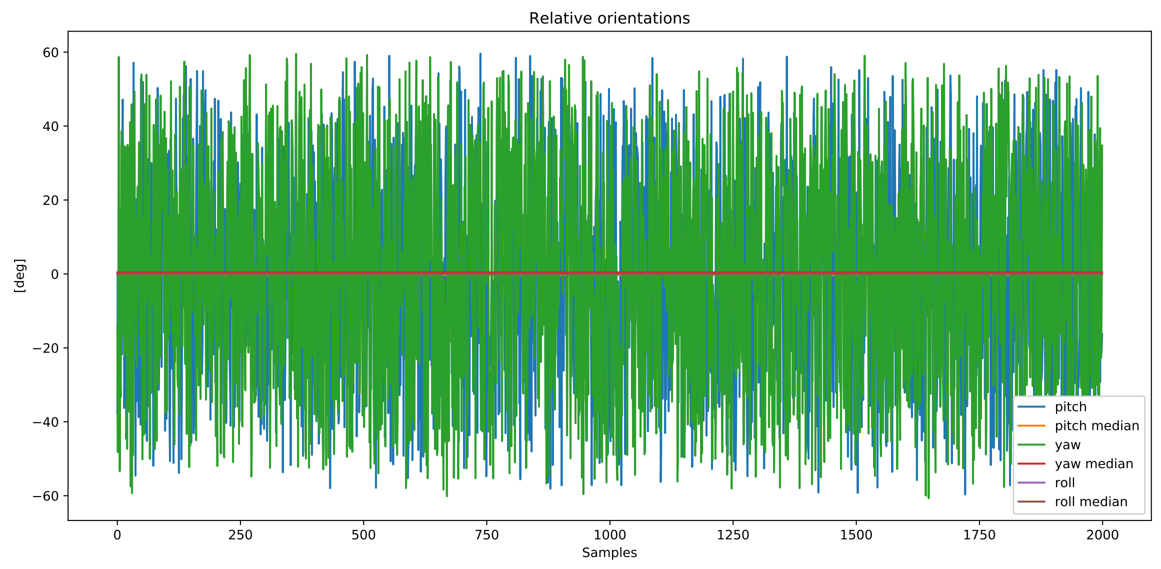


Figure C.1: Relative poses dataset 0

The relative orientations in roll direction are close to zero for all the samples in this dataset, which is not clear in the plot in Figure C.1.

C.2 Dataset 1

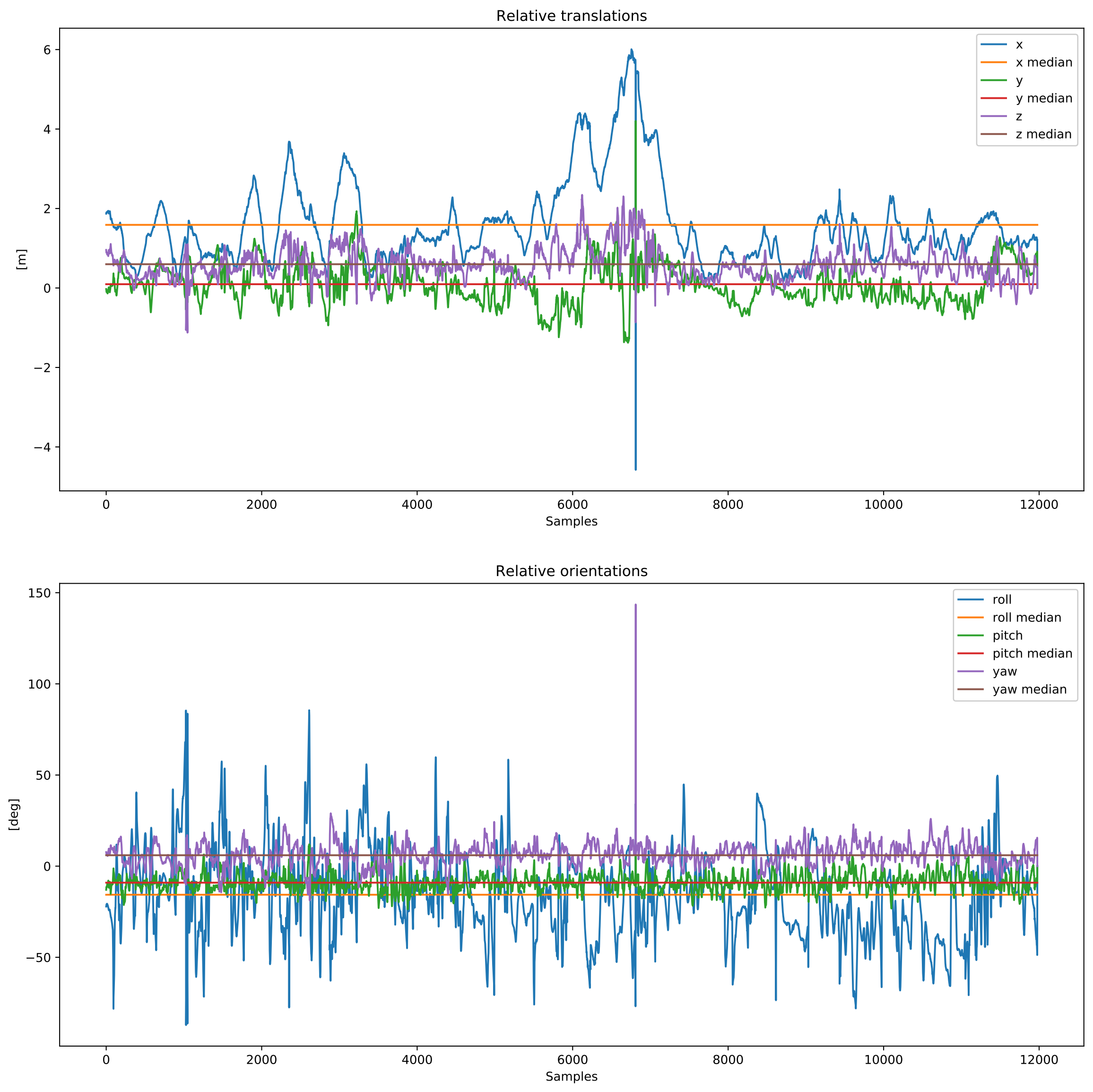
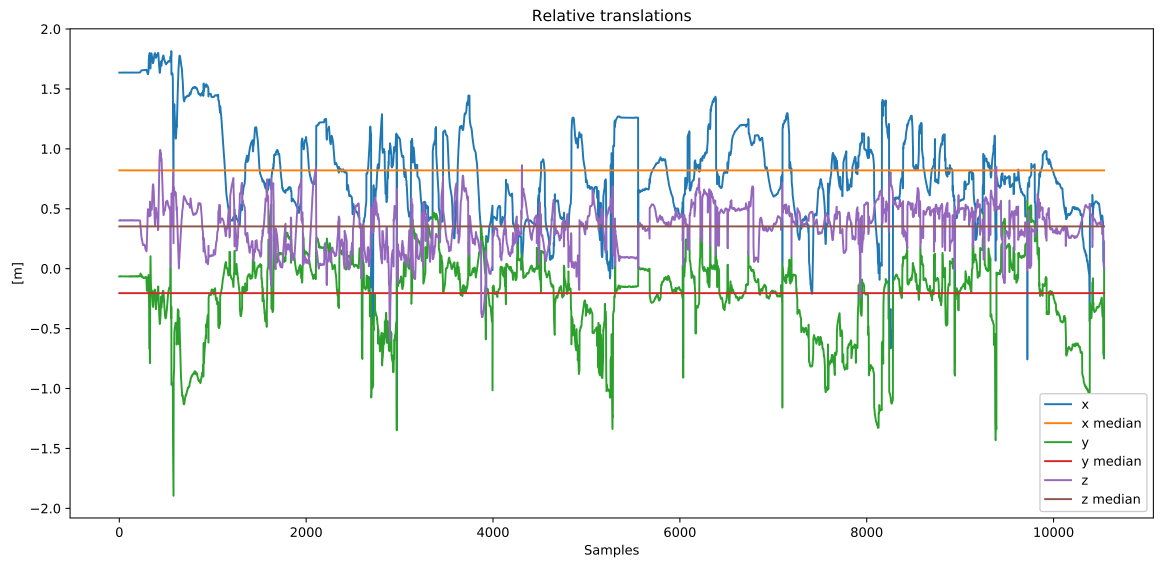


Figure C.2: Relative poses dataset 1

C.3 Dataset 2



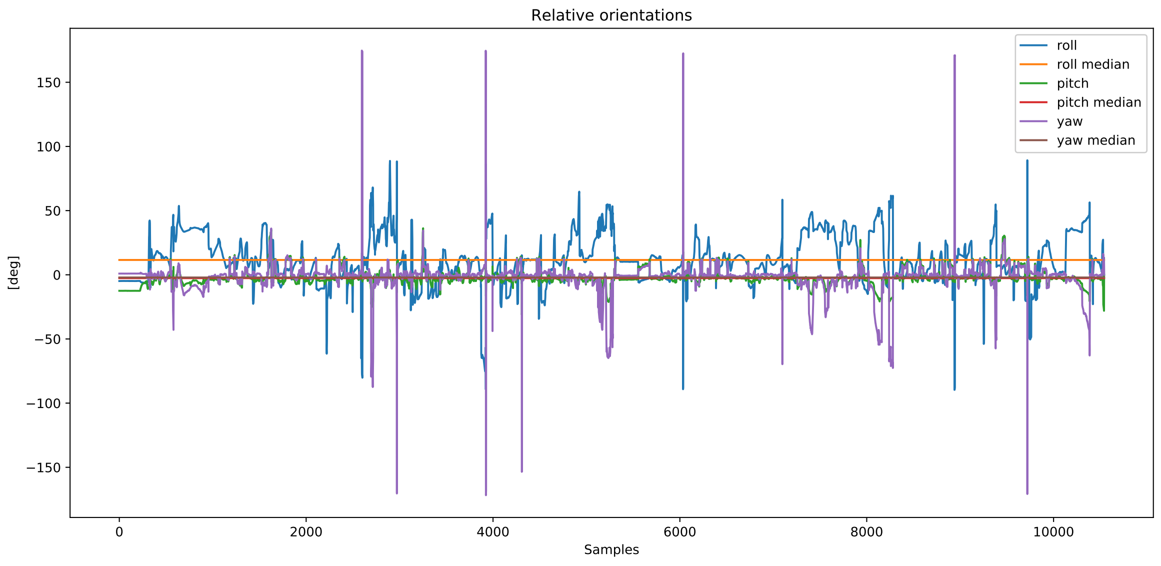
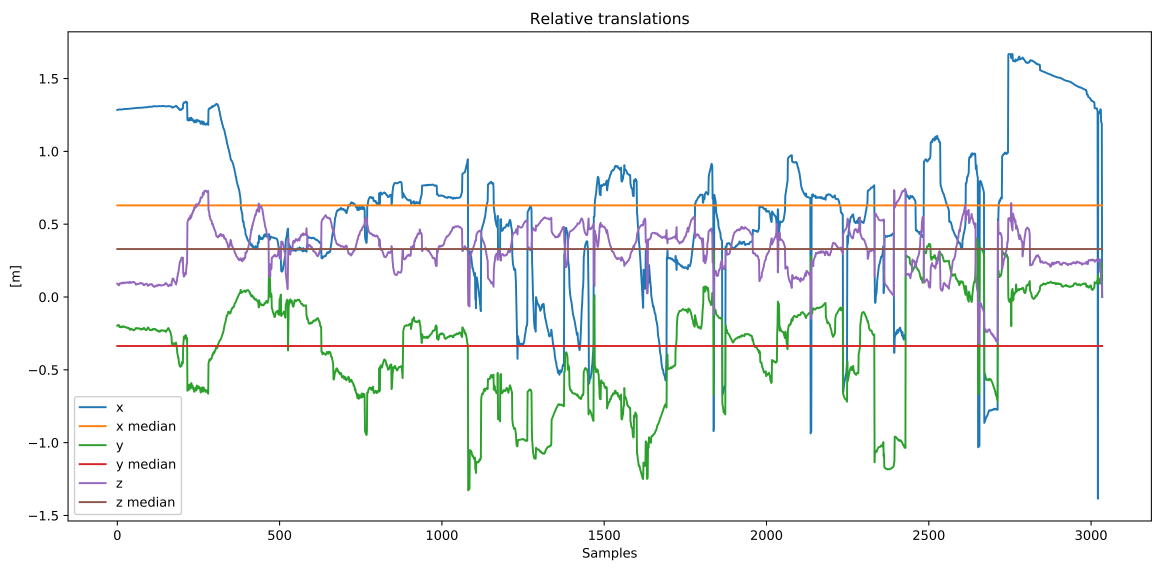


Figure C.3: Relative poses dataset 2

C.4 Dataset 3



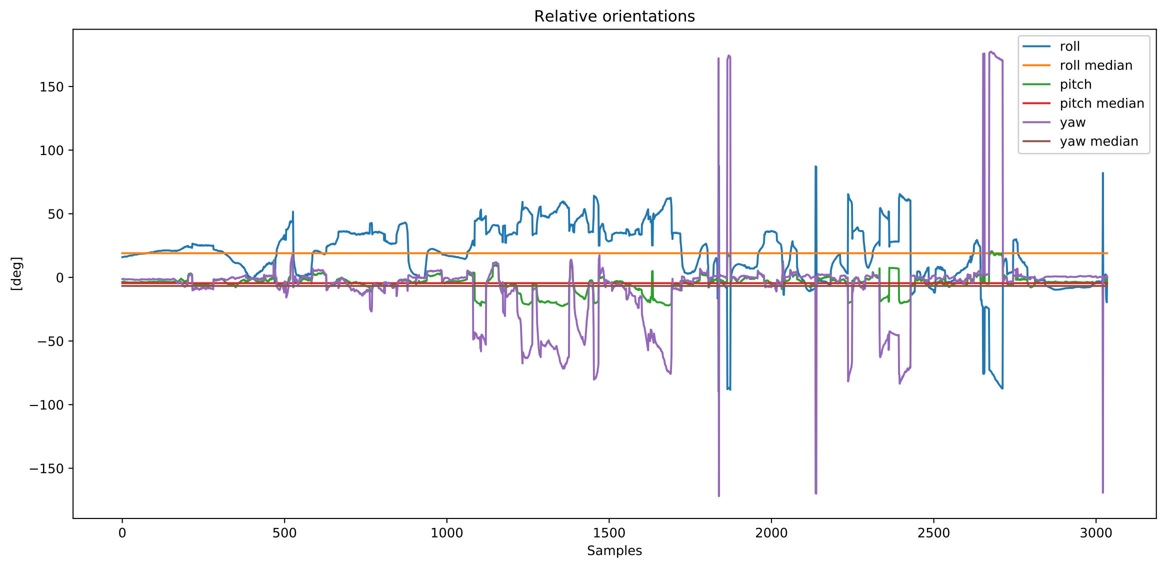
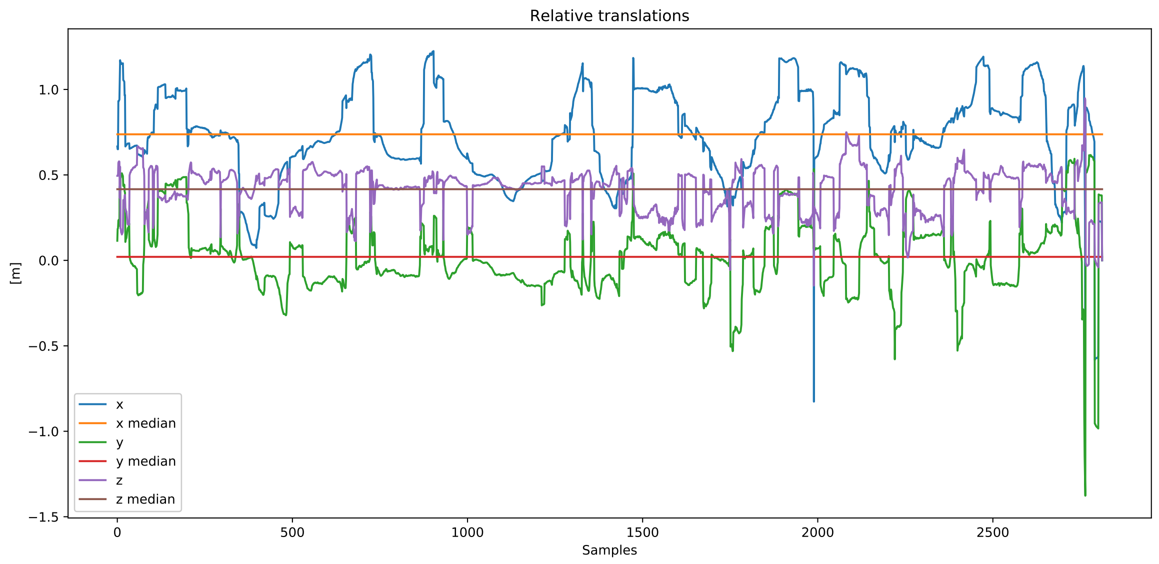


Figure C.4: Relative poses dataset 3

C.5 Dataset 4



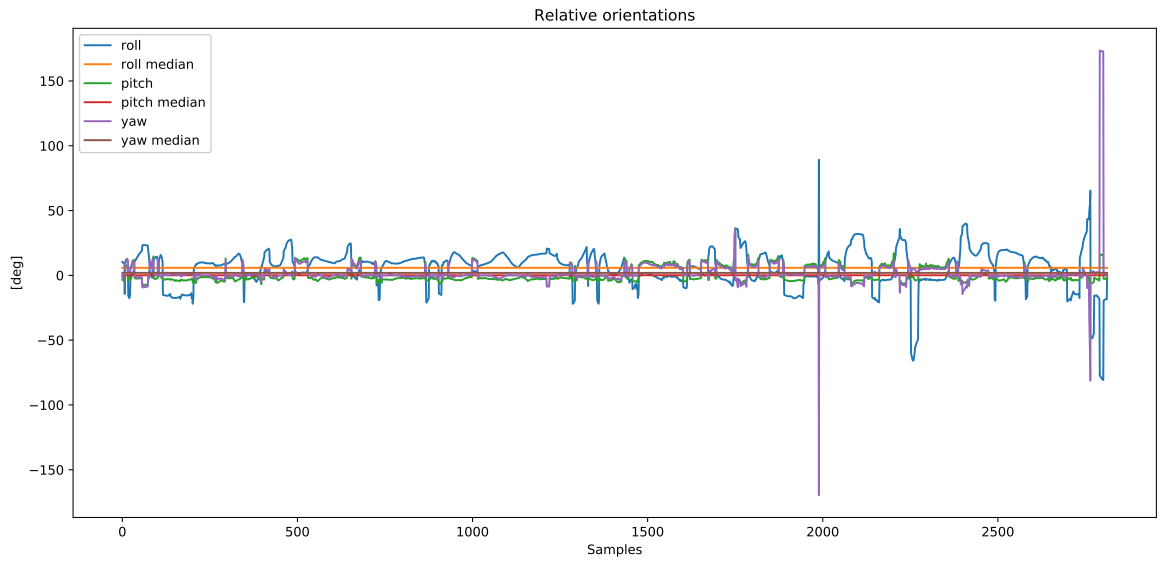


Figure C.5: Relative poses dataset 4

C.6 Dataset 5

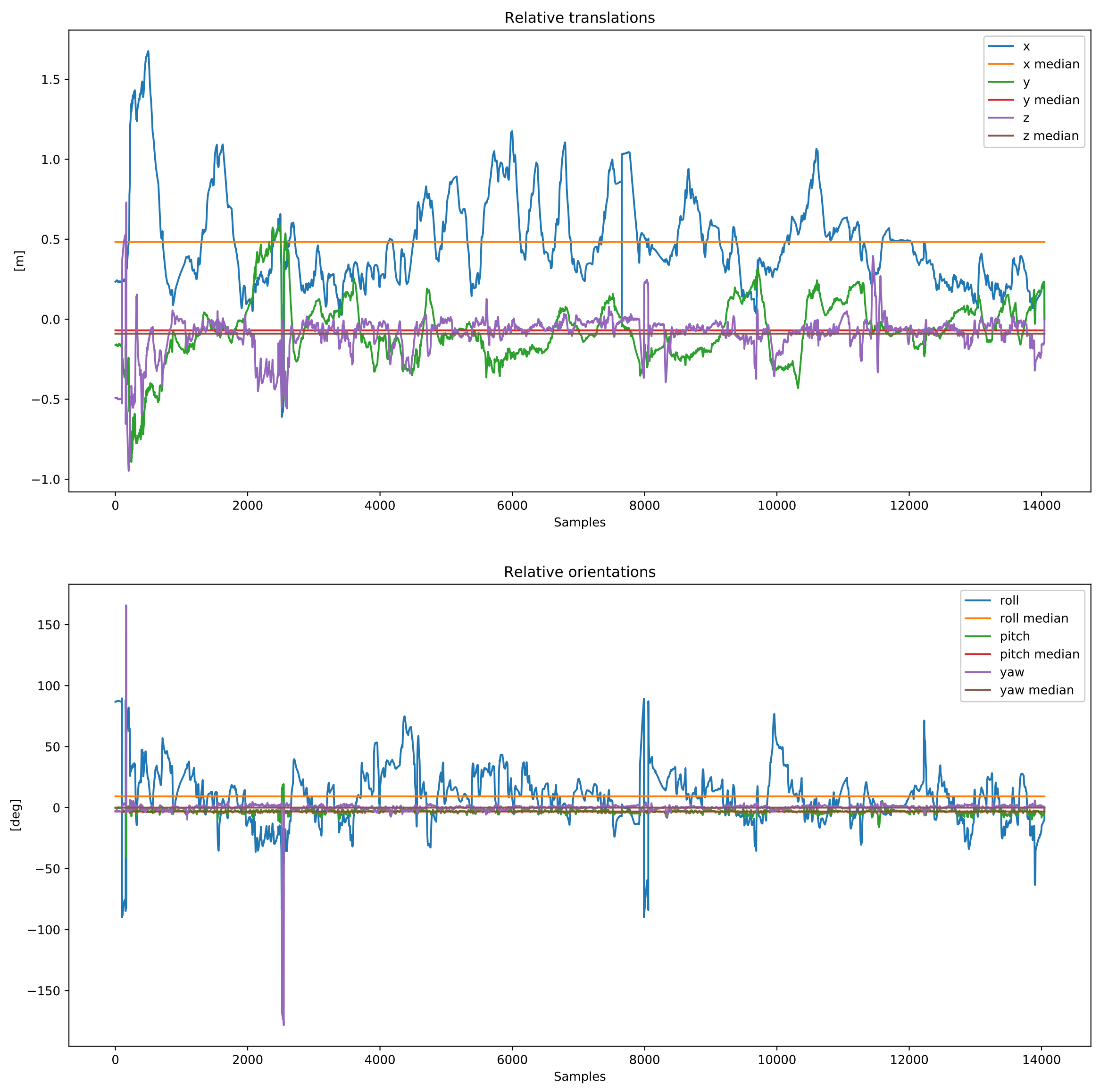
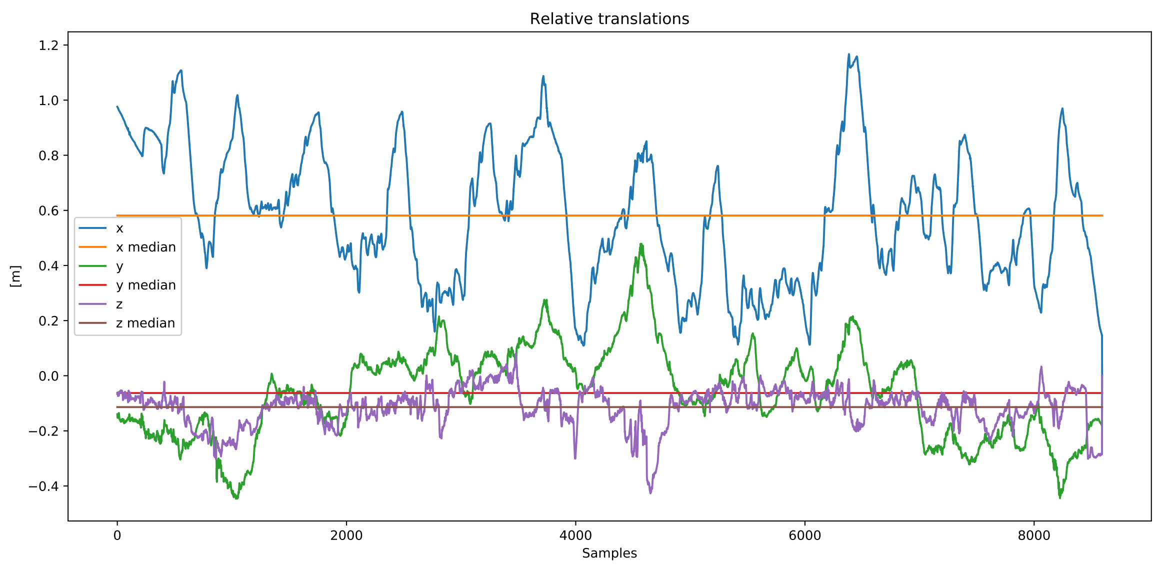


Figure C.6: Relative poses dataset 5

C.7 Dataset 6



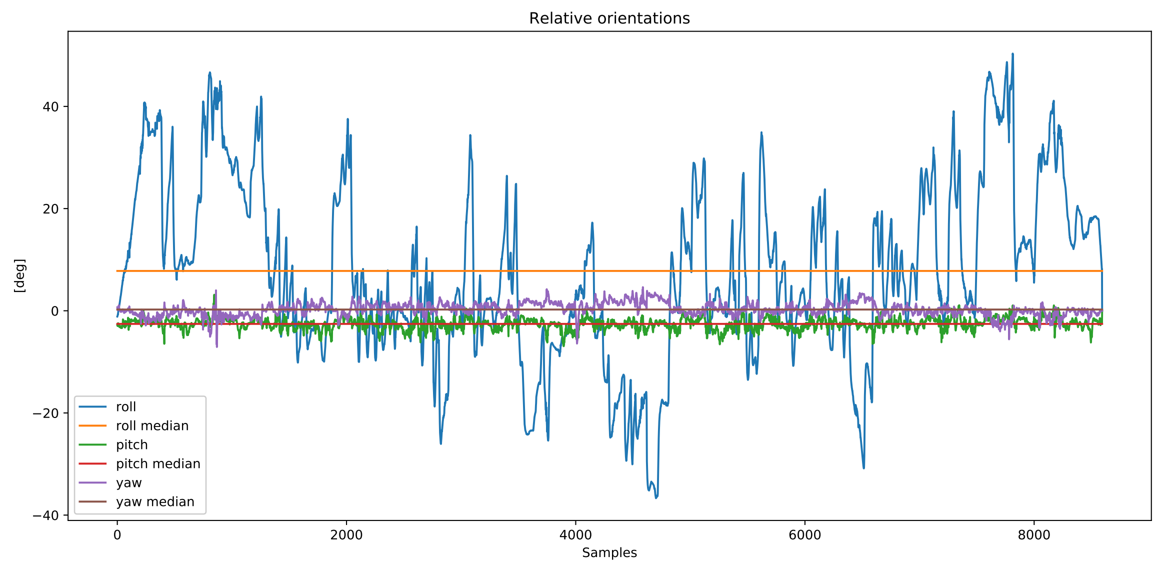


Figure C.7: Relative poses dataset 6

Appendix D

Additional Results

D.1 Case 1

Data ID	Description	# Train images	# Test images	Splitting
1	AruCo marker on metal box, camera distance maximum 1 m	3 600	826	Random

Table D.1: Case 1: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	38.95	47.45	65.96
Angle error [deg]	1.112	1.793	4.709

Table D.2: Case 1: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

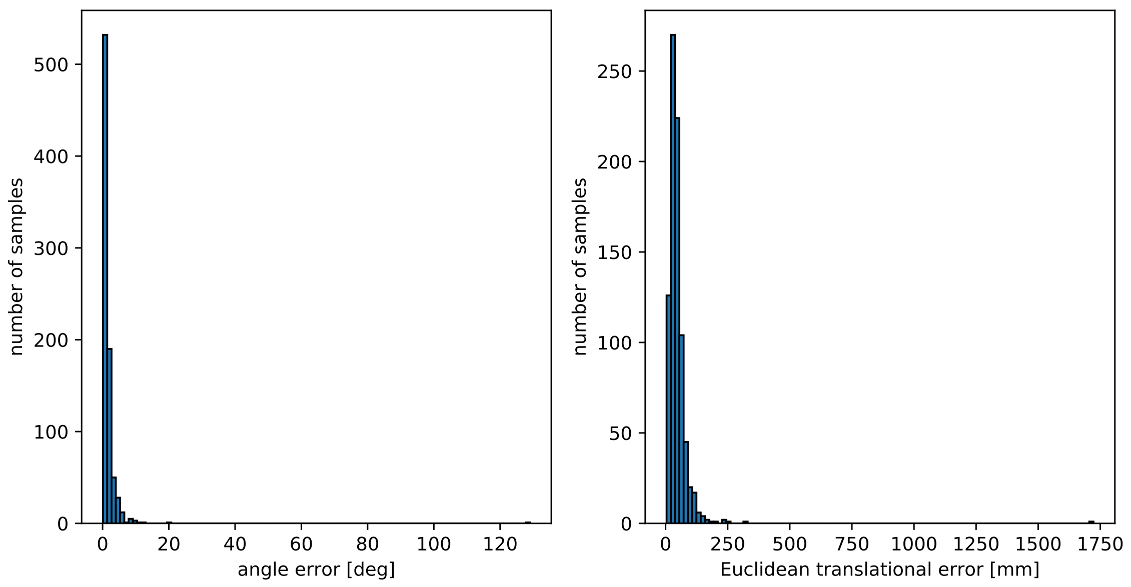


Figure D.1: Case 1: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	Mean	Standard Deviation
X [mm]	-0.87	60.19
Y [mm]	-0.43	34.40
Z [mm]	-7.76	42.28
Roll [deg]	-0.450	1.153
Pitch [deg]	-0.073	0.727
Yaw [deg]	0.155	0.664

Table D.3: Case 1: Summary statistics of pose errors in each DoF

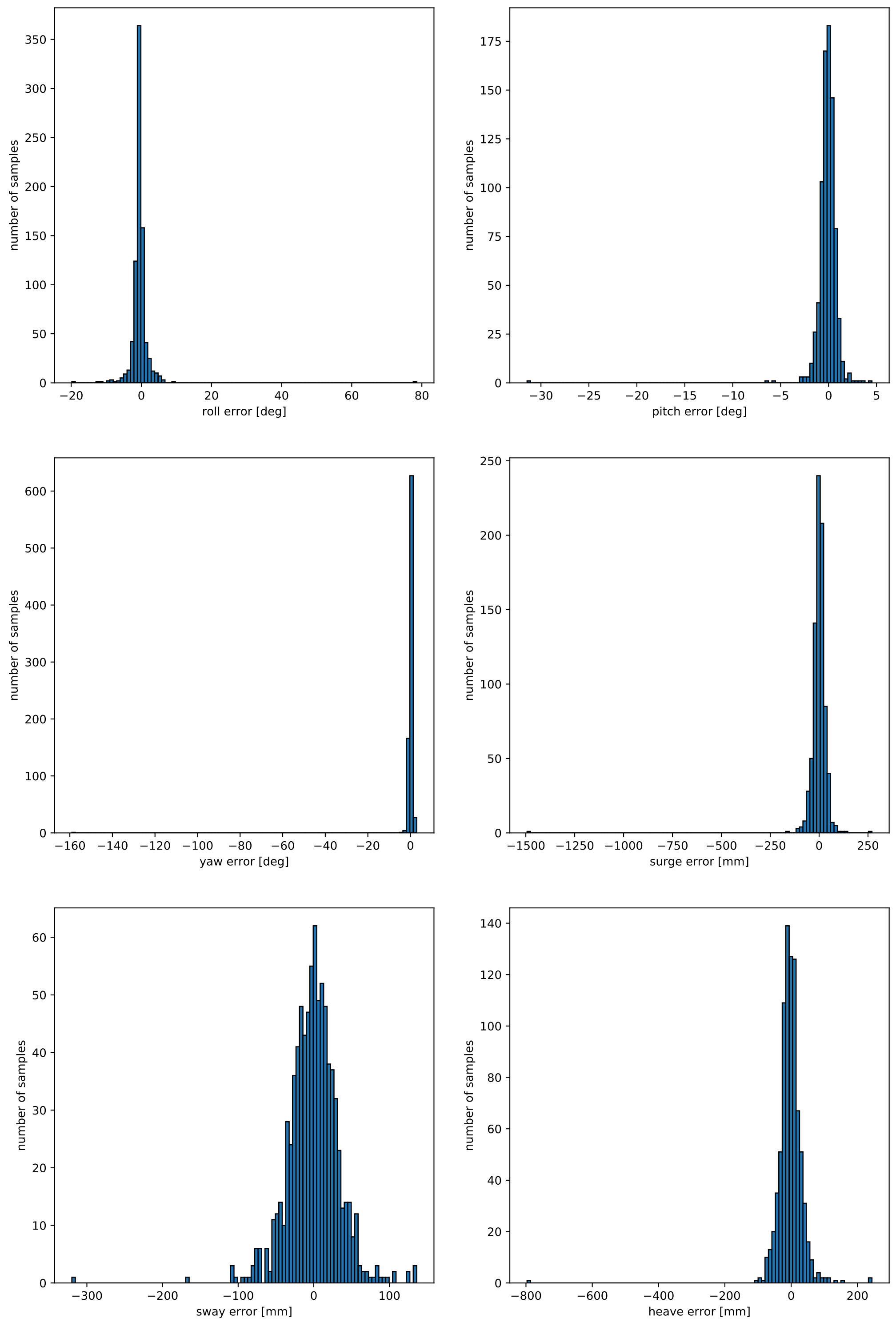


Figure D.2: Case 1: Pose errors in each DoF

Measurement	**Amount**
Translation errors less than 10 % of distance, $T_{\%}$	86 %
Rotation errors less than 10 % of rotation, $A_{\%}$	79 %

Table D.4: Case 1: Amount of test data satisfying the 10-percent requirements

D.2 Case 2

Data ID	**Description**	**# Train images**	**# Test images**	**Splitting**
1	AruCo marker on metal box, camera distances up to 7 m	8 000	3 000	Random

Table D.5: Case 2: Details of train and test datasets

Measurement	**Median**	**Mean**	**Standard Deviation**
Euclidean translation error [mm]	89.98	106.97	84.63
Angle error [deg]	1.154	1.632	3.493

Table D.6: Case 2: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

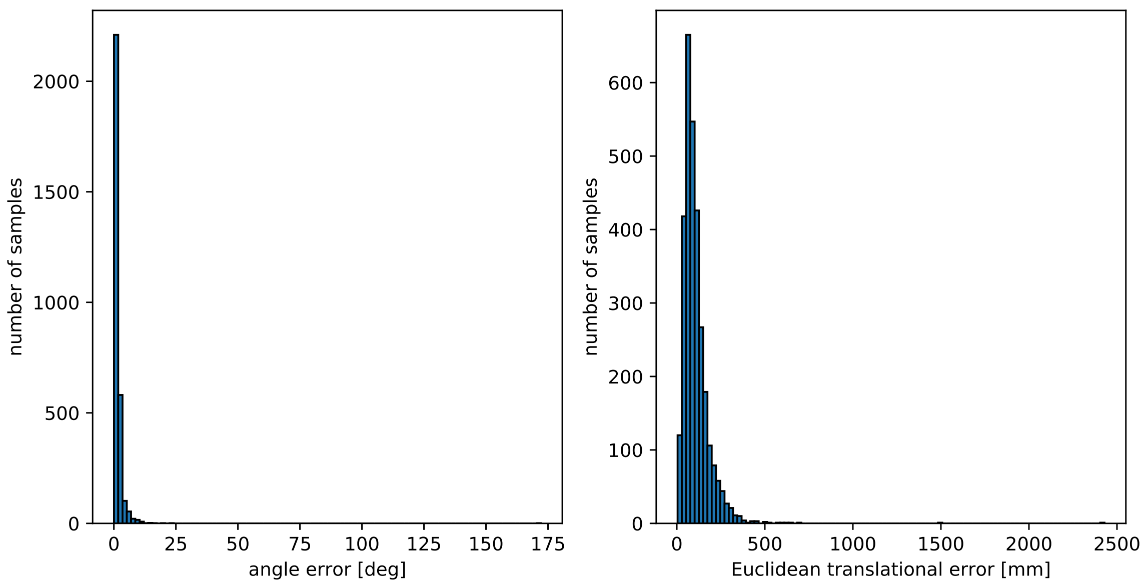


Figure D.3: Case 2: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	Mean	Standard Deviation
X [mm]	24.80	87.94
Y [mm]	21.11	70.46
Z [mm]	-11.44	68.65
Roll [deg]	-0.271	1.139
Pitch [deg]	0.087	0.827
Yaw [deg]	0.124	0.706

Table D.7: Case 2: Summary statistics of pose errors in each DoF

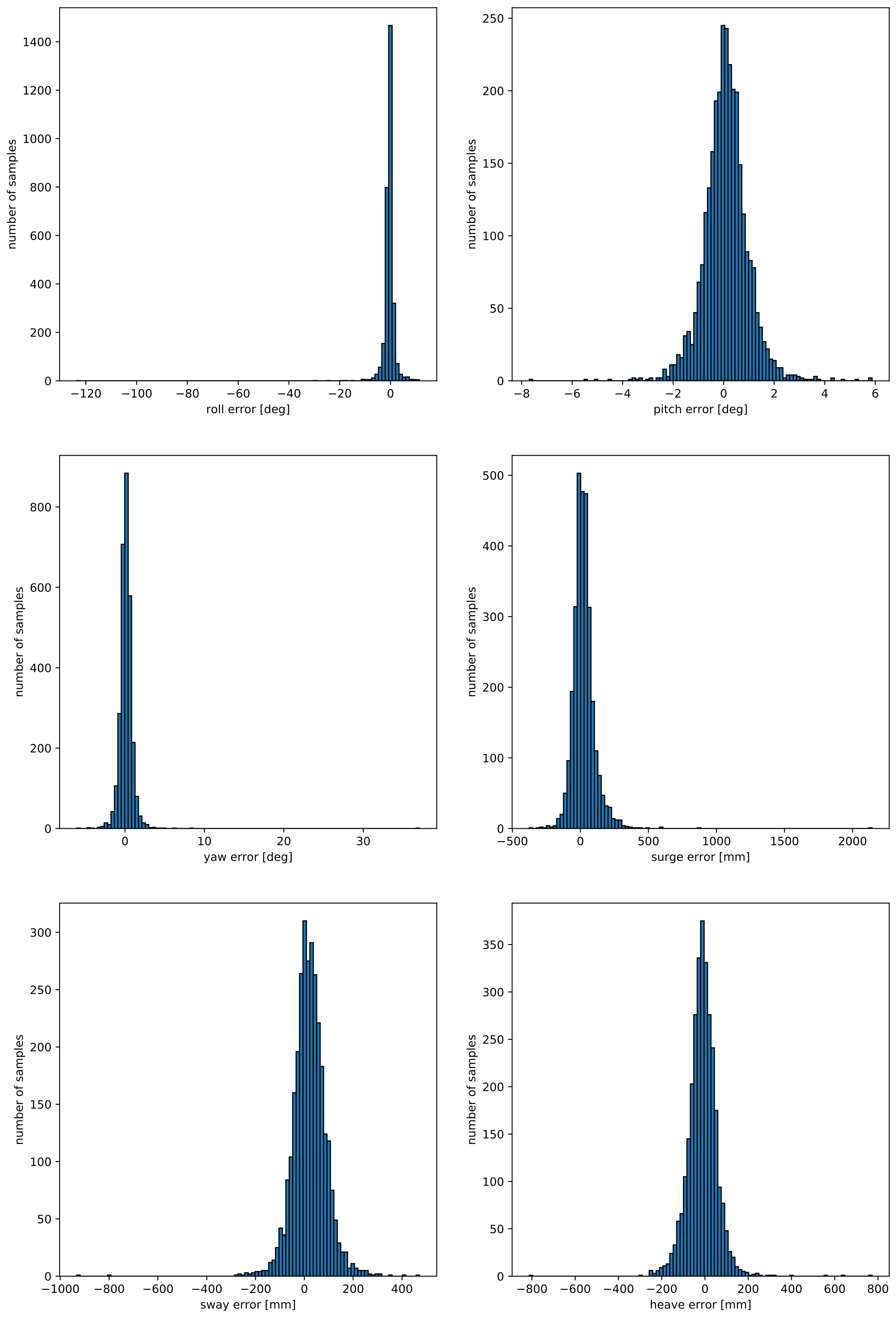


Figure D.4: Case 2: Pose errors in each DoF

Measurement	Amount
Translation errors less than 10 % of distance, $T_{\%}$	82 %
Rotation errors less than 10 % of rotation, $A_{\%}$	79 %

Table D.8: Case 2: Amount of test data satisfying the 10-percent requirements

D.3 Case 3

Data ID	Description	# Train images	# Test images	Splitting
2, 3, 4	AruCo marker on metal stick	10 500		Temporal
2	AruCo marker on metal stick, all lights on		986	Temporal

Table D.9: Case 3: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	81.96	246.90	343.32
Angle error [deg]	2.481	12.630	20.973

Table D.10: Case 3: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

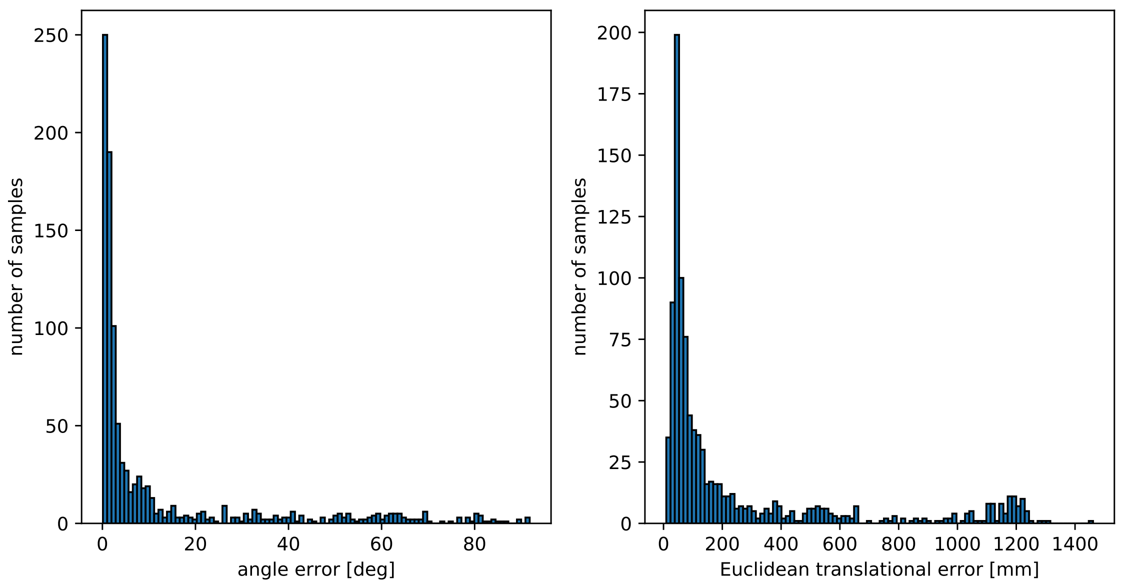


Figure D.5: Case 3: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	24.59	286.89
Y [mm]	-13.06	294.08
Z [mm]	-14.95	95.11
Roll [deg]	-0.460	14.866
Pitch [deg]	0.823	8.484
Yaw [deg]	1.381	21.882

Table D.11: Case 3: Summary statistics of pose errors in each DoF

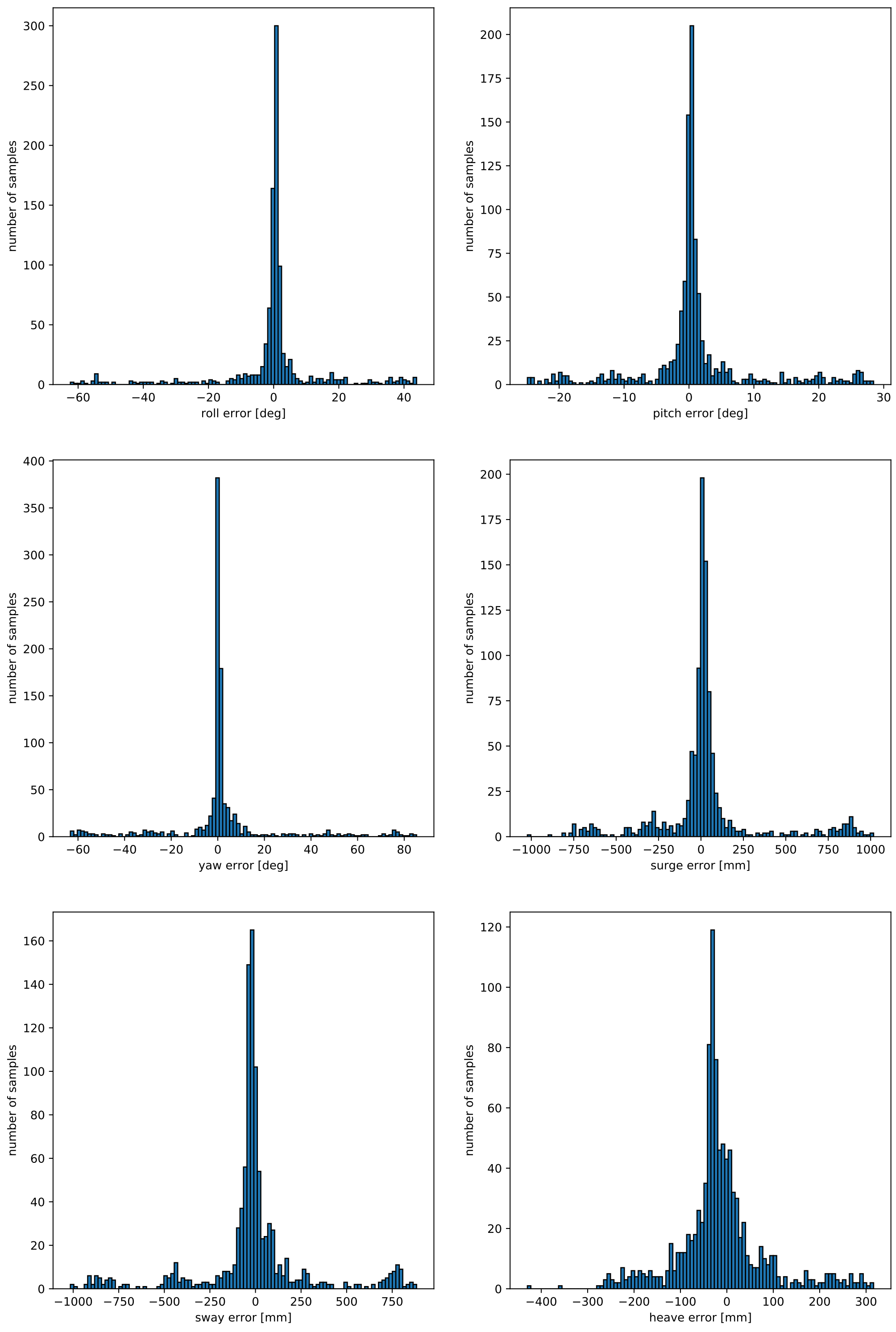


Figure D.6: Case 3: Pose errors in each DoF

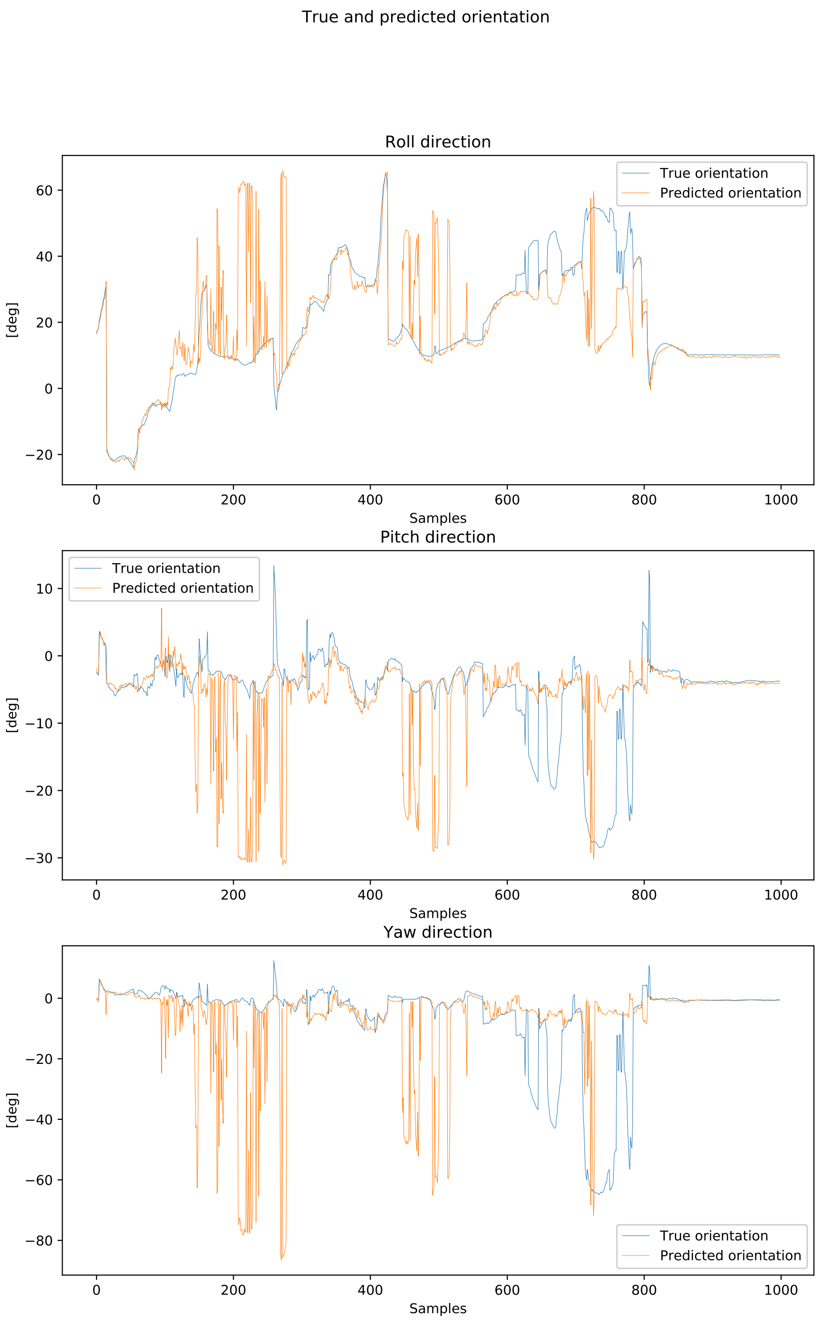


Figure D.7: Case 3: True and predicted orientation in roll, pitch and yaw

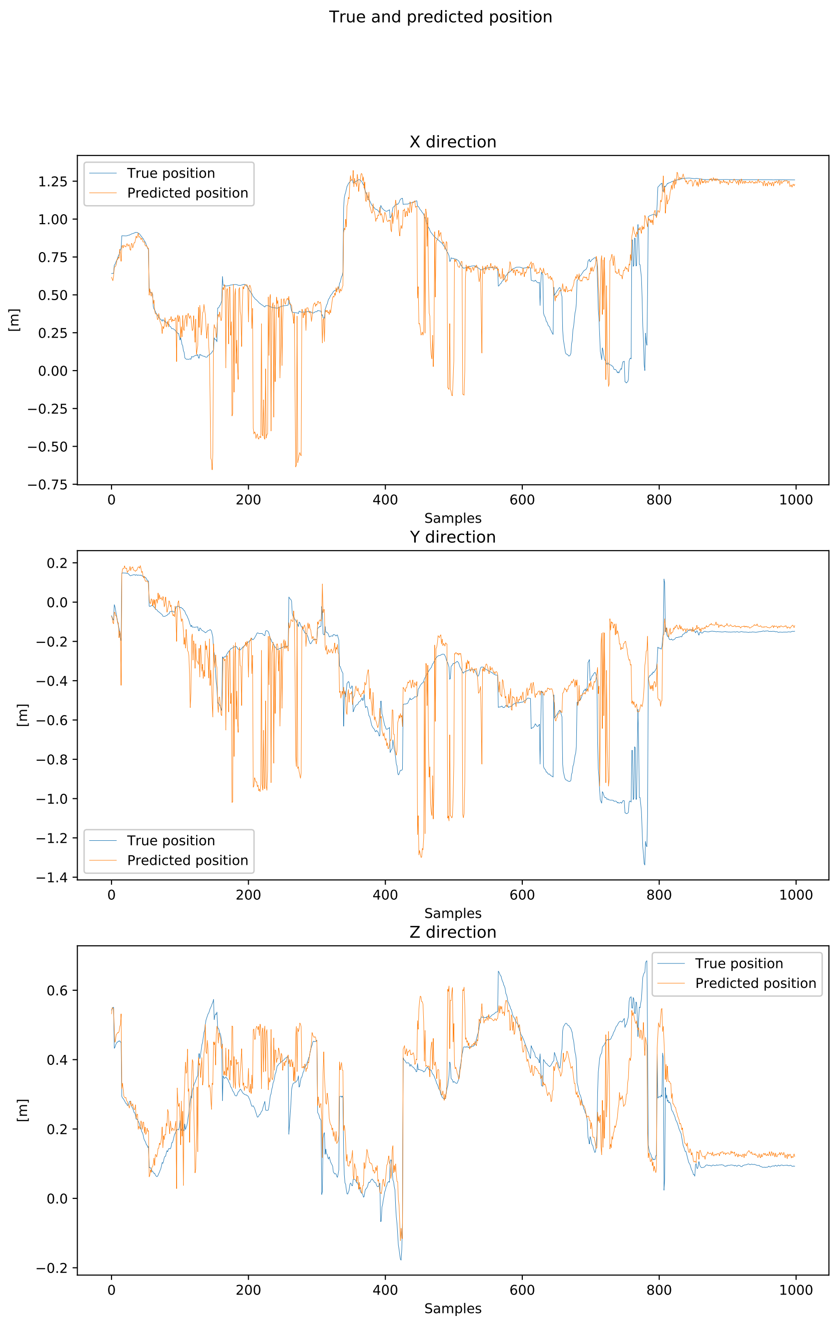


Figure D.8: Case 3: True and predicted position in surge, sway and heave

D.4 Case 4

Data ID	Description	# Train images	# Test images	Splitting
2, 3, 4	AruCo marker on metal stick	10 500		Temporal
3	AruCo marker on metal stick, reduced lights A		534	Temporal

Table D.12: Case 4: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	125.35	285.61	464.52
Angle error [deg]	1.887	18.586	50.296

Table D.13: Case 4: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

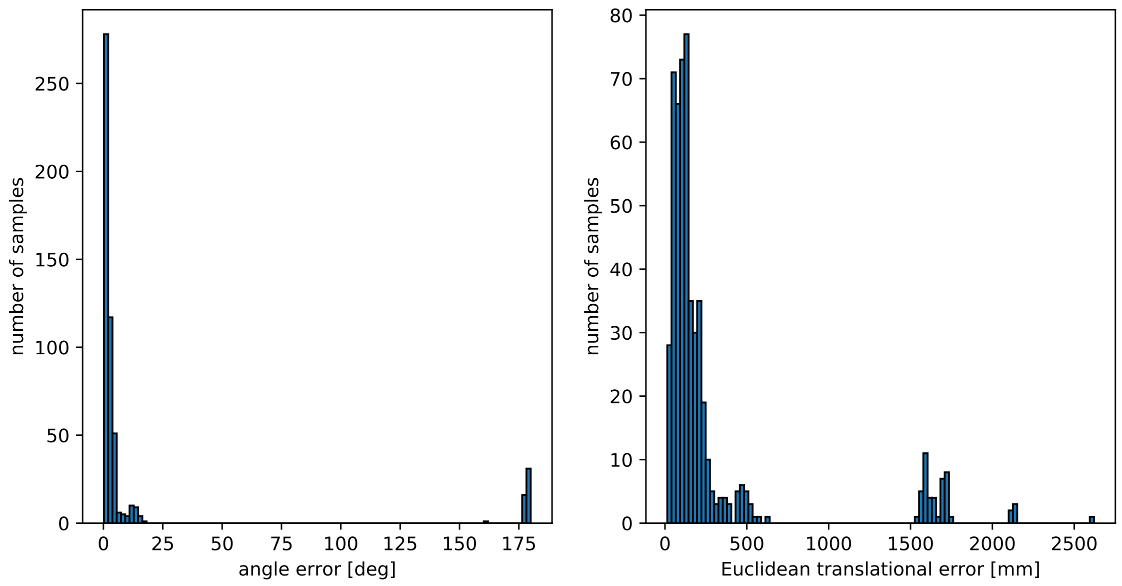


Figure D.9: Case 4: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-51.65	456.25
Y [mm]	-17.48	274.79
Z [mm]	-42.13	94.51
Roll [deg]	-5.990	18.366
Pitch [deg]	1.756	5.058
Yaw [deg]	15.653	49.587

Table D.14: Case 4: Summary statistics of pose errors in each DoF

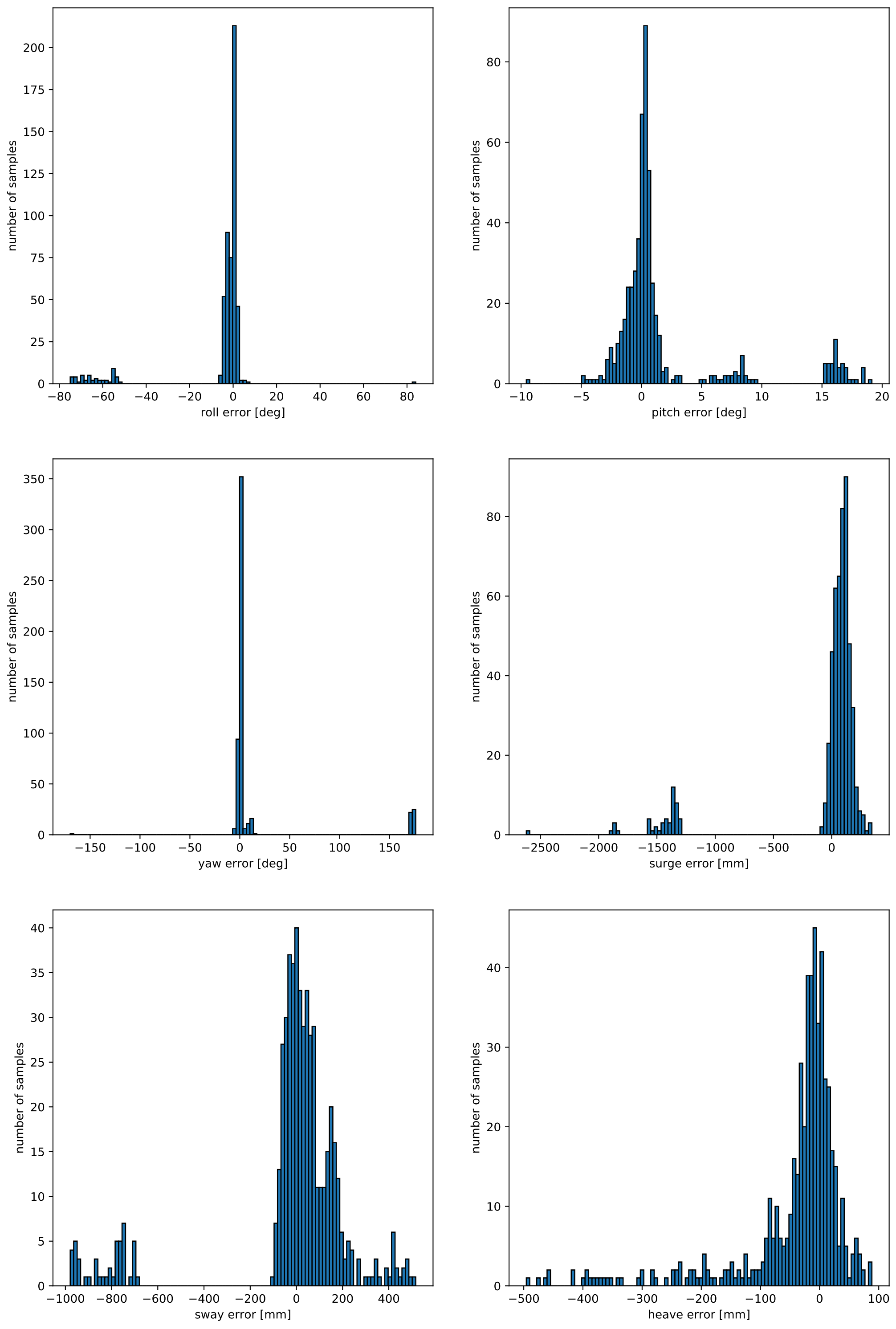


Figure D.10: Case 4: Pose errors in each DoF

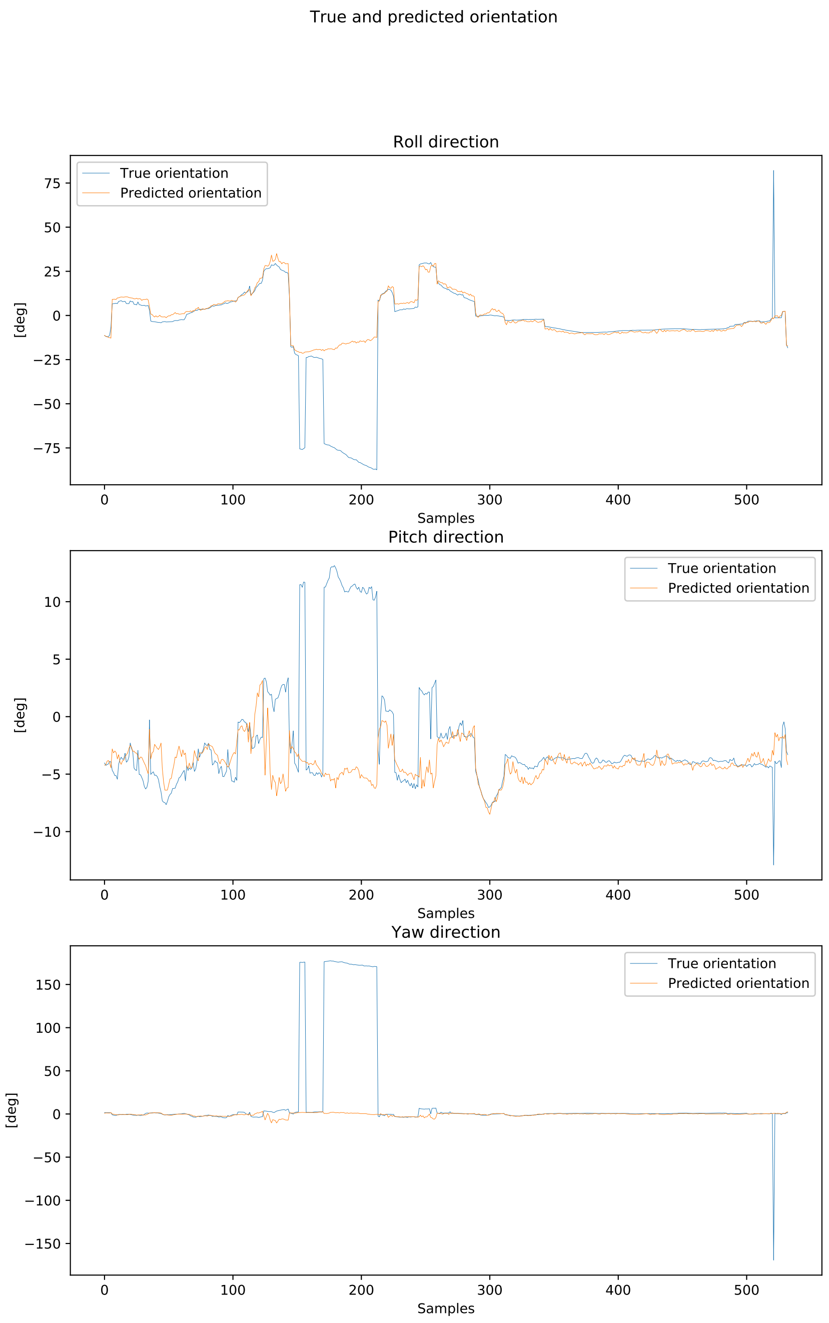


Figure D.11: Case 4: True and predicted orientation in roll, pitch and yaw

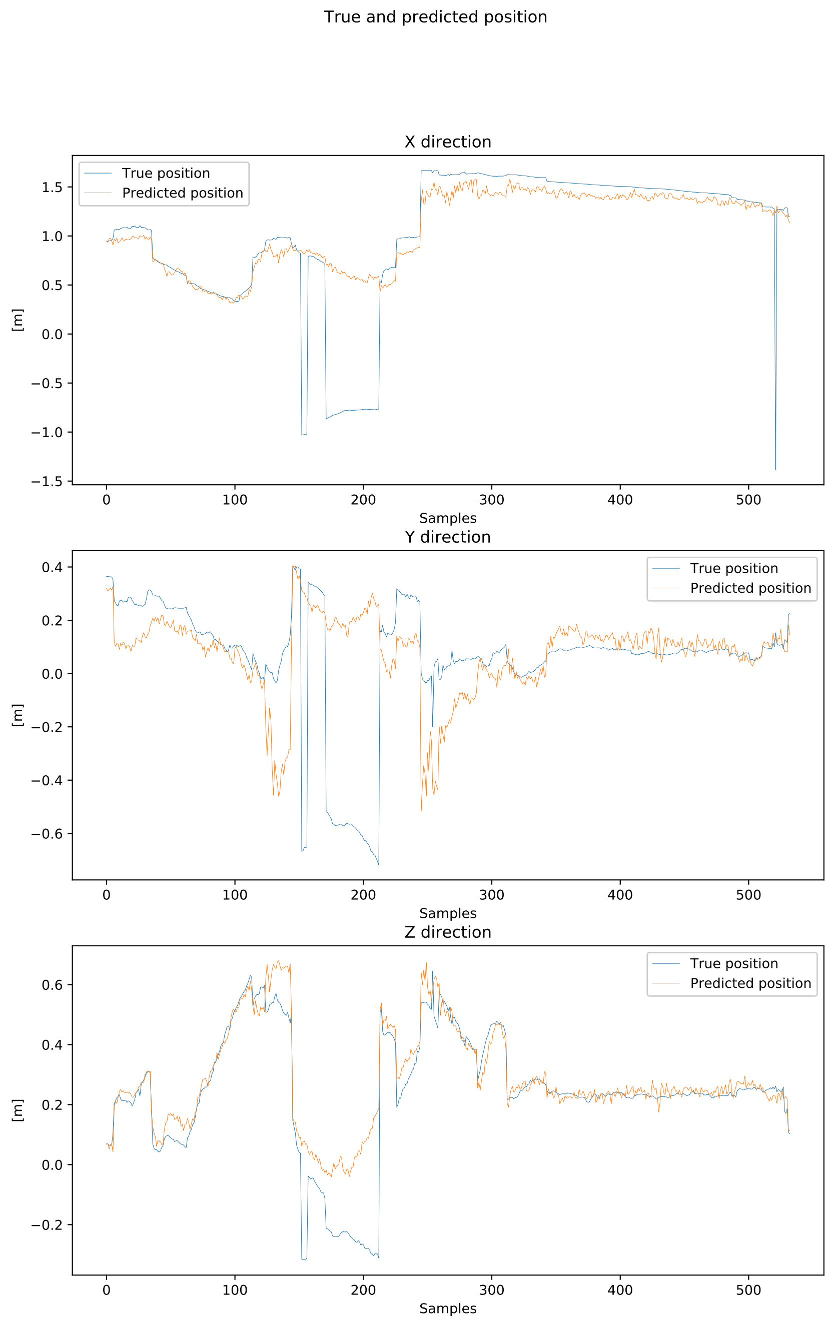


Figure D.12: Case 4: True and predicted position in surge, sway and heave

D.5 Case 5

Data ID	**Description**	**# Train images**	**# Test images**	**Splitting**
2, 3, 4	AruCo marker on metal stick	10 500		Temporal
4	AruCo marker on metal stick, reduced lights B		812	Temporal

Table D.15: Case 5: Details of train and test datasets

Measurement	**Median**	**Mean**	**Standard Deviation**
Euclidean translation error [mm]	75.52	132.97	216.15
Angle error [deg]	2.024	6.300	21.021

Table D.16: Case 5: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

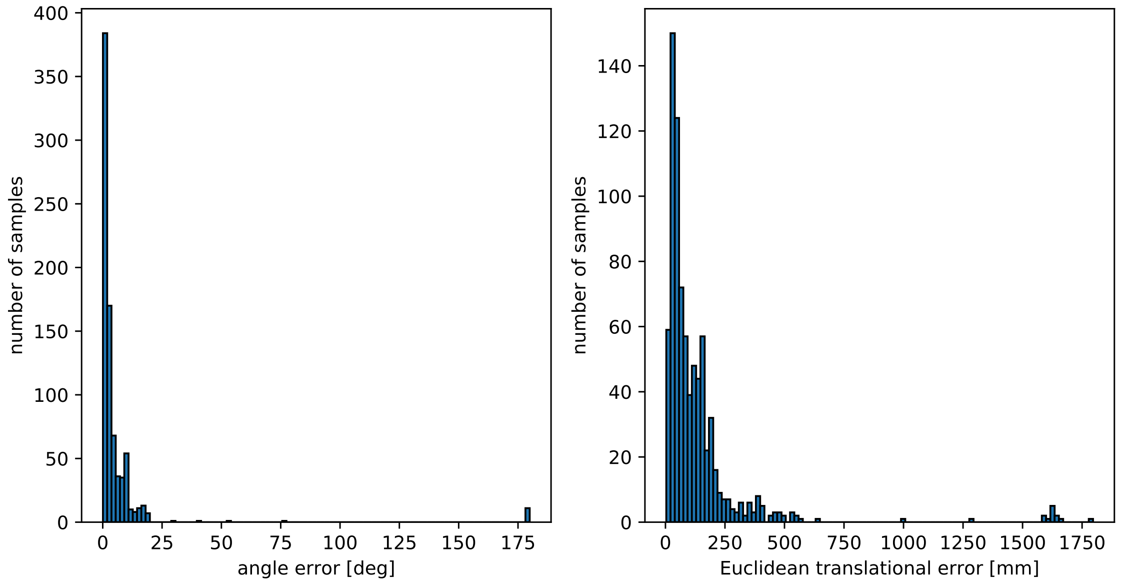


Figure D.13: Case 5: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	12.90	129.32
Y [mm]	13.61	197.29
Z [mm]	-16.37	90.19
Roll [deg]	-0.985	7.822
Pitch [deg]	0.613	3.959
Yaw [deg]	2.188	20.468

Table D.17: Case 5: Summary statistics of pose errors in each DoF

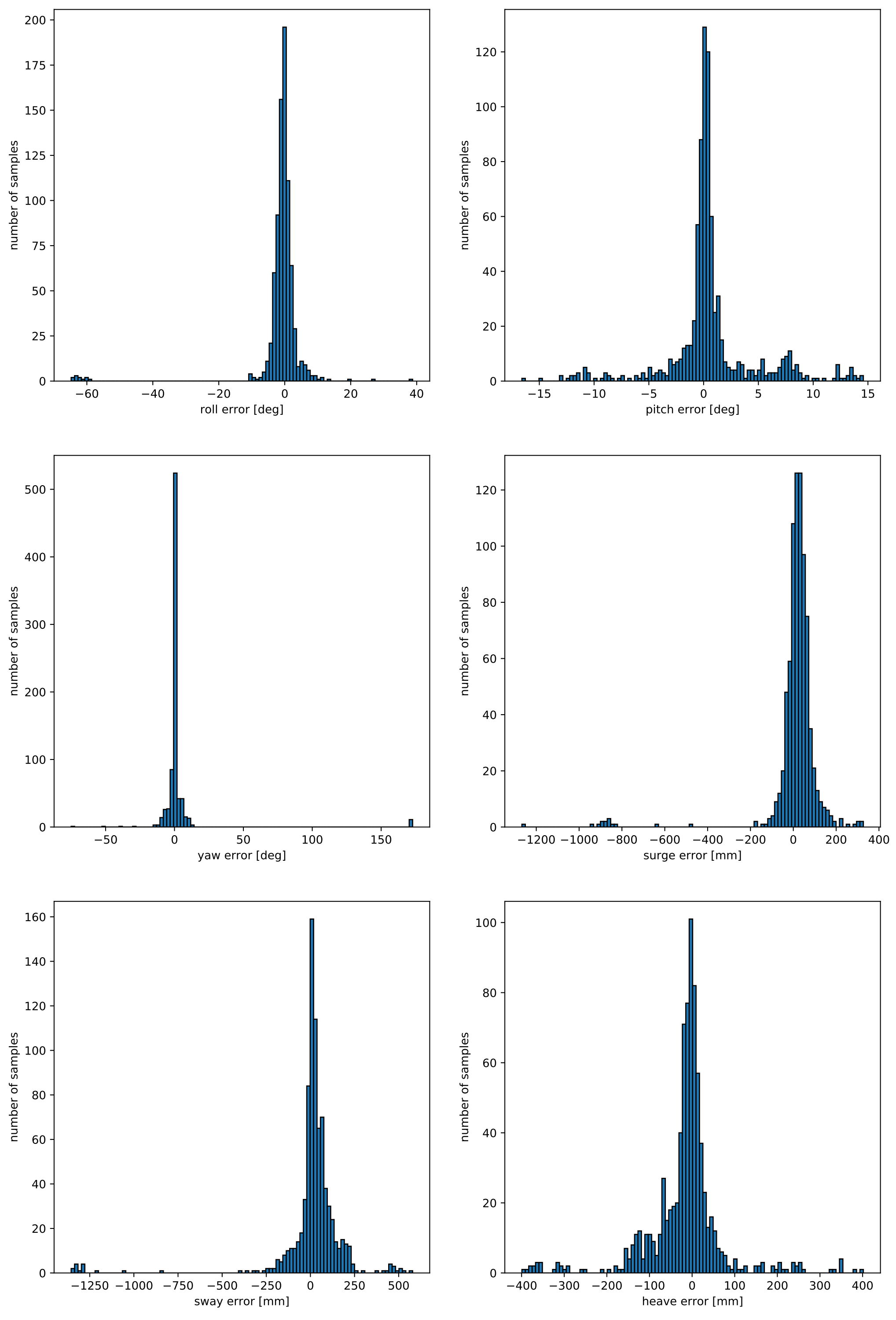


Figure D.14: Case 5: Pose errors in each DoF

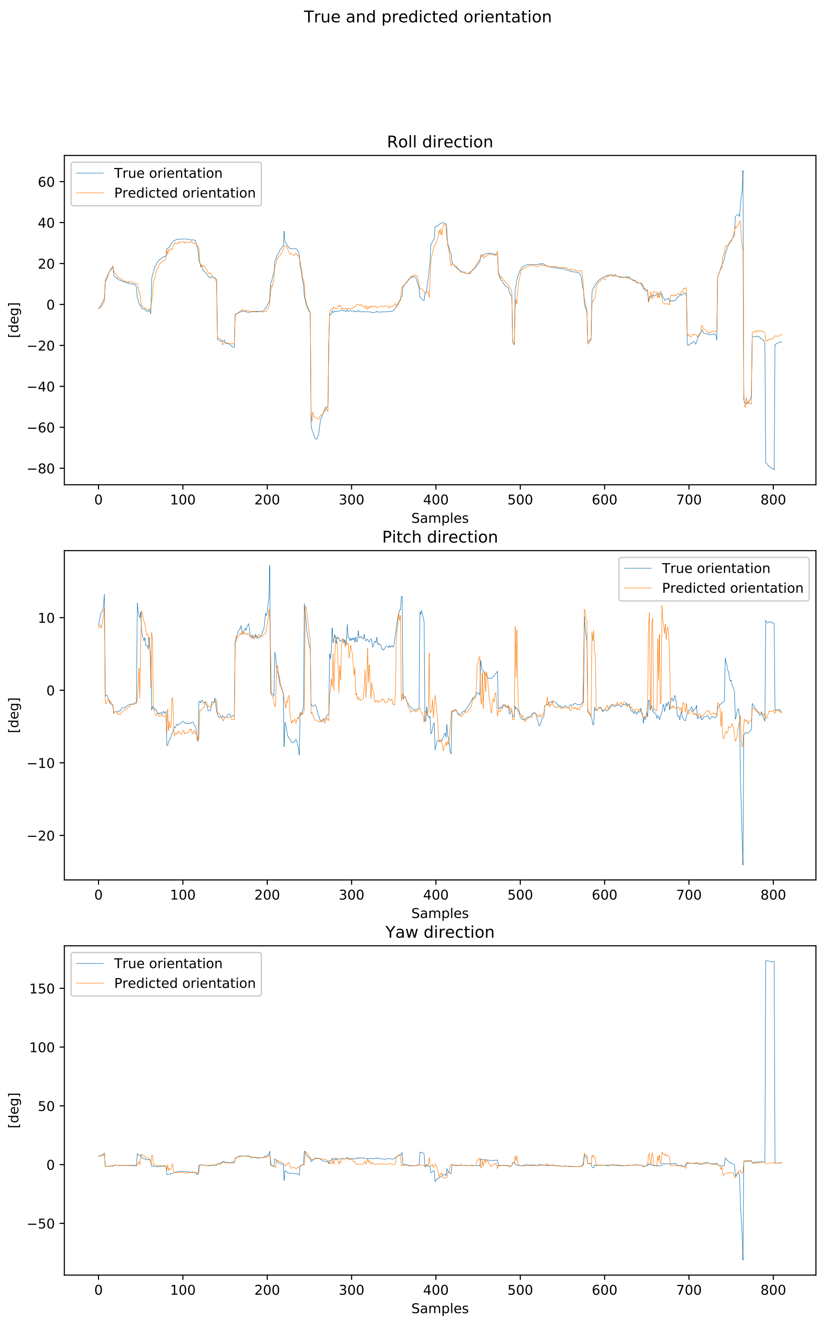


Figure D.15: Case 5: True and predicted orientation in roll, pitch and yaw

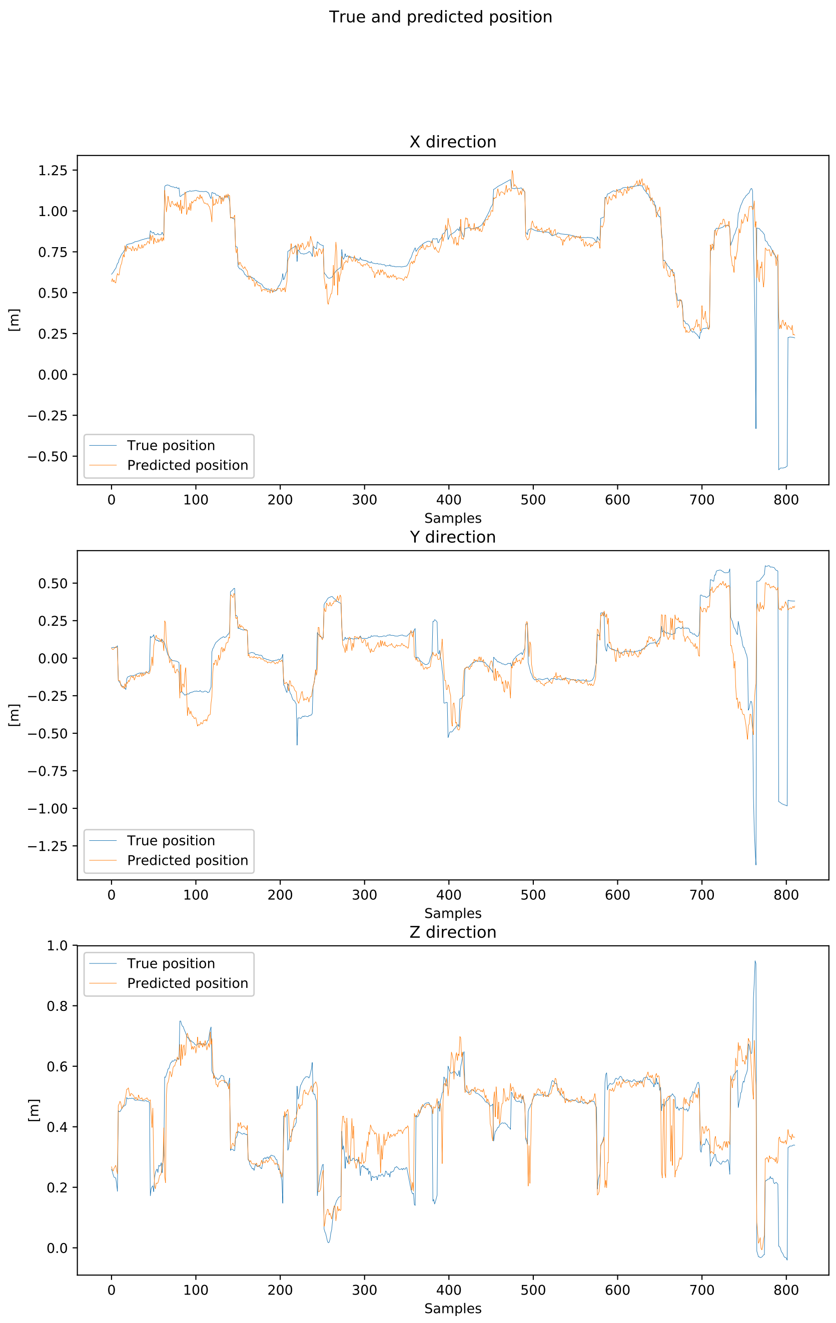


Figure D.16: Case 5: True and predicted position in surge, sway and heave

D.6 Case 6

Data ID	Description	# Train images	# Test images	Splitting
6	Plastic ring on metal box	6 500	2 093	Random

Table D.18: Case 6: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	21.46	22.54	8.98
Angle error [deg]	0.503	0.670	0.587

Table D.19: Case 6: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

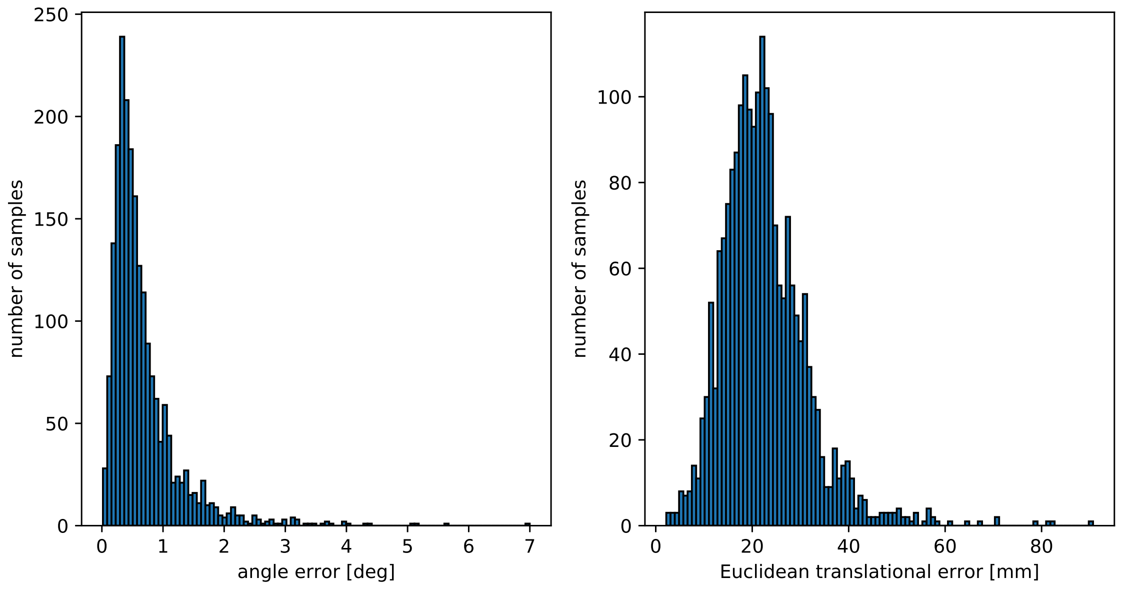


Figure D.17: Case 6: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-4.62	12.51
Y [mm]	5.58	10.41
Z [mm]	13.86	8.91
Roll [deg]	0.097	0.714
Pitch [deg]	-0.155	0.396
Yaw [deg]	-0.016	0.339

Table D.20: Case 6: Summary statistics of pose errors in each DoF

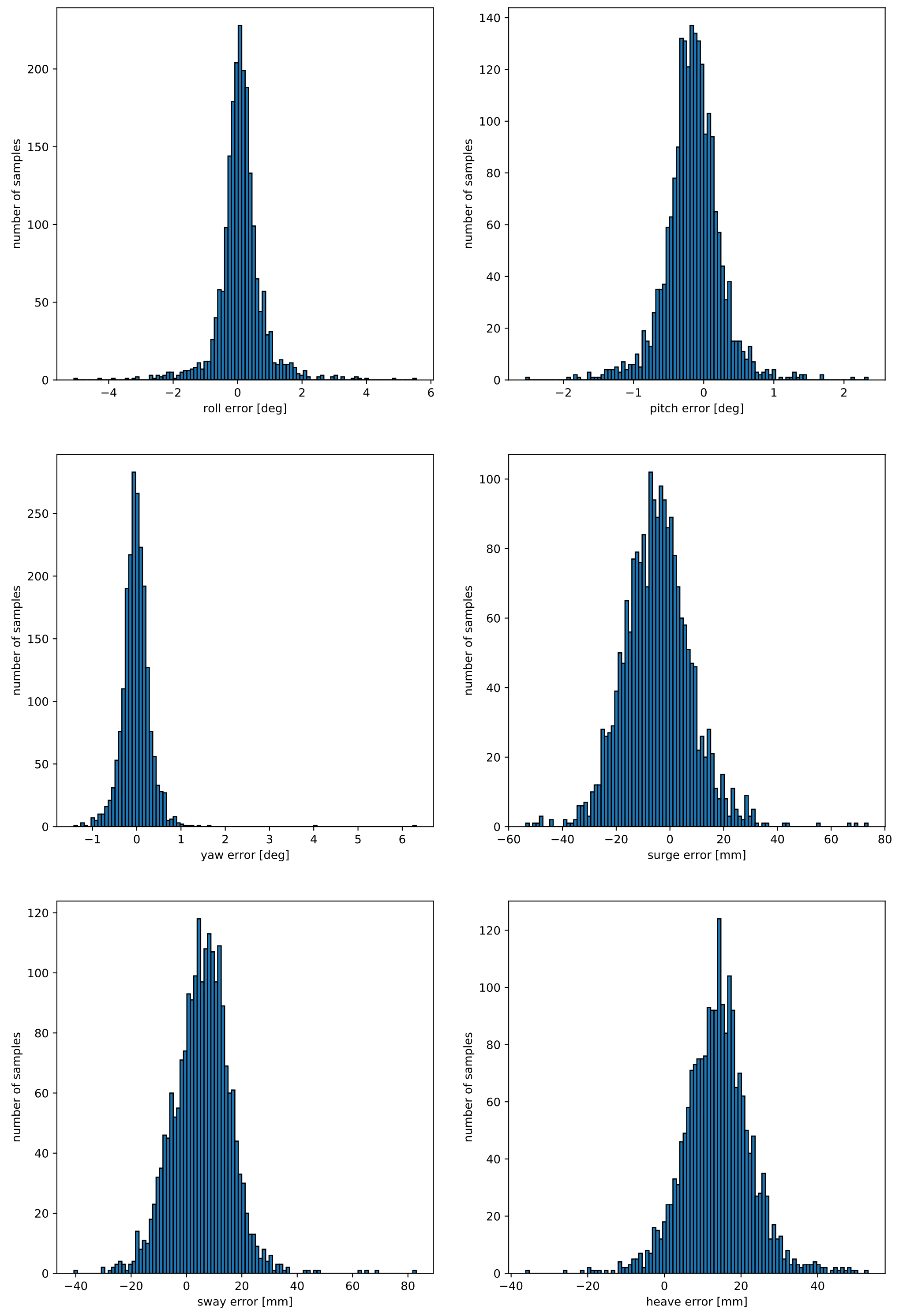


Figure D.18: Case 6: Pose errors in each DoF

D.7 Case 7

Data ID	Description	# Train images	# Test images	Splitting
1	AruCo marker on metal box	8 000	3 000	Temporal

Table D.21: Case 7: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	205.18	236.19	153.16
Angle error [deg]	3.517	5.513	5.498

Table D.22: Case 7: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

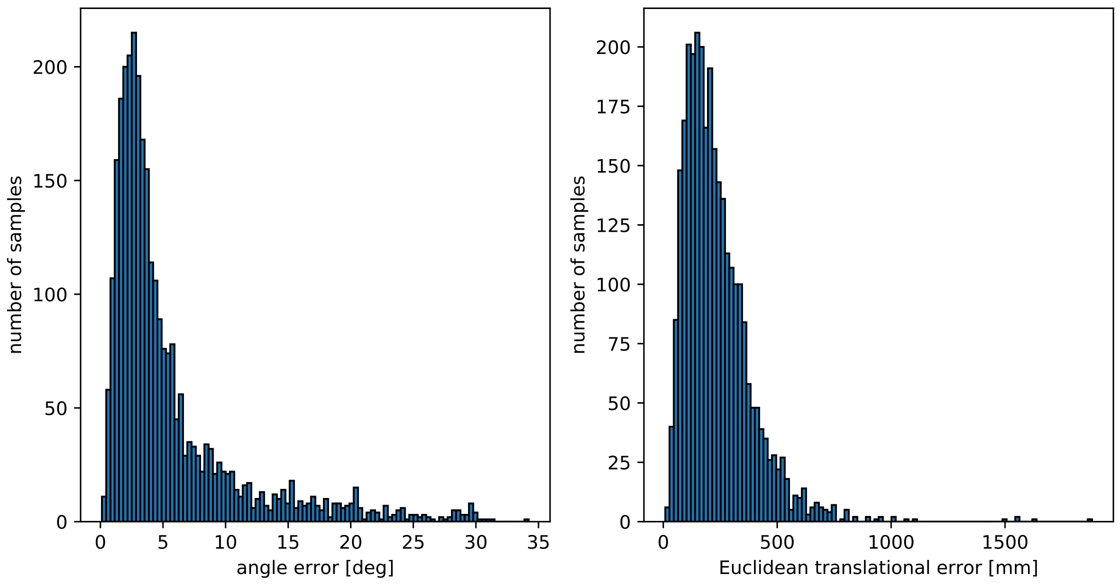


Figure D.19: Case 7: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	11.49	167.77
Y [mm]	- 36.44	160.25
Z [mm]	22.86	153.08
Roll [deg]	-0.110	1.728
Pitch [deg]	0.178	1.604
Yaw [deg]	-0.111	1.481

Table D.23: Case 7: Summary statistics of pose errors in each DoF

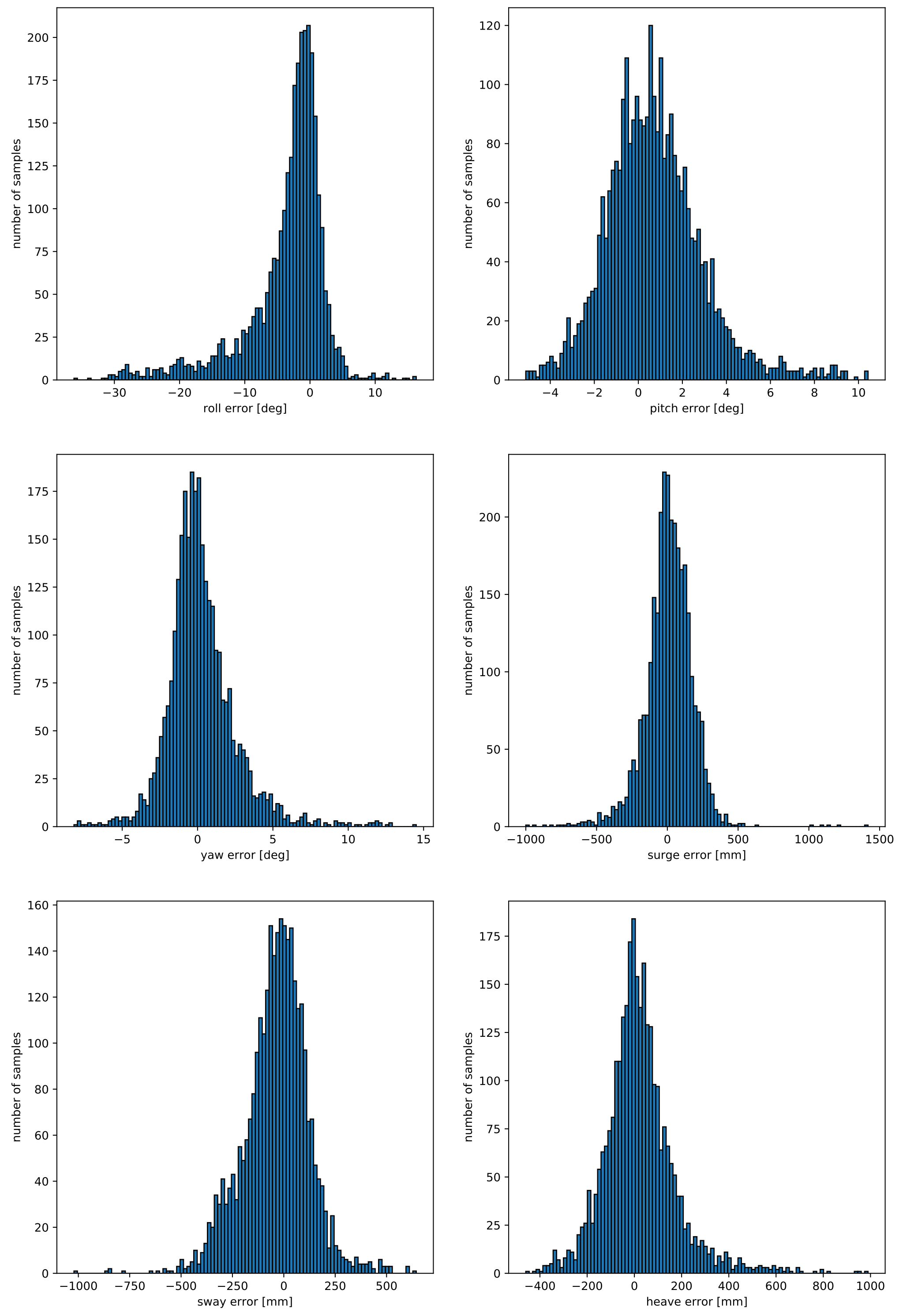


Figure D.20: Case 7: Pose errors in each DoF

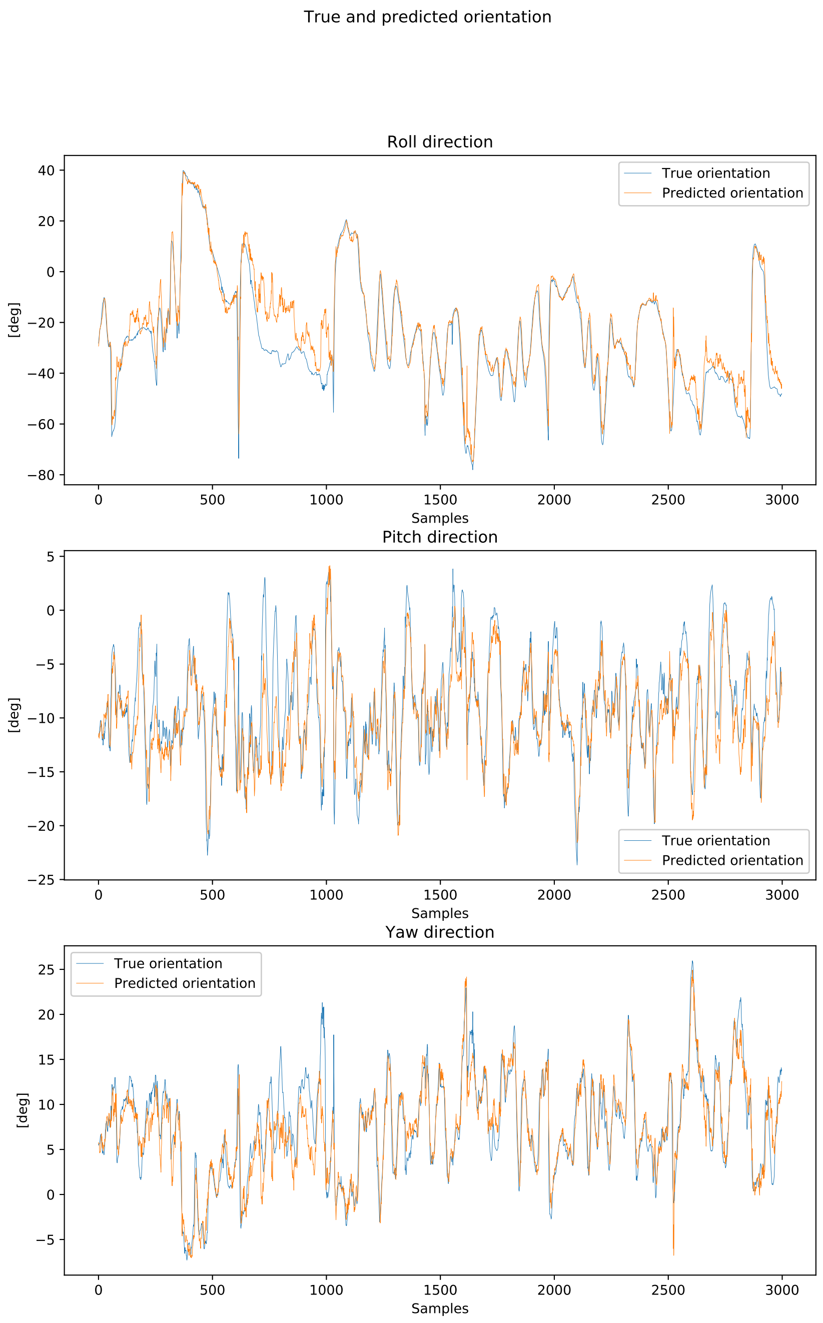


Figure D.21: Case 7: True and predicted orientation in roll, pitch and yaw

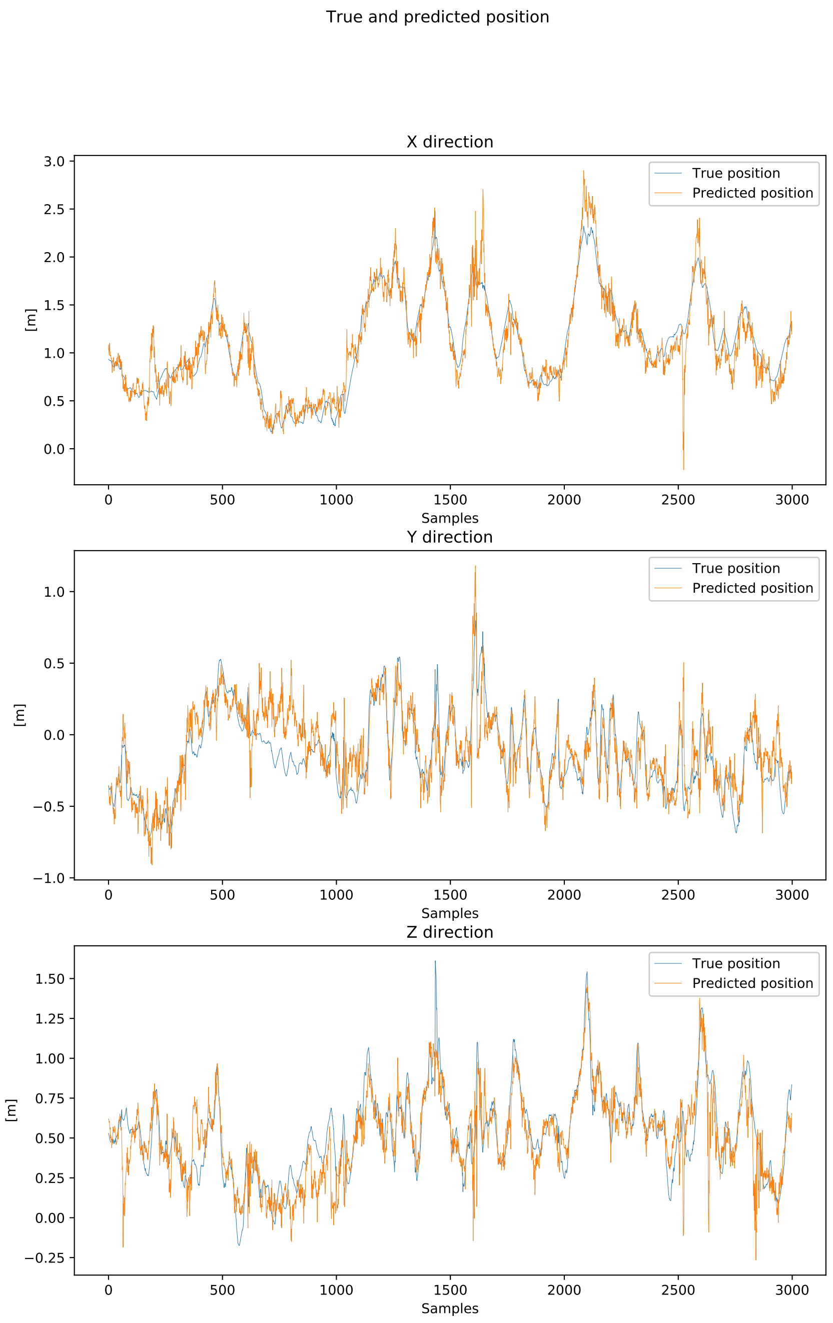


Figure D.22: Case 7: True and predicted position in surge, sway and heave

D.8 Case 8

Data ID	Description	# Train images	# Test images	Splitting
2,4	AruCo marker on metal stick	10 500	2 798	Temporal

Table D.24: Case 8: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	77.85	181.39	269.41
Angle error [deg]	2.376	8.825	18.668

Table D.25: Case 8: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

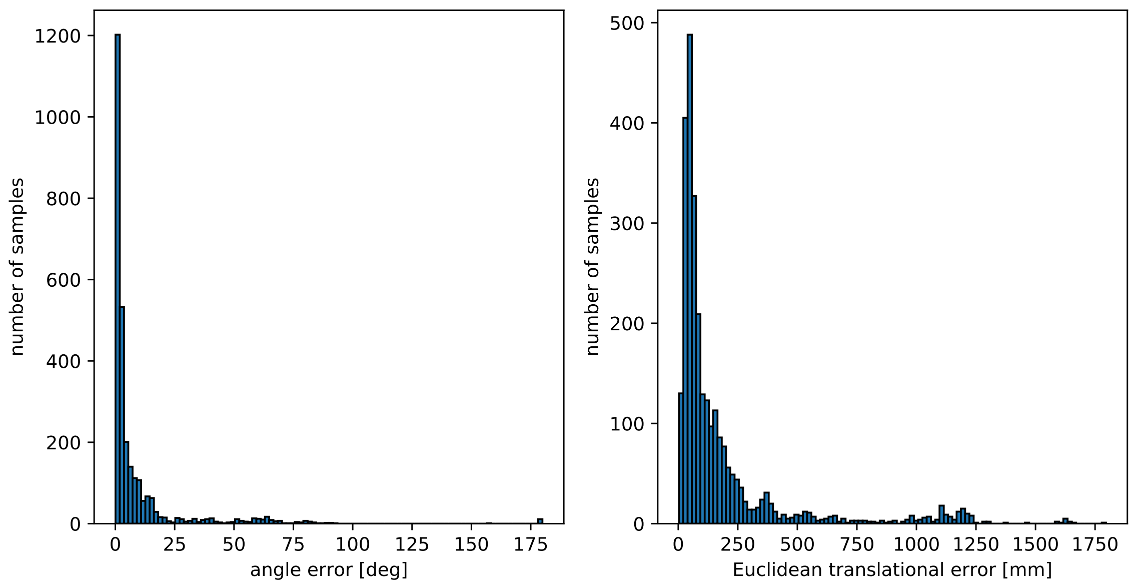


Figure D.23: Case 8: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	21.82	209.04
Y [mm]	2.91	228.86
Z [mm]	-25.97	90.83
Roll [deg]	-0.707	10.859
Pitch [deg]	1.033	6.675
Yaw [deg]	1.985	18.658

Table D.26: Case 8: Summary statistics of pose errors in each DoF

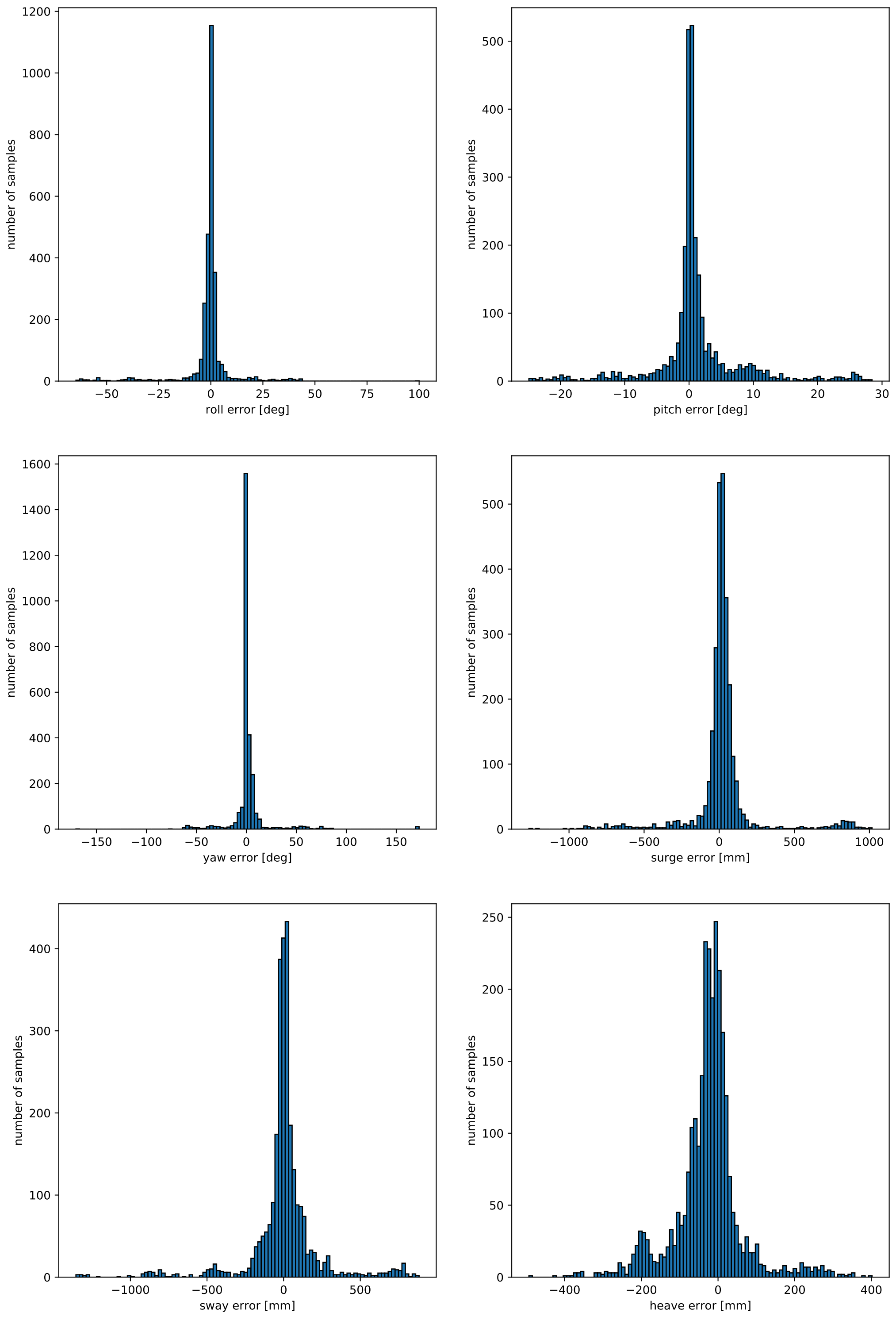


Figure D.24: Case 8: Pose errors in each DoF

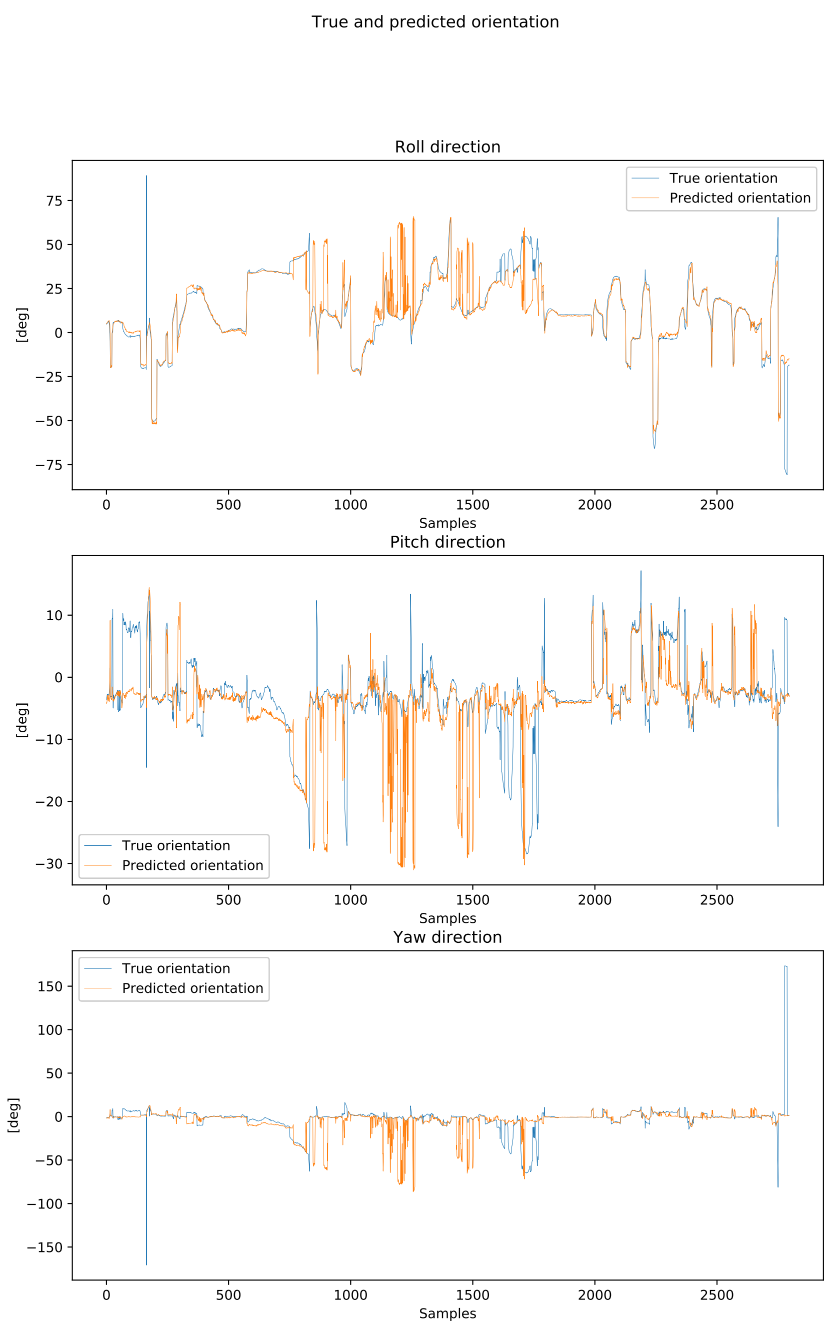


Figure D.25: Case 8: True and predicted orientation in roll, pitch and yaw

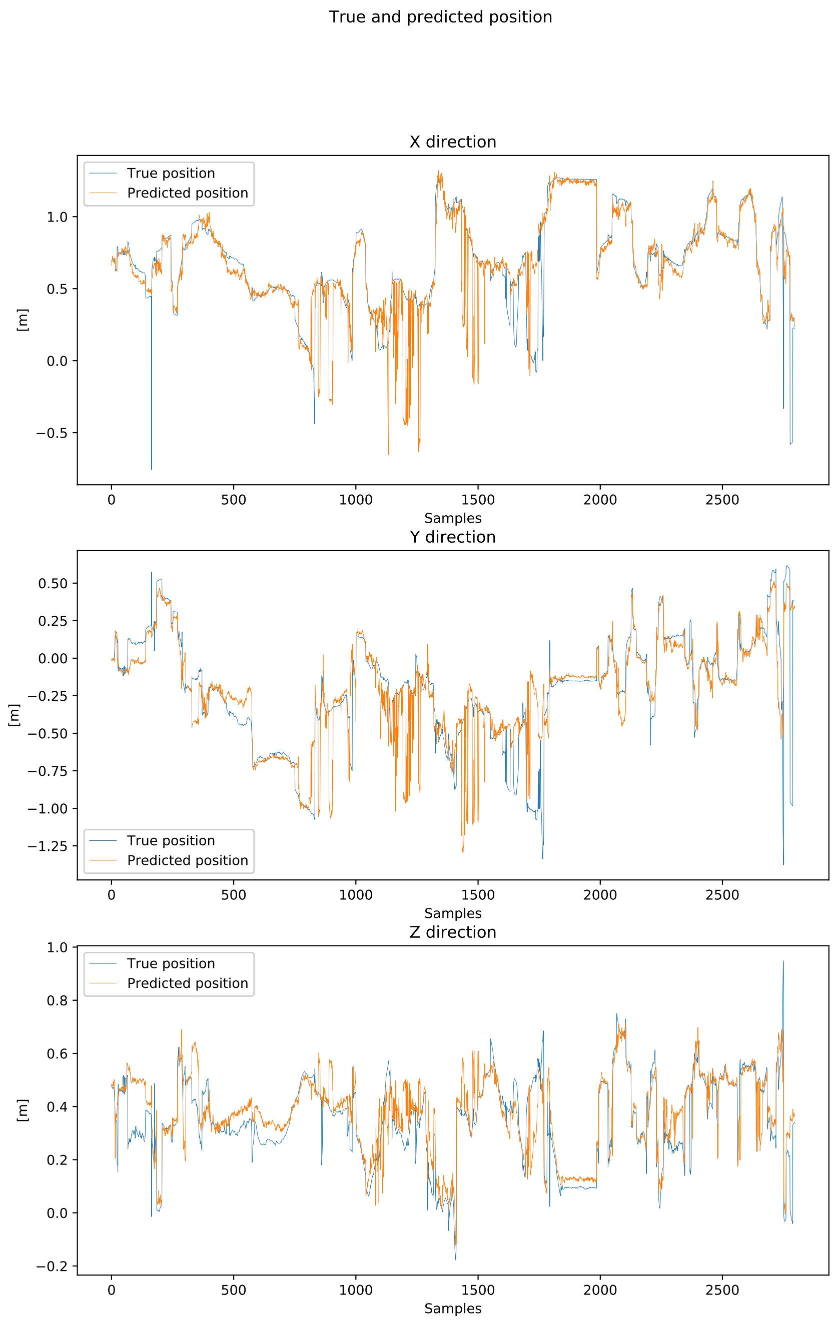


Figure D.26: Case 8: True and predicted position in surge, sway and heave

D.9 Case 9

Data ID	**Description**	**# Train images**	**# Test images**	**Splitting**
6	Plastic ring on metal box	6 500	2 093	Temporal

Table D.27: Case 9: Details of train and test datasets

Measurement	**Median**	**Mean**	**Standard Deviation**
Euclidean translation error [mm]	50.88	63.15	47.66
Angle error [deg]	1.316	1.757	1.418

Table D.28: Case 9: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

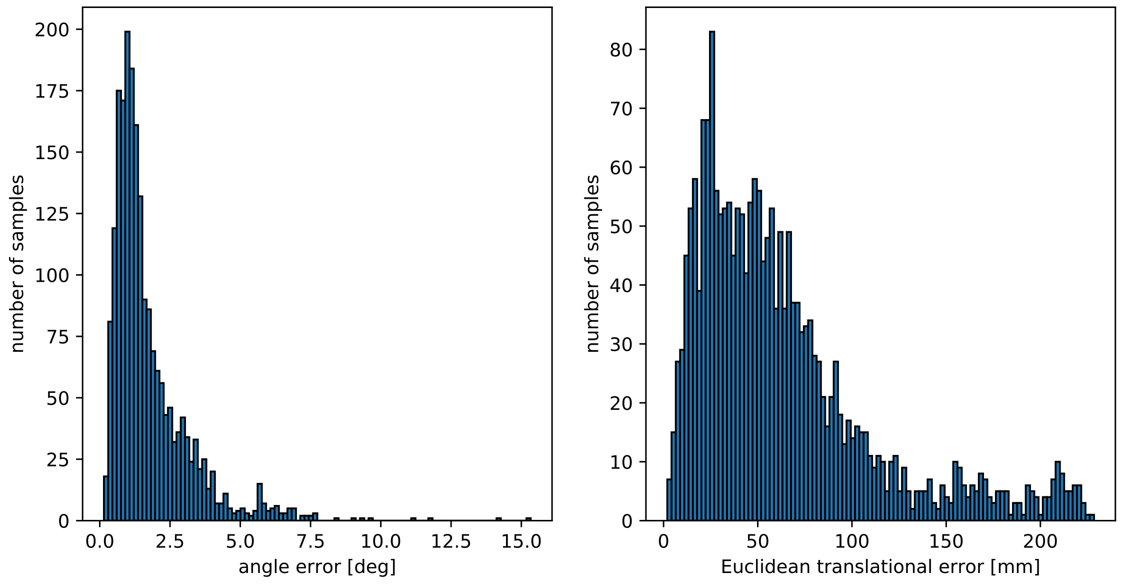


Figure D.27: Case 9: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-22.47	38.52
Y [mm]	-29.70	38.30
Z [mm]	-3.31	43.70
Roll [deg]	0.879	1.762
Pitch [deg]	0.261	0.788
Yaw [deg]	0.019	0.654

Table D.29: Case 9: Summary statistics of pose errors in each DoF

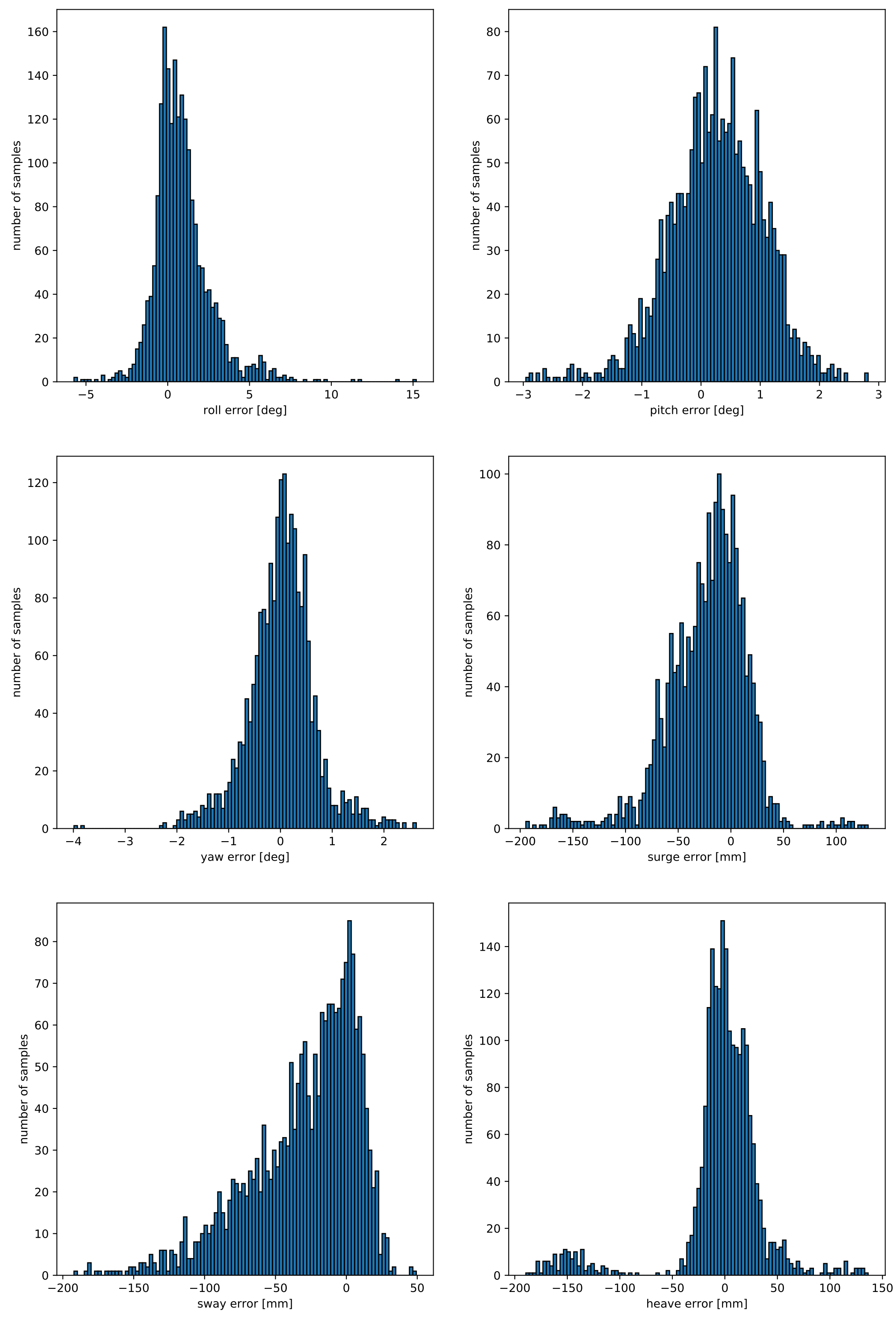


Figure D.28: Case 9: Pose errors in each DoF

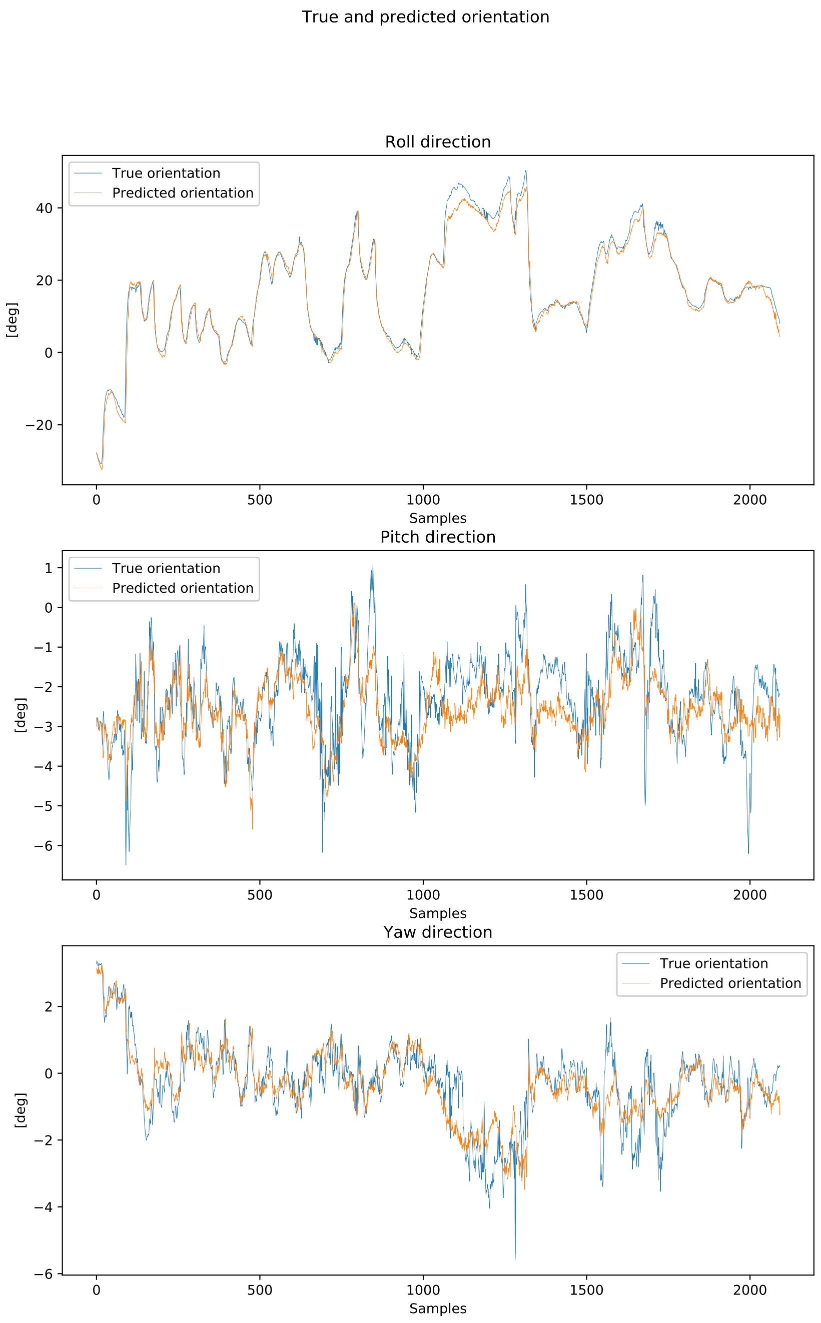


Figure D.29: Case 9: True and predicted orientation in roll, pitch and yaw

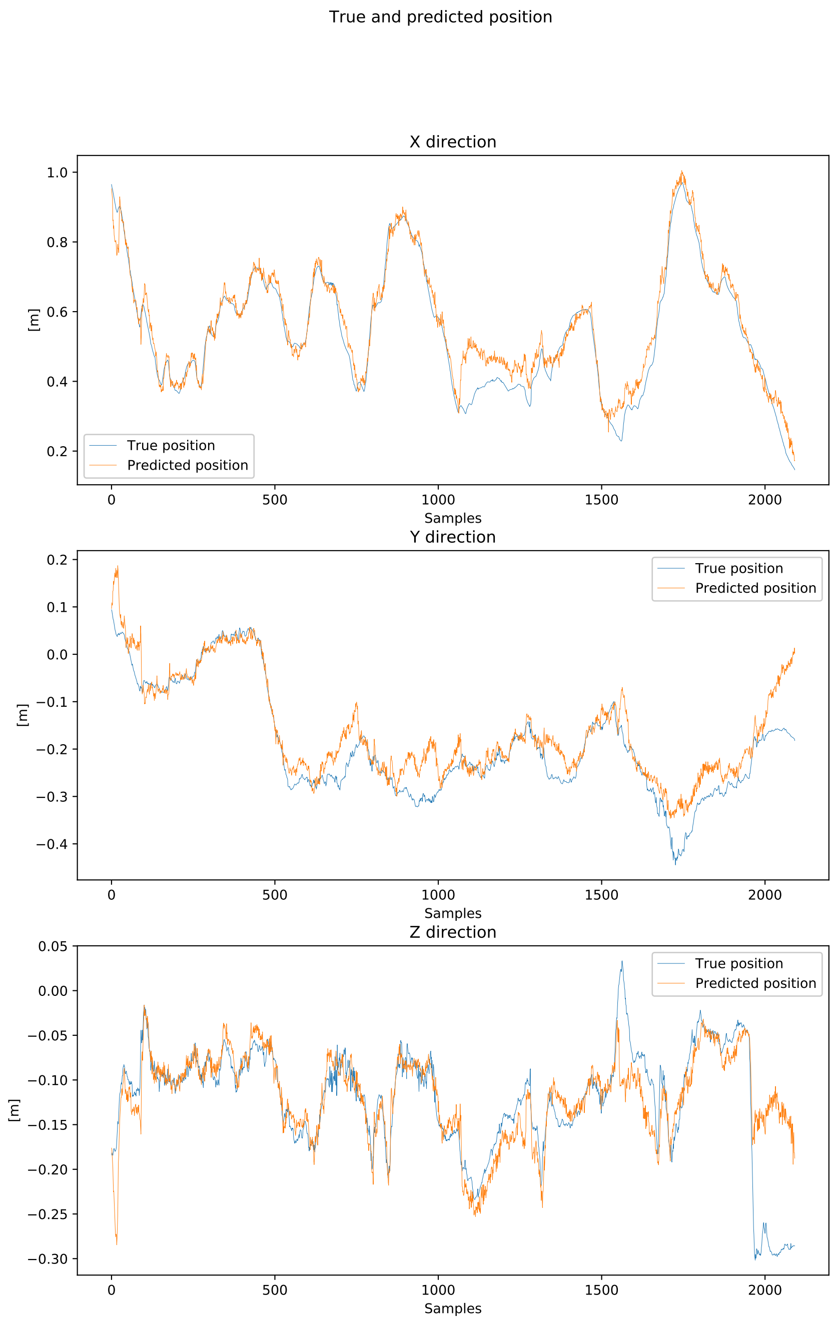


Figure D.30: Case 9: True and predicted position in surge, sway and heave

D.10 Case 10

Data ID	Description	# Train images	# Test images	Splitting
1,2,4	AruCo on metal box and stick	18 000		Temporal
1	AruCo on metal box		3 000	Temporal

Table D.30: Case 10: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	232.98	269.62	156.54
Angle error [deg]	3.910	6.377	6.889

Table D.31: Case 10: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

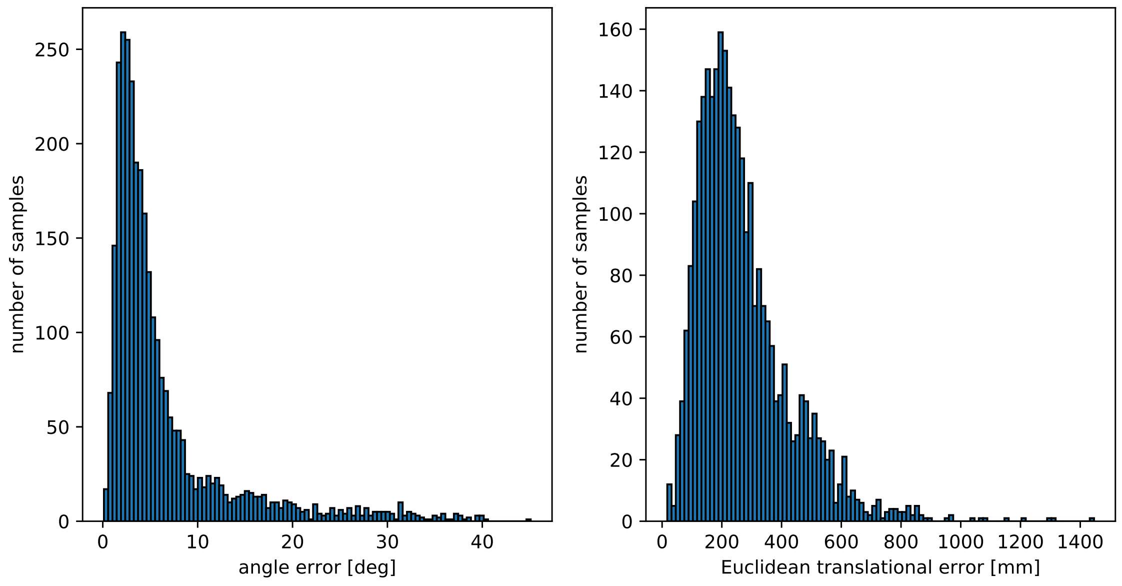


Figure D.31: Case 10: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-12.77	169.72
Y [mm]	-37.88	182.75
Z [mm]	43.18	177.77
Roll [deg]	-3.941	7.961
Pitch [deg]	0.759	2.618
Yaw [deg]	0.788	2.707

Table D.32: Case 10: Summary statistics of pose errors in each DoF

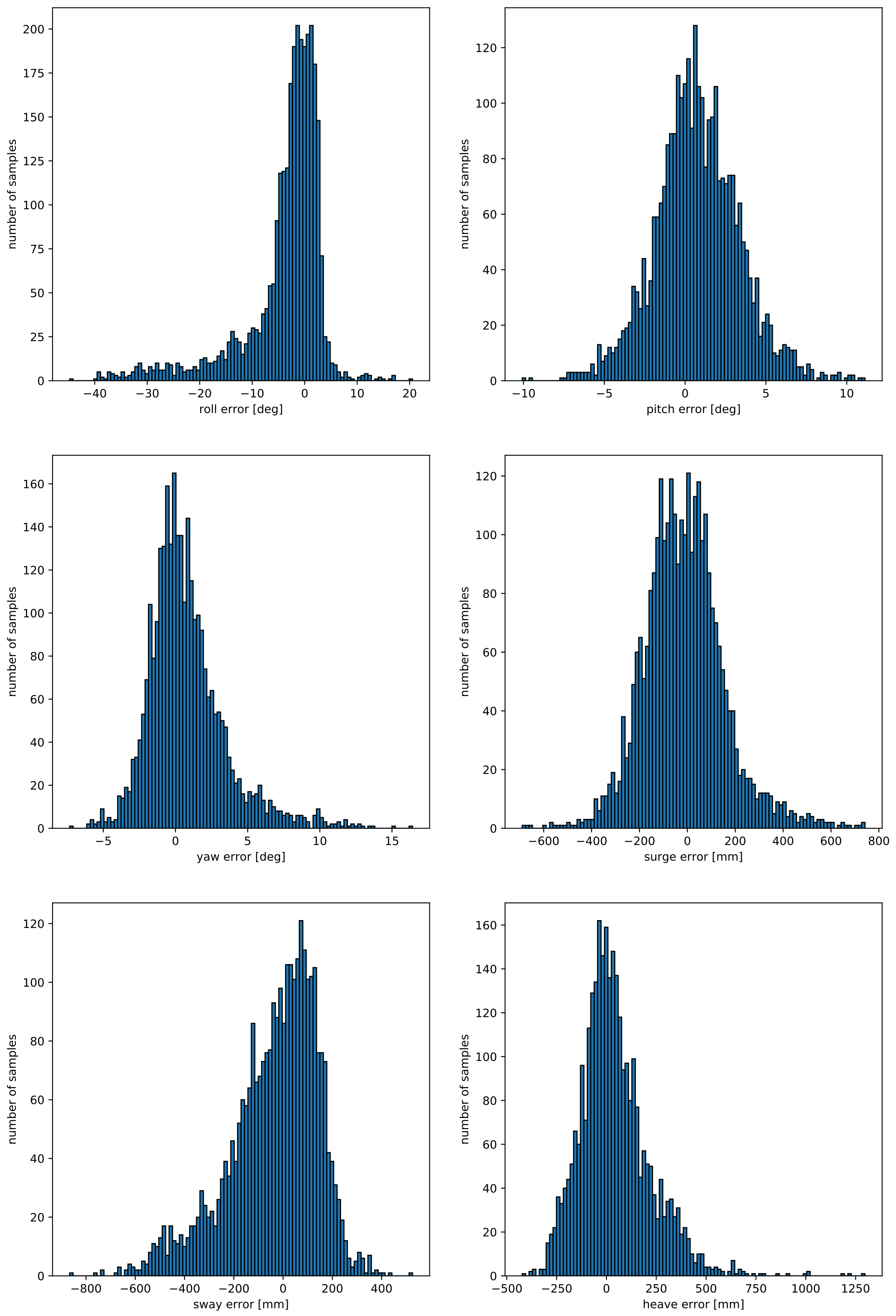


Figure D.32: Case 10: Pose errors in each DoF

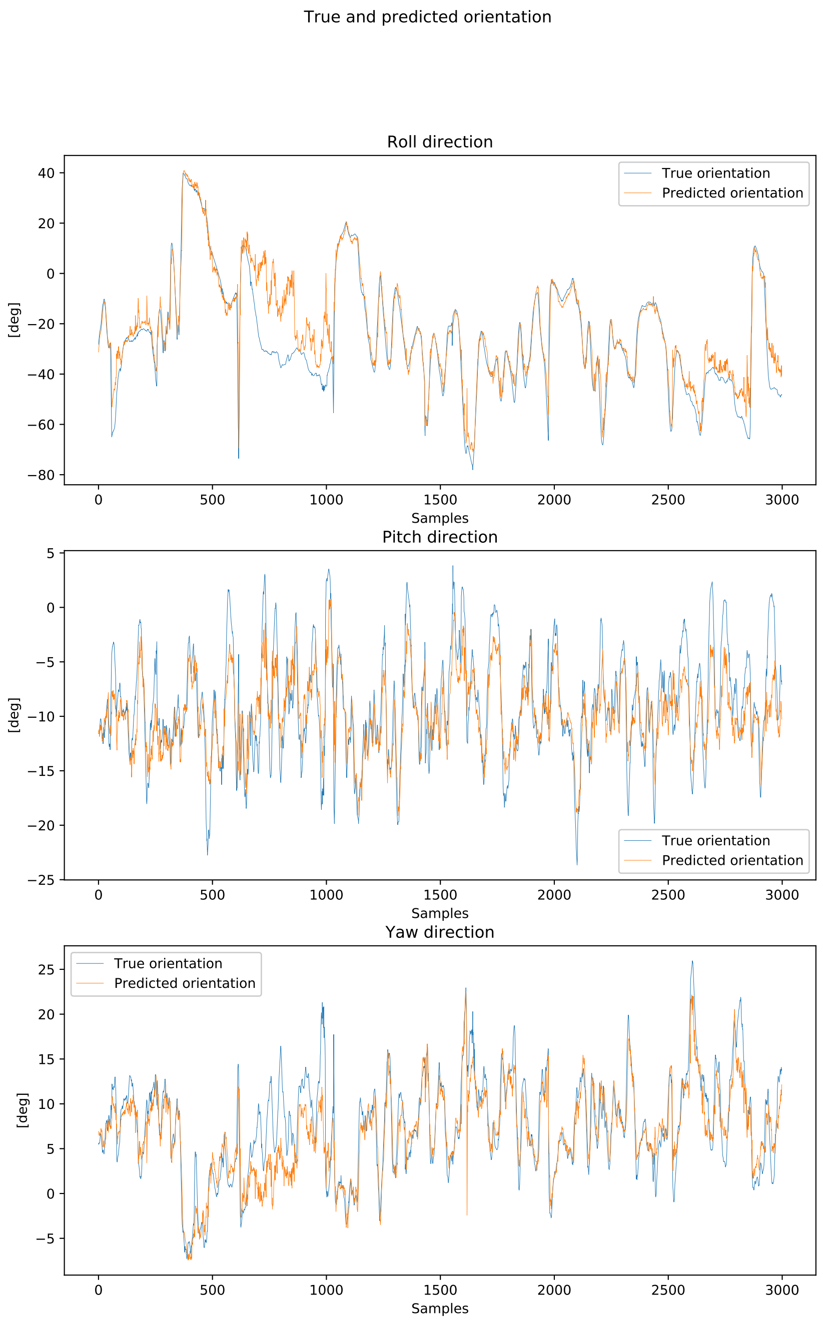


Figure D.33: Case 10: True and predicted orientation in roll, pitch and yaw

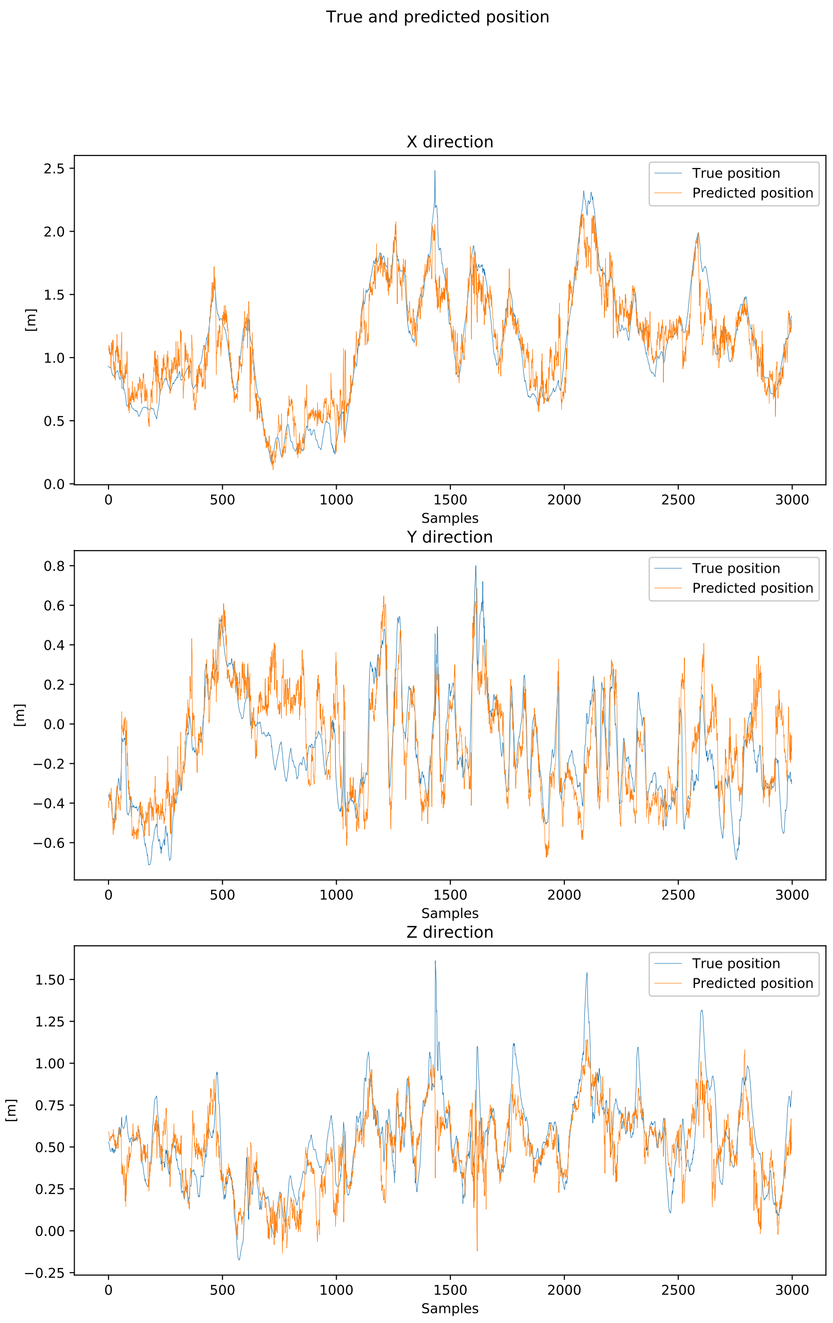


Figure D.34: Case 10: True and predicted position in surge, sway and heave

D.11 Case 11

Data ID	Description	# Train images	# Test images	Splitting
1,2,4	AruCo on metal box and stick	18 000		Temporal
2,4	AruCo on stick		2 798	Temporal

Table D.33: Case 11: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	92.27	168.63	227.13
Angle error [deg]	1.955	6.830	15.796

Table D.34: Case 11: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

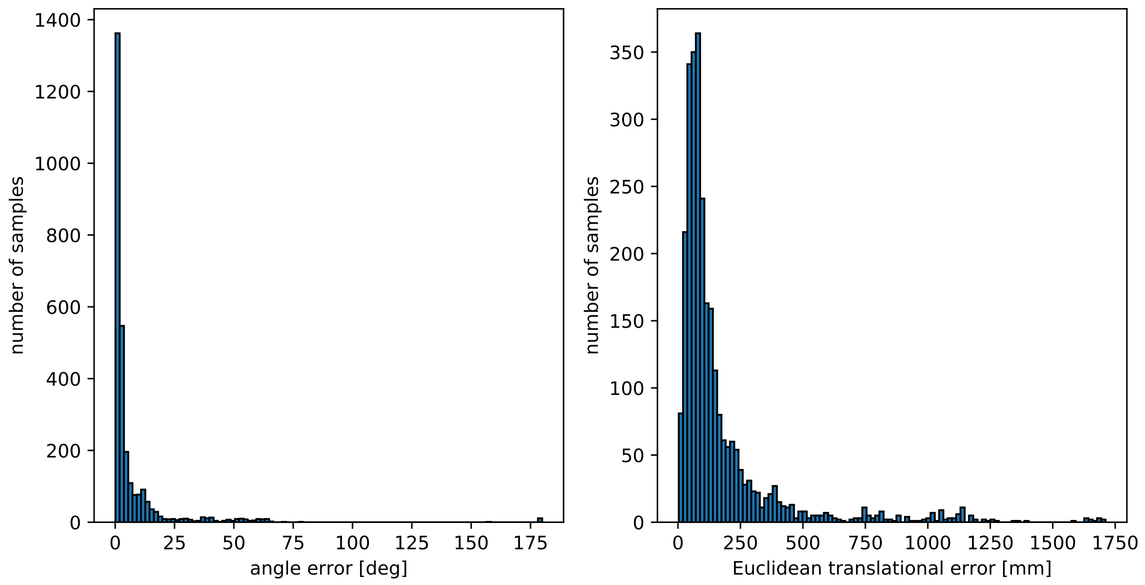


Figure D.35: Case 11: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-27.72	175.30
Y [mm]	-18.39	199.25
Z [mm]	-16.92	90.56
Roll [deg]	0.084	8.201
Pitch [deg]	0.057	5.534
Yaw [deg]	0.102	15.706

Table D.35: Case 11: Summary statistics of pose errors in each DoF

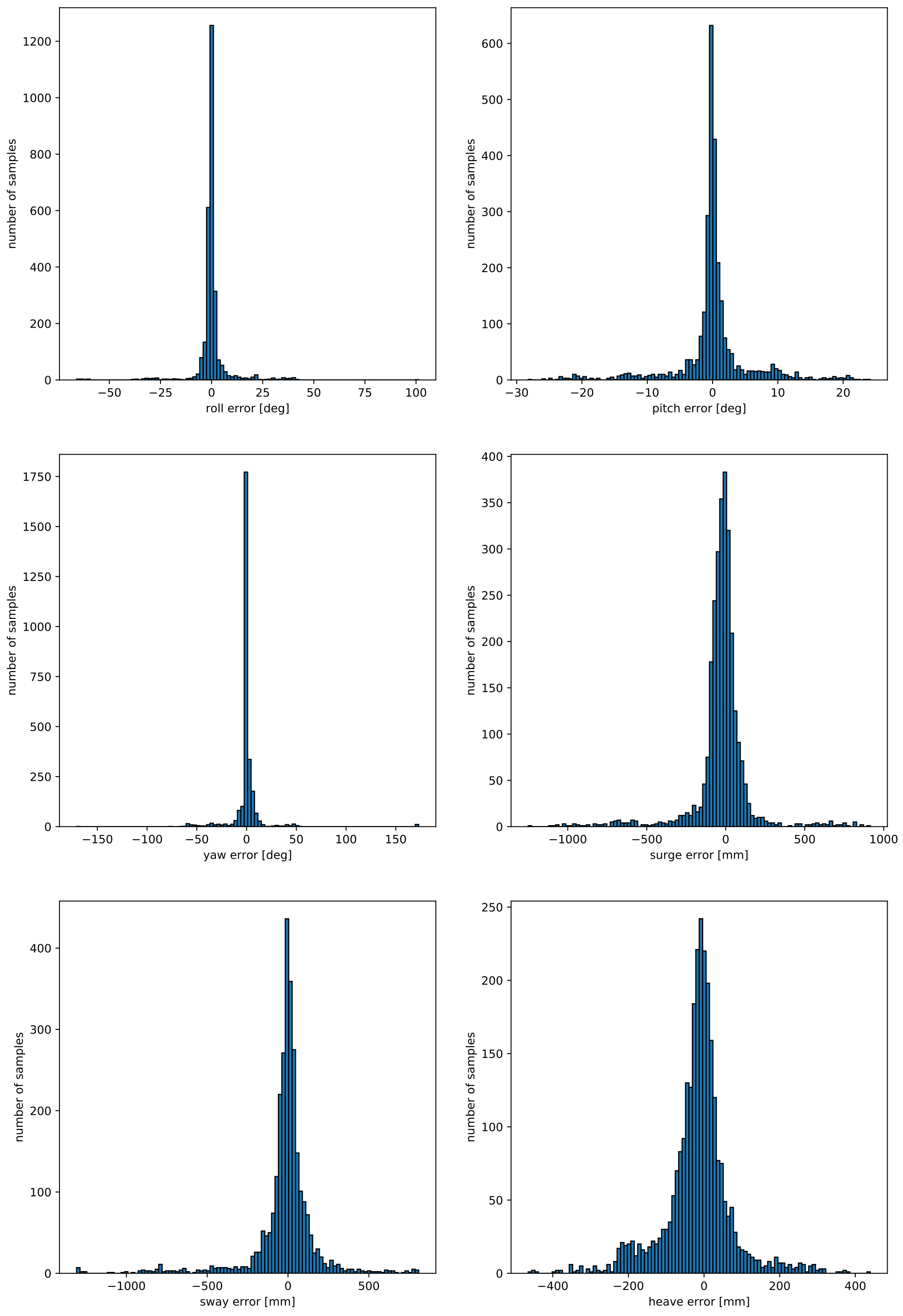


Figure D.36: Case 11: Pose errors in each DoF

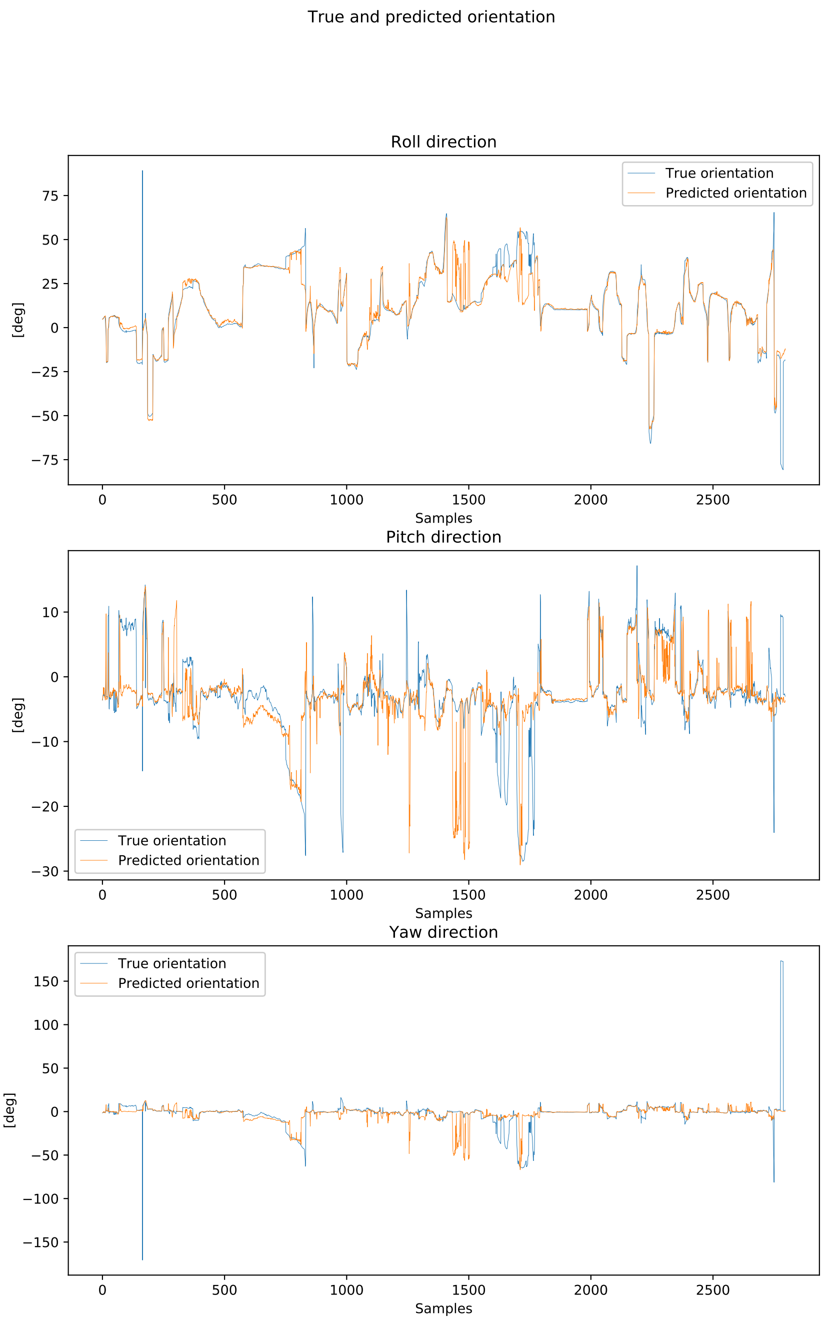


Figure D.37: Case 11: True and predicted orientation in roll, pitch and yaw

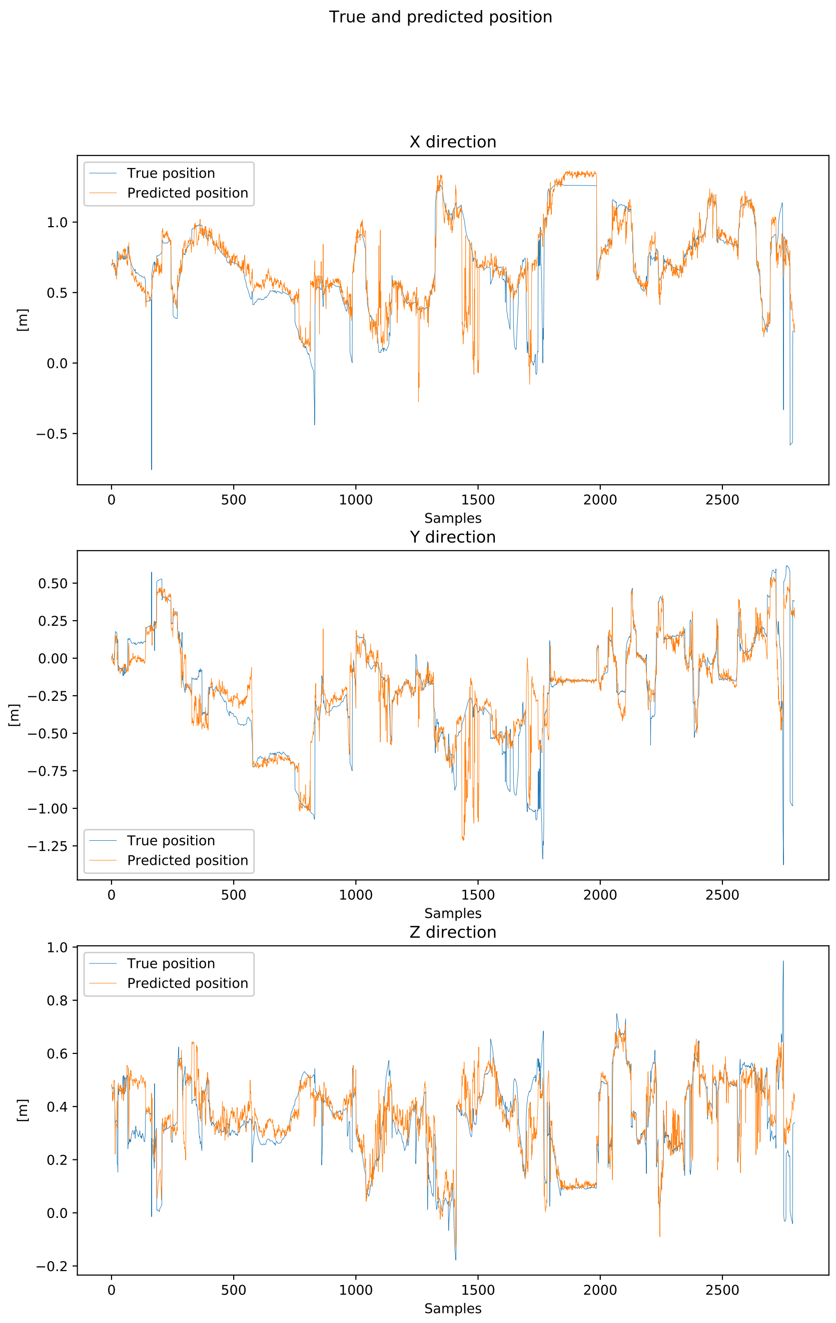


Figure D.38: Case 11: True and predicted position in surge, sway and heave

D.12 Case 12

Data ID	Description	Train set size	Test set size	Splitting
5,6	Ring on stick + ring on box	17 500		Temporal
6	Ring on box		2 093	Temporal

Table D.36: Case 12: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	41.54	54.07	46.03
Angle error [deg]	1.182	1.537	1.323

Table D.37: Case 12: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

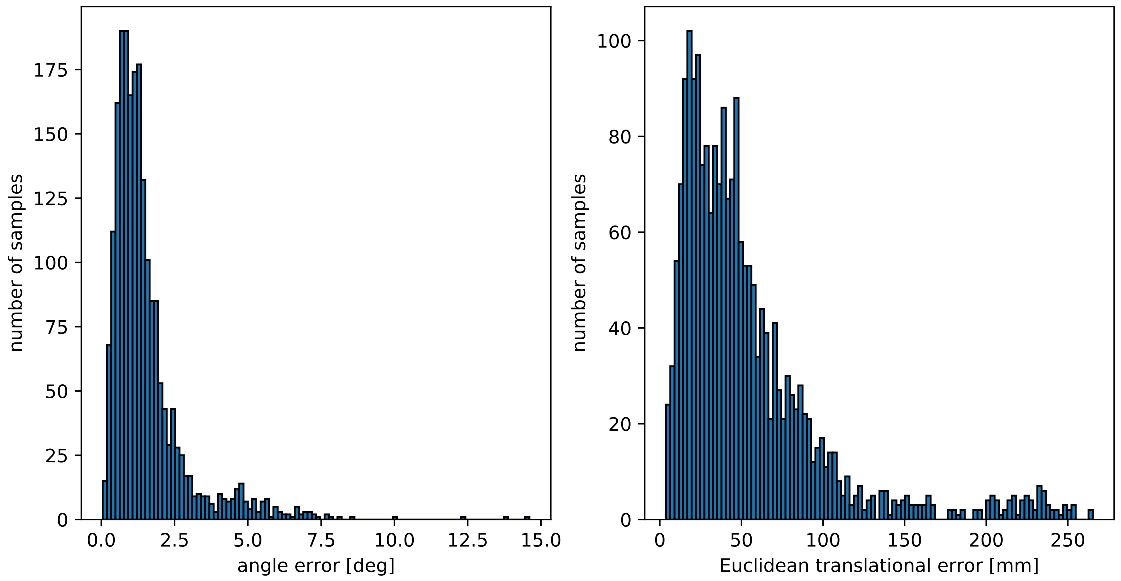


Figure D.39: Case 12: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-1.54	31.04
Y [mm]	-21.02	36.51
Z [mm]	-14.38	45.77
Roll [deg]	0.524	1.718
Pitch [deg]	-0.024	0.745
Yaw [deg]	-0.079	0.592

Table D.38: Case 12: Summary statistics of pose errors in each DoF

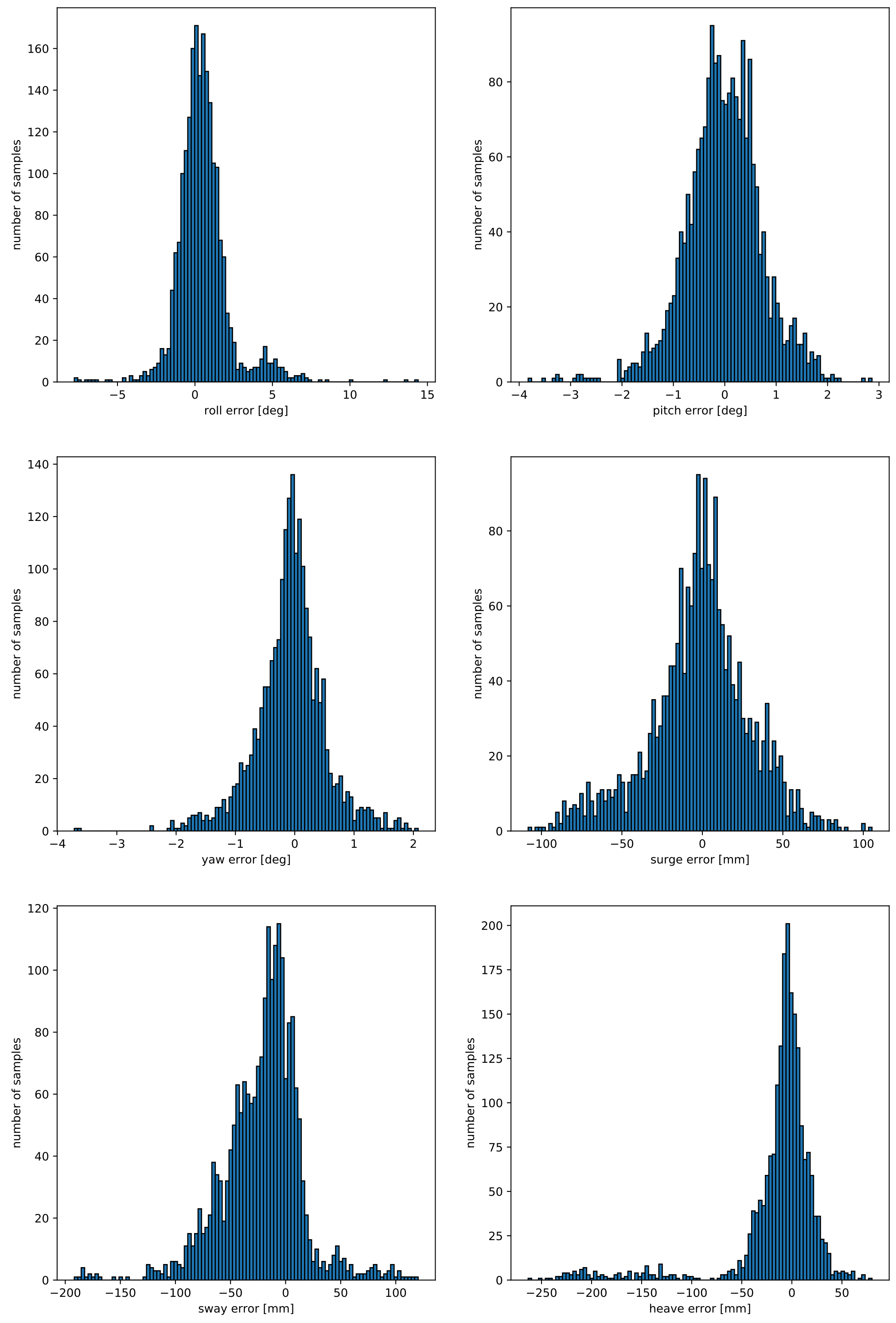


Figure D.40: Case 12: Pose errors in each DoF

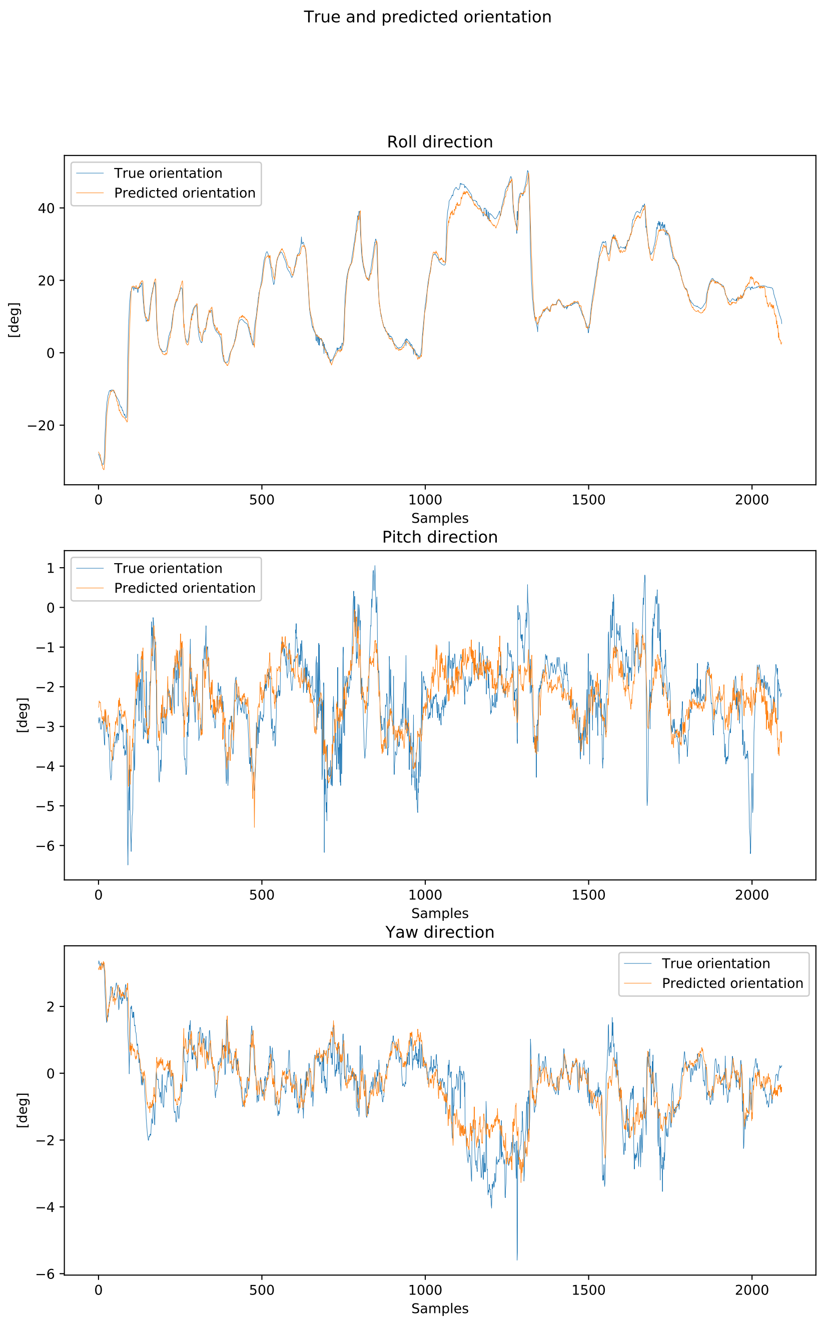


Figure D.41: Case 12: True and predicted orientation in roll, pitch and yaw

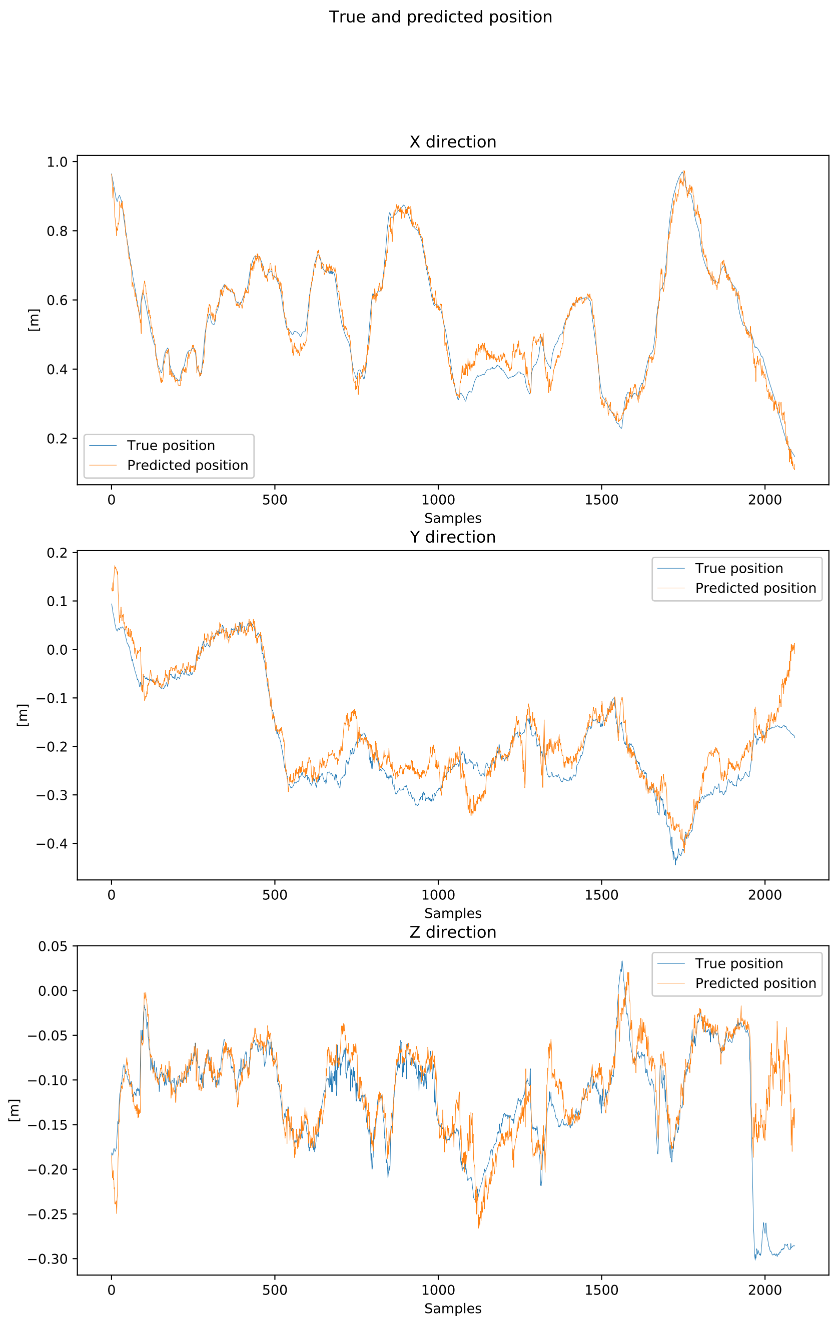


Figure D.42: Case 12: True and predicted position in surge, sway and heave

D.13 Case 13

Data ID	Description	# Train images	# Test images	Splitting
5	Ring on stick	11 000		Temporal
6	Ring on box		2 093	Temporal

Table D.39: Case 13: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	196.91	201.21	74.27
Angle error [deg]	4.962	5.112	2.449

Table D.40: Case 13: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

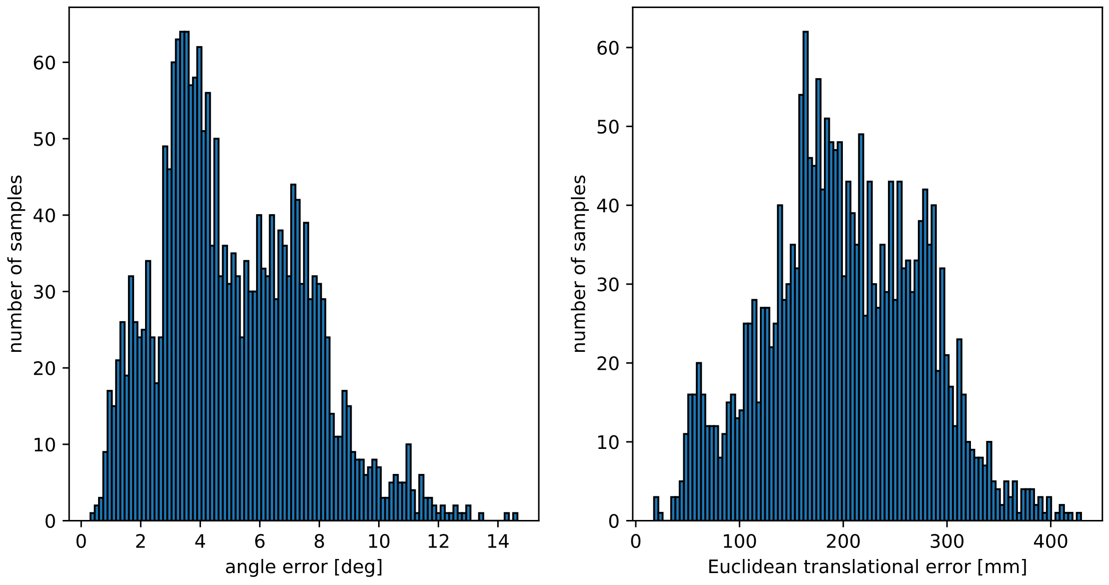


Figure D.43: Case 13: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	167.69	78.12
Y [mm]	-30.64	89.42
Z [mm]	8.92	52.58
Roll [deg]	-3.985	4.013
Pitch [deg]	0.010	1.139
Yaw [deg]	0.000	0.733

Table D.41: Case 13: Summary statistics of pose errors in each DoF

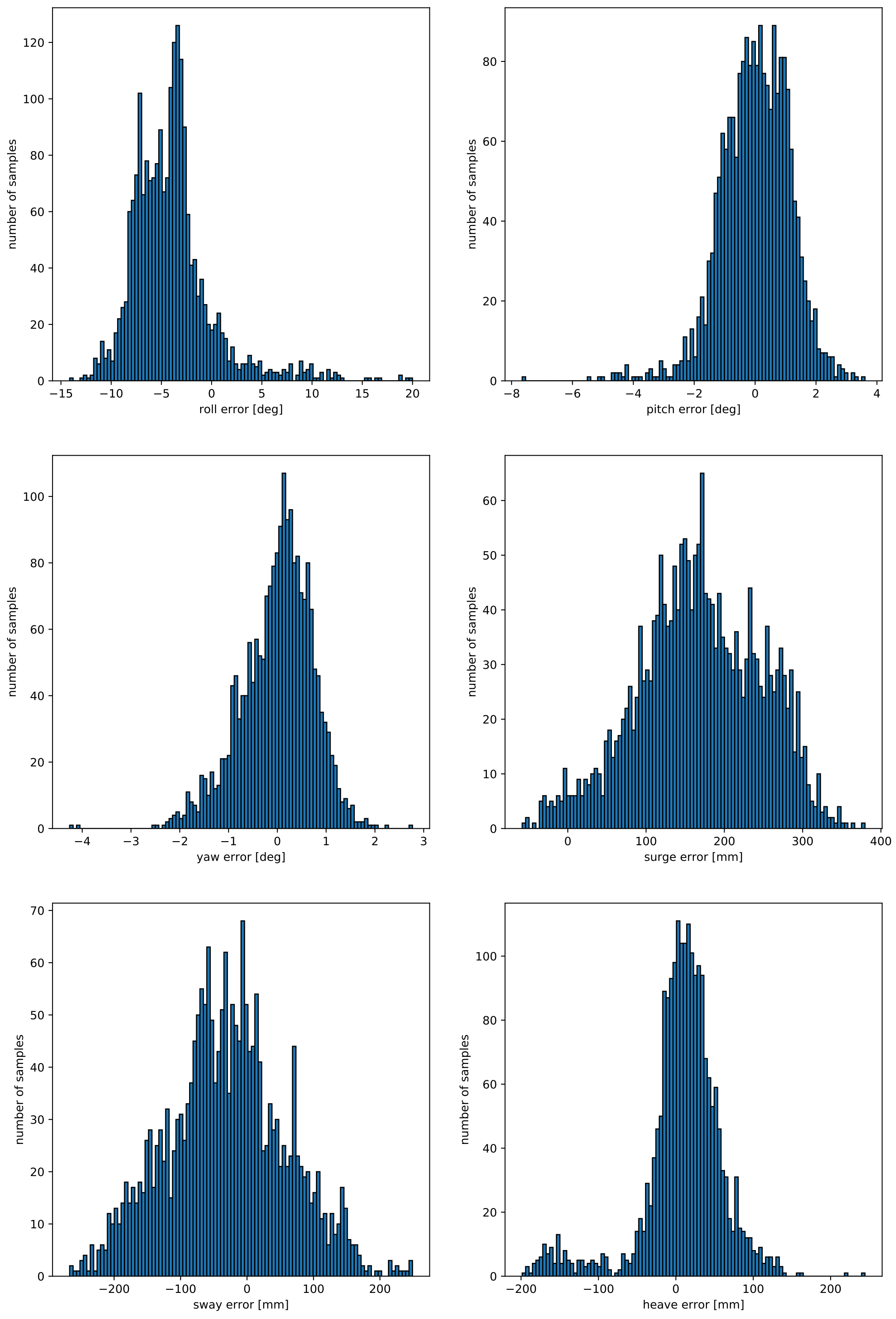


Figure D.44: Case 13: Pose errors in each DoF

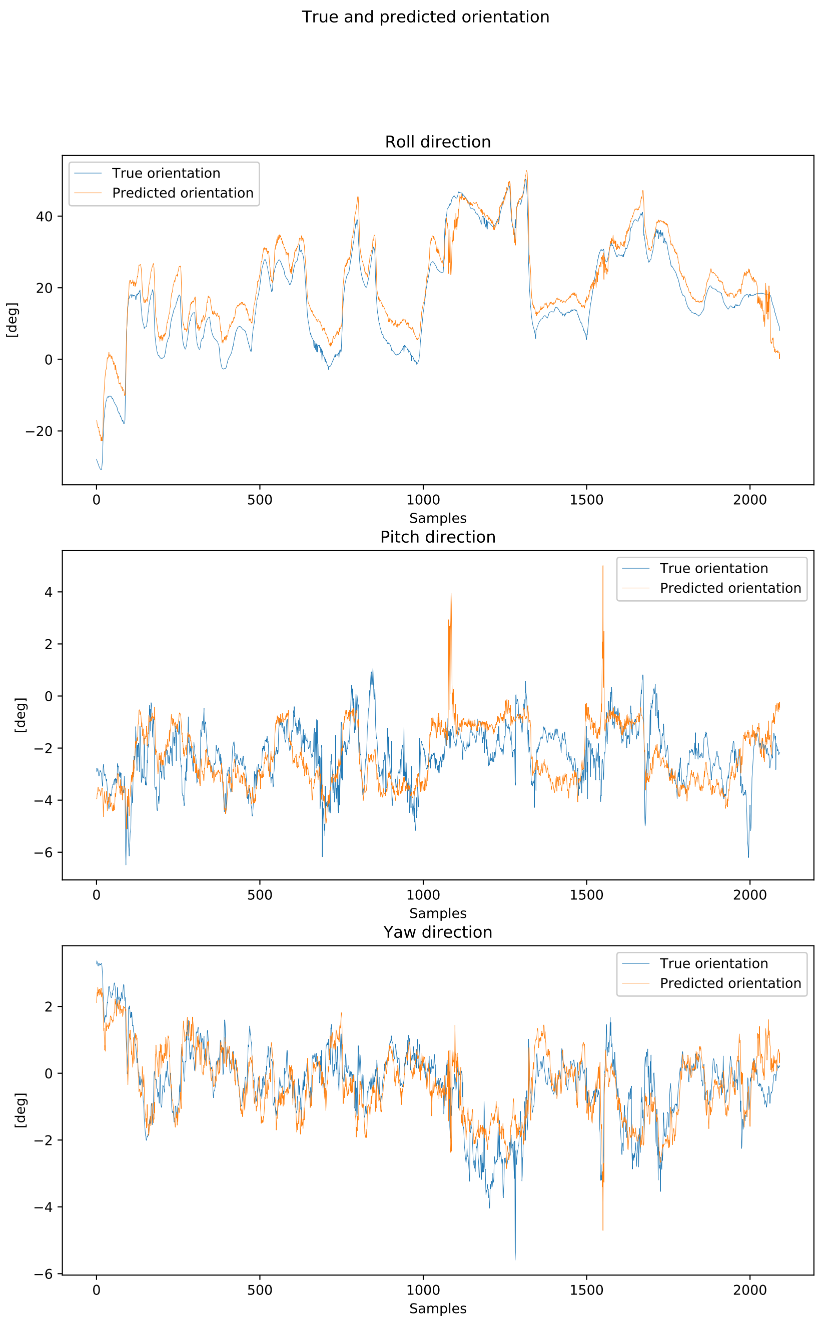


Figure D.45: Case 13: True and predicted orientation in roll, pitch and yaw

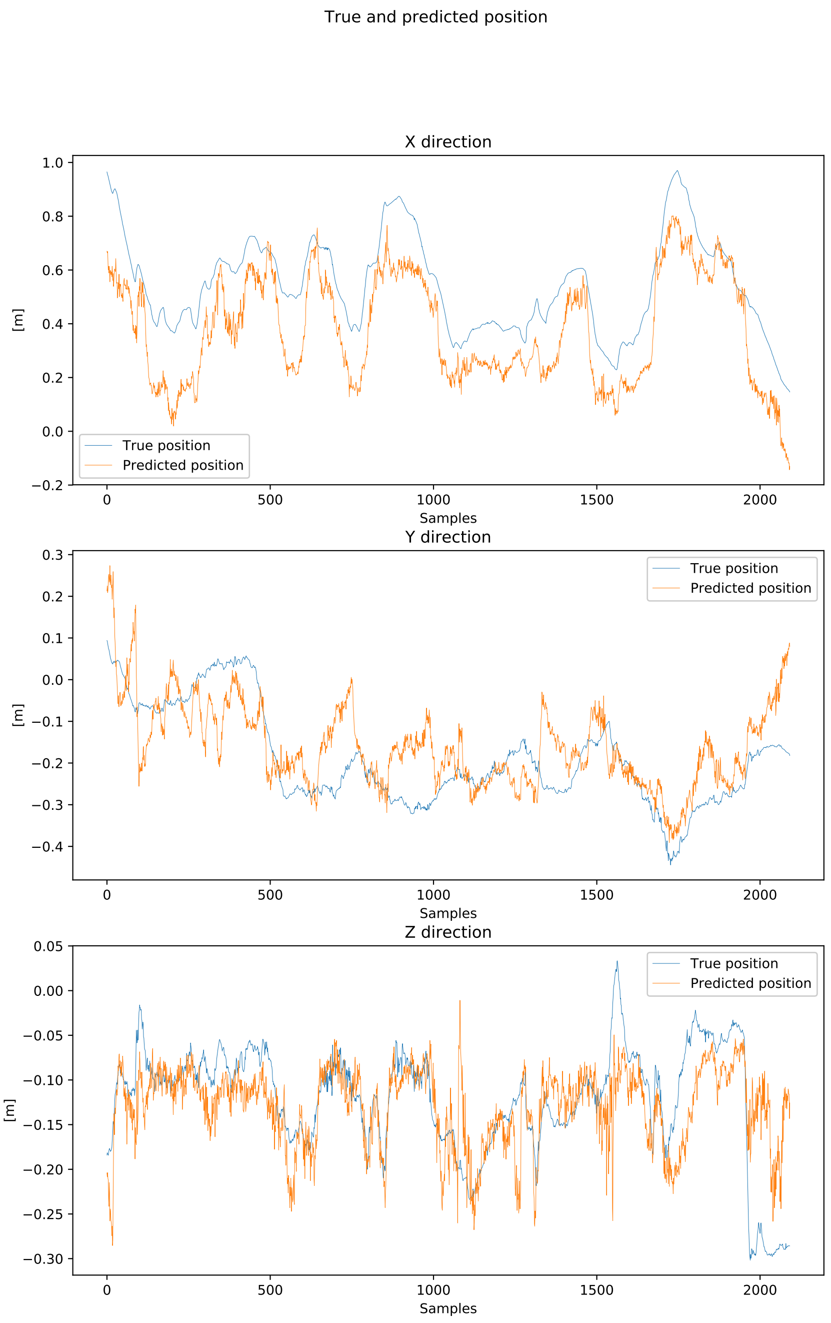


Figure D.46: Case 13: True and predicted position in surge, sway and heave

D.14 Case 14

Data ID	Description	# Train images	# Test images	Splitting
5,6	Ring on stick + ring on box	17 500		Temporal
5	Ring on stick		2 888	Temporal

Table D.42: Case 14: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	26.59	38.09	97.41
Angle error [deg]	0.965	2.042	11.349

Table D.43: Case 14: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

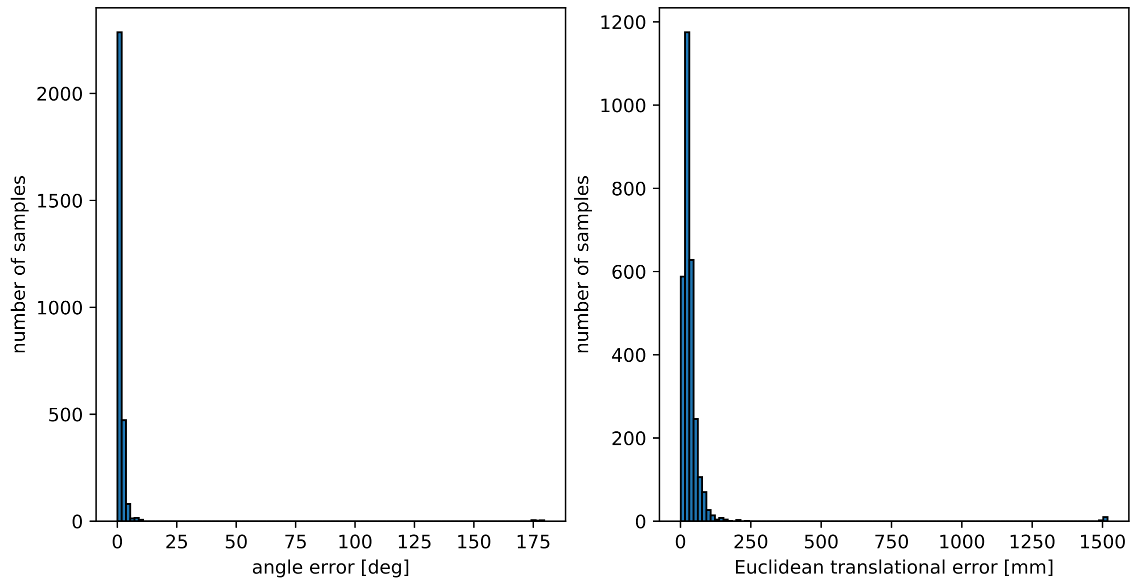


Figure D.47: Case 14: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-0.62	46.45
Y [mm]	3.34	88.34
Z [mm]	4.11	30.82
Roll [deg]	-0.402	2.613
Pitch [deg]	-0.209	1.285
Yaw [deg]	0.657	11.334

Table D.44: Case 14: Summary statistics of pose errors in each DoF

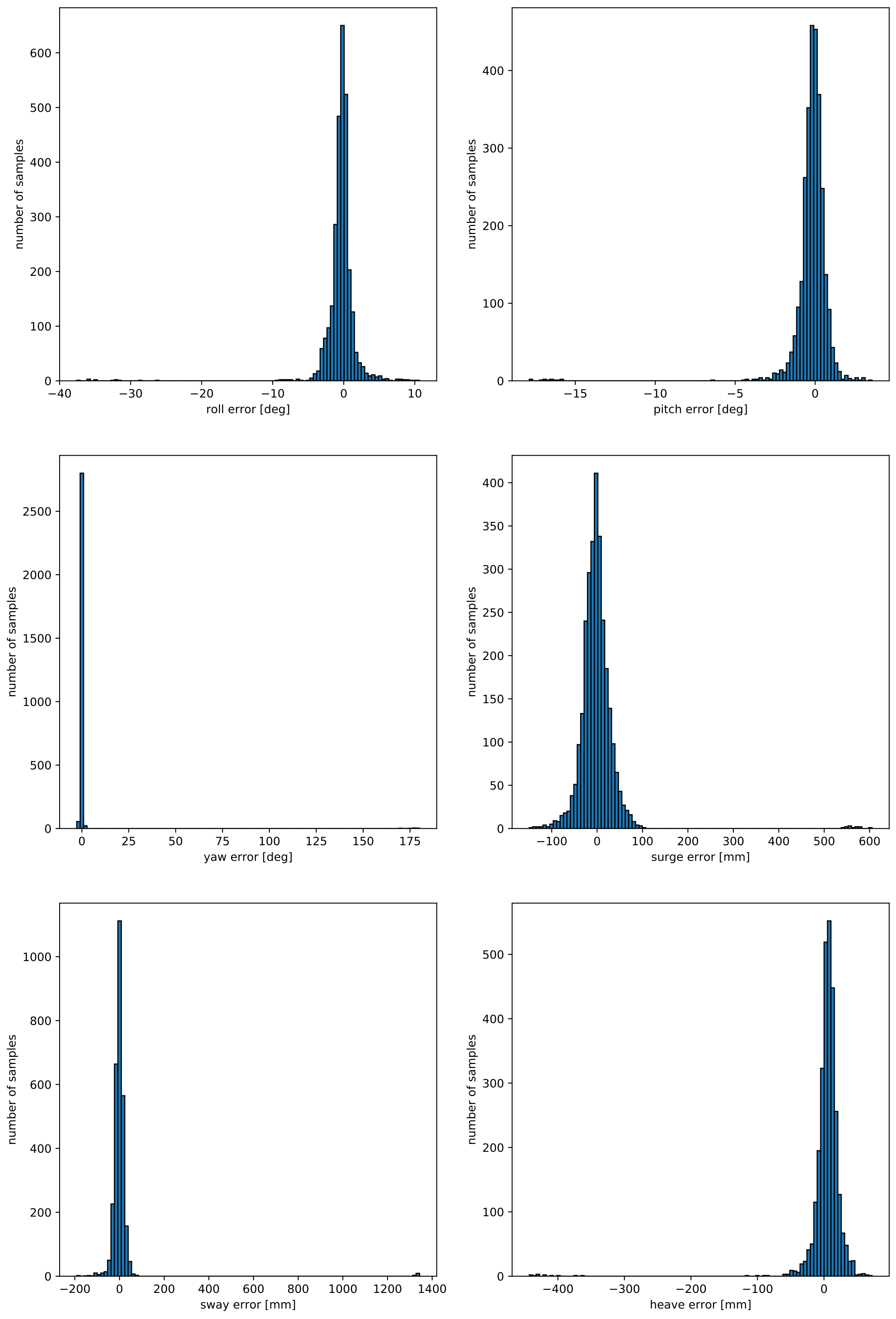


Figure D.48: Case 14: Pose errors in each DoF

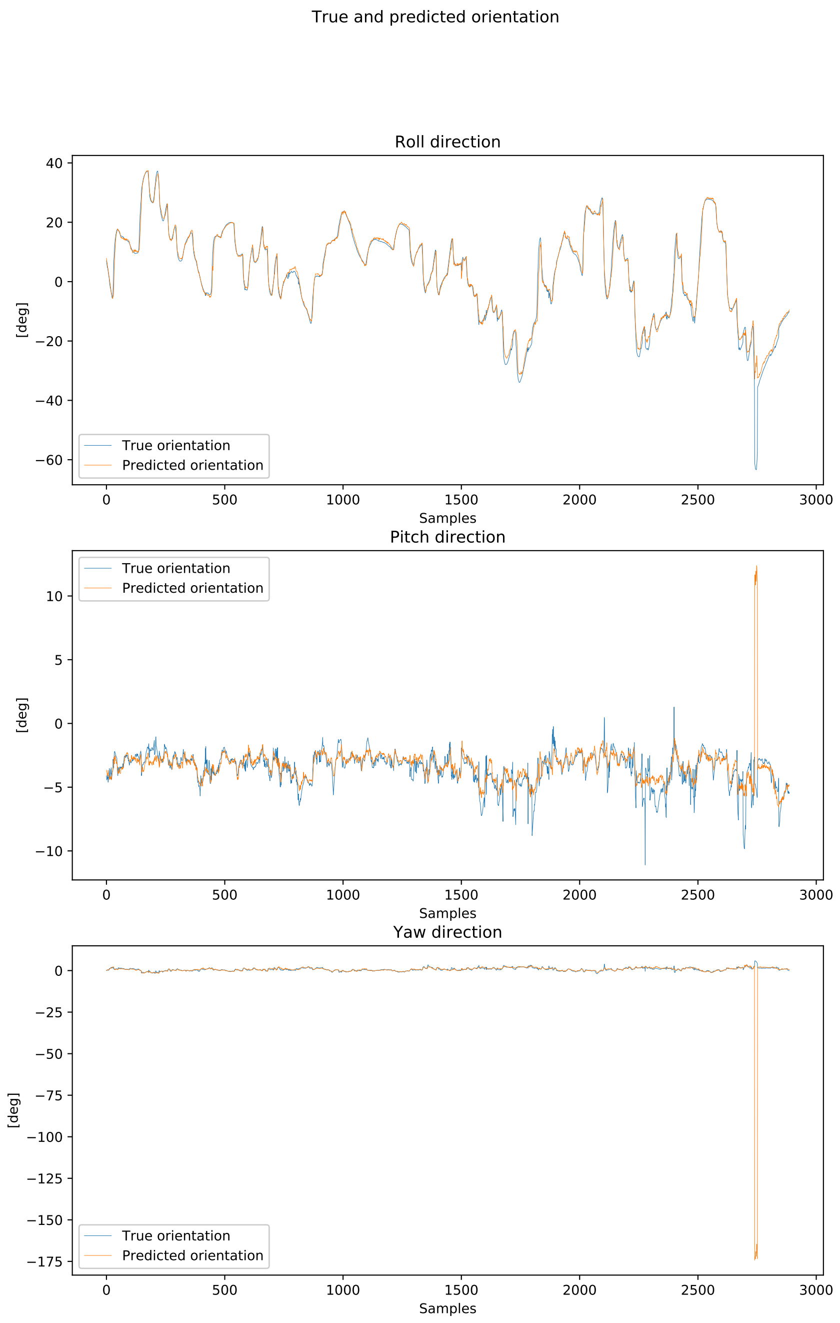


Figure D.49: Case 14: True and predicted orientation in roll, pitch and yaw

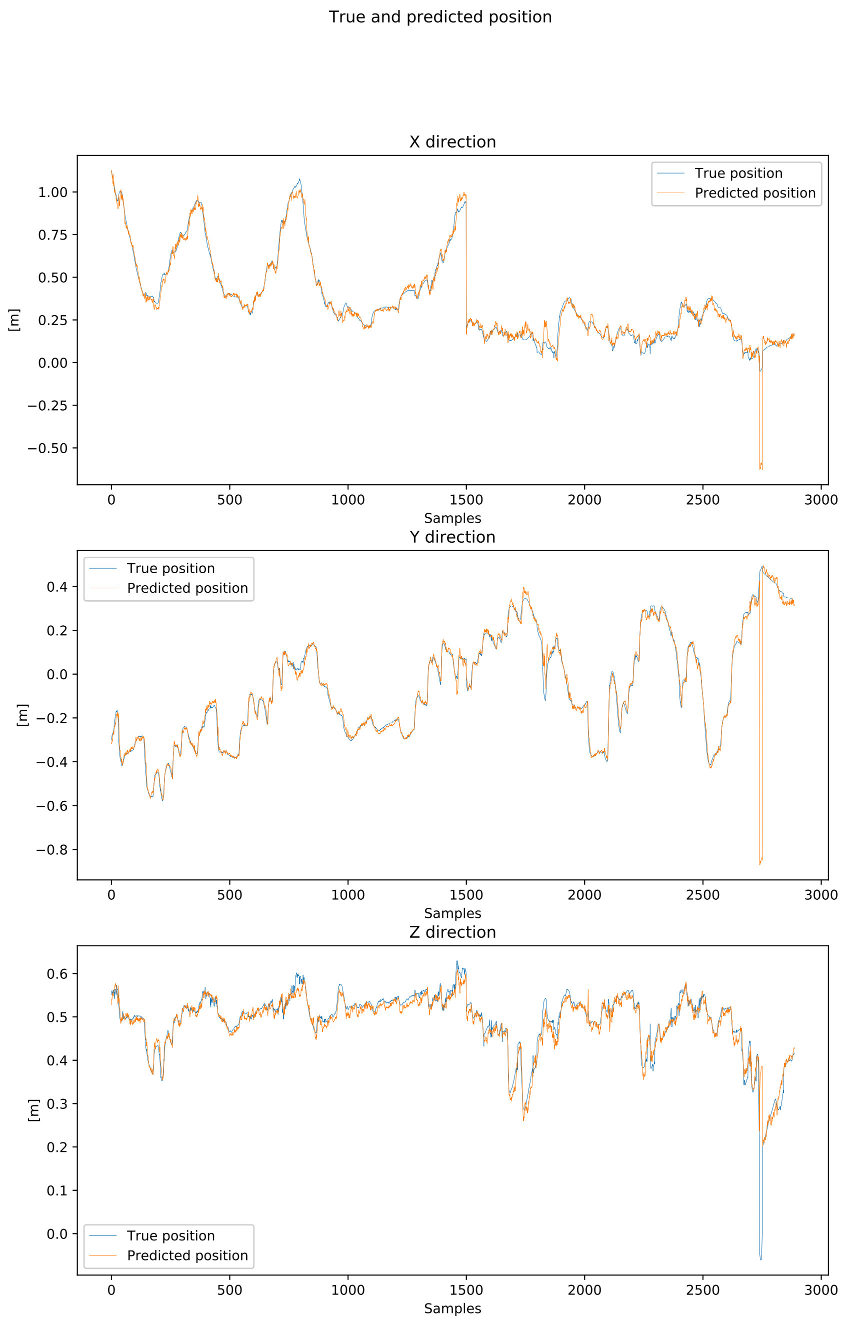


Figure D.50: Case 14: True and predicted position in surge, sway and heave

D.15 Case 15

Data ID	Description	# Train images	# Test images	Splitting
6	Ring on box	6 500		Temporal
5	Ring on stick		2 888	Temporal

Table D.45: Case 15: Details of train and test datasets

Measurement	Median	Mean	Standard Deviation
Euclidean translation error [mm]	203.21	215.21	102.28
Angle error [deg]	5.962	7.194	6.418

Table D.46: Case 15: Summary statistics of the pose errors decomposed into the relative angle error and Euclidean translation error

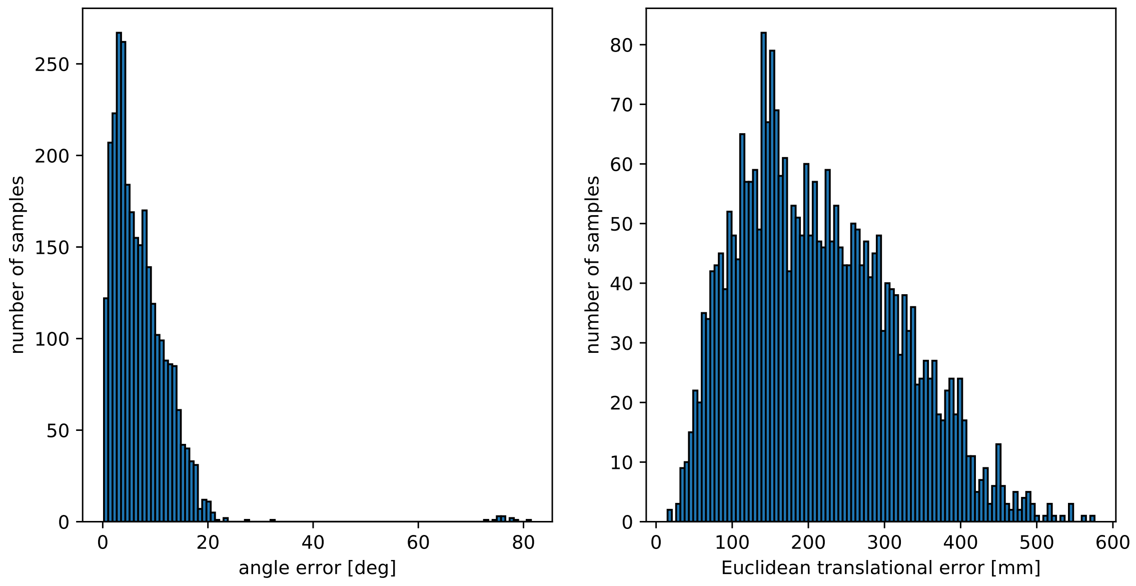


Figure D.51: Case 15: Relative angle error (left) and Euclidean translation error (right) of the pose estimates

DoF	**Mean**	**Standard Deviation**
X [mm]	-140.30	168.09
Y [mm]	-6.49	78.24
Z [mm]	24.39	45.61
Roll [deg]	0.504	9.537
Pitch [deg]	-0.383	1.144
Yaw [deg]	-0.058	0.674

Table D.47: Case 15: Summary statistics of pose errors in each DoF

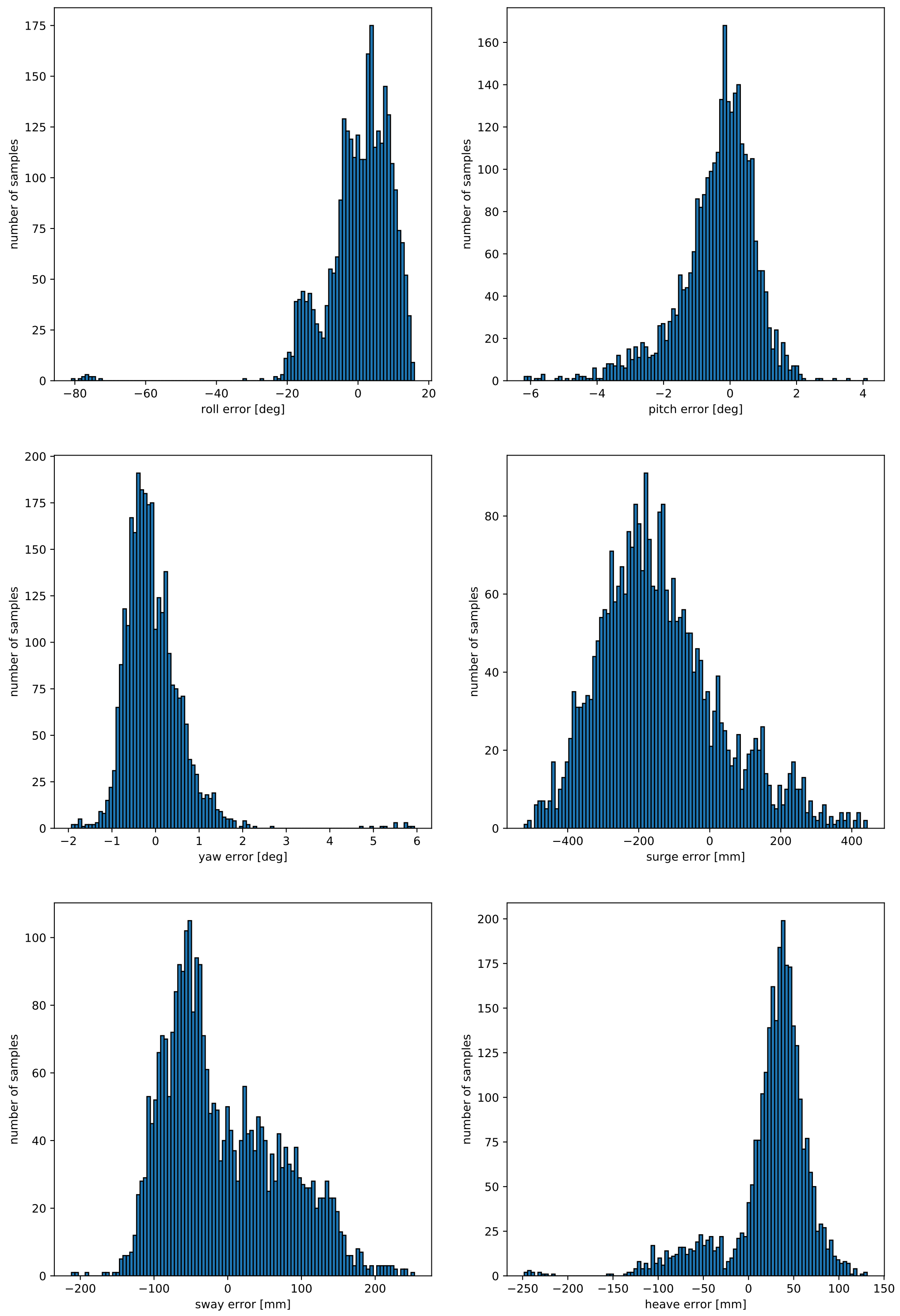


Figure D.52: Case 15: Pose errors in each DoF

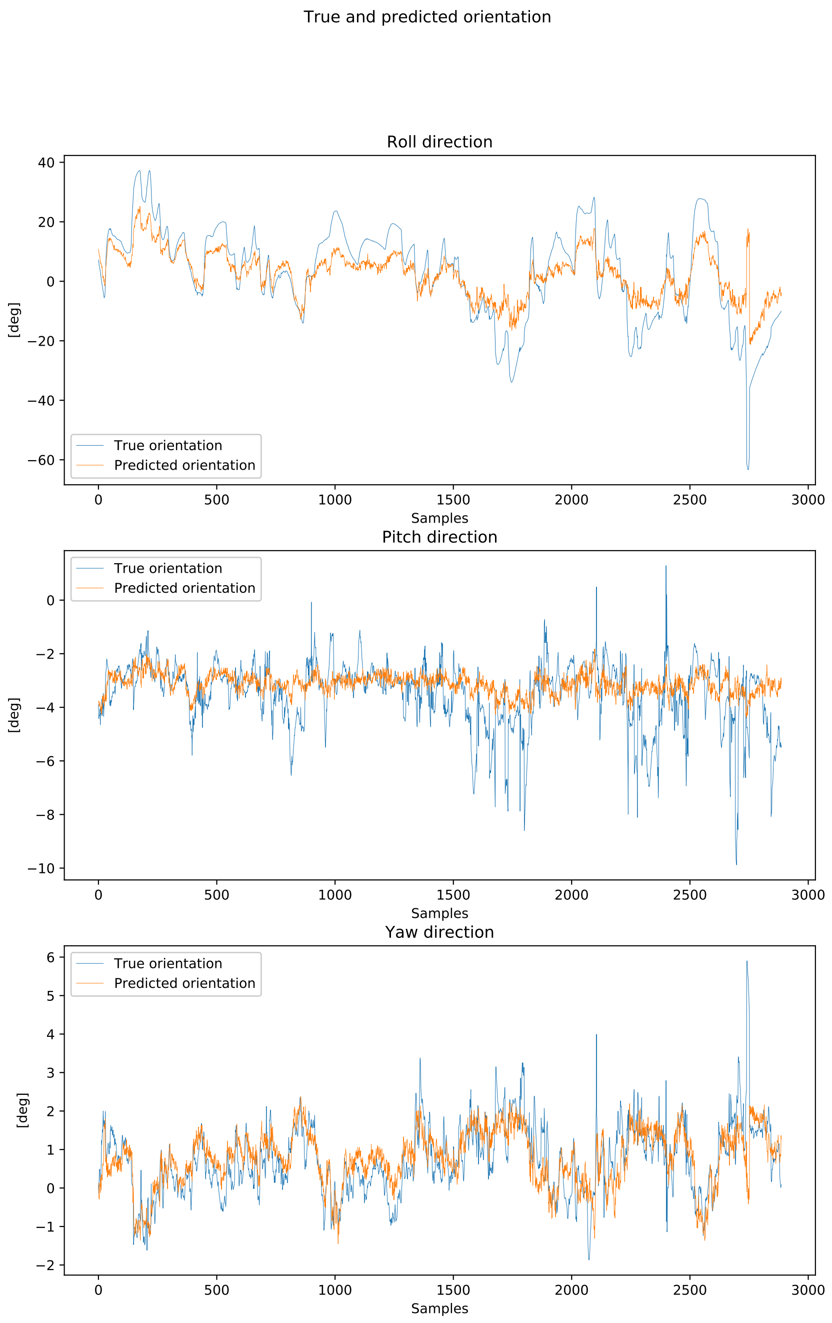


Figure D.53: Case 15: True and predicted orientation in roll, pitch and yaw

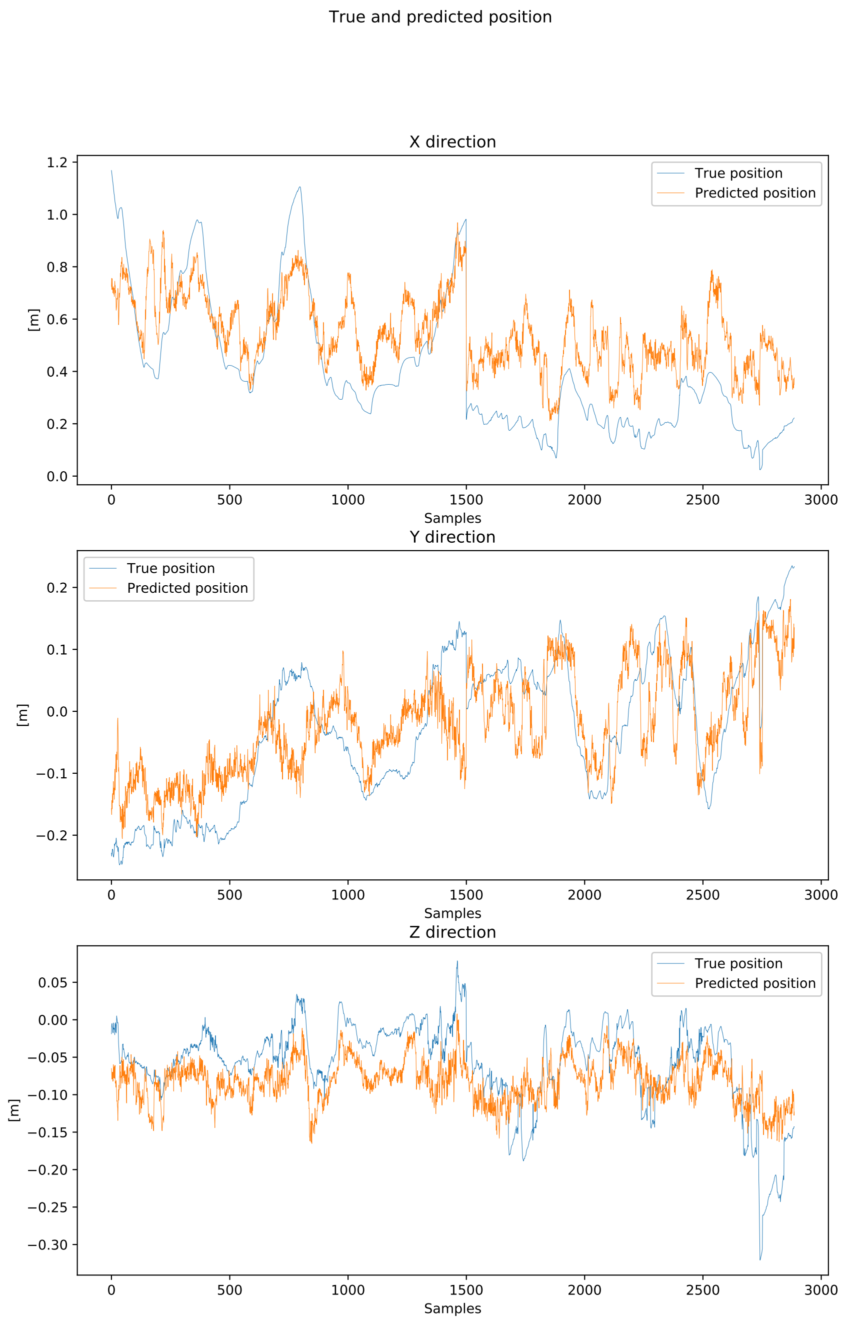


Figure D.54: Case 15: True and predicted position in surge, sway and heave

Mari Hovem Leonhardsen

Underwater Pose Estimation with Deep Learning

NTNU
Norwegian University of
Science and Technology