NTNU – Trondheim
Norwegian University of
Science and Technology

Automatic Ship Landing System for Fixed-Wing UAV

Marcus Frølich

Master of Science in Cybernetics and Robotics
Submission date: June 2015
Supervisor: Thor Inge Fossen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU – Trondheim
Norwegian University of
Science and Technology

Automatic Ship Landing System for Fixed-Wing UAV

Marcus Frølich

MSc Thesis in Engineering Cybernetics
Submisson date: June 14, 2015
Supervisor: Thor I. Fossen
Co-supervisor: João Fortuna

Norwegian University of Science and Technology
Department of Engineering Cybernetics

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

MSC THESIS DESCRIPTION SHEET

Name:	Marcus Frølich
Department:	Engineering Cybernetics
Thesis title (Norwegian):	Automatisk landingssystem for ubemannet fly ombord i skip
Thesis title (English):	Automatic Ship Landing System for Fixed-Wing UAV

Thesis Description: The purpose of this project is to design and verify the performance of an automatic landing system for a fixed-wing UAV to be used with a net on board a moving ship. The following items should be considered:

1. Define the scope of the thesis and clarify what your contributions are.
2. Design of optimal guidance systems for automatic landing onboard ships.
3. Design of system for ship motion prediction to ensure safe landing onboard moving ships.
4. Implement and test the ship motion prediction system and the automatic landing system for an X8 fixed-wing UAV using climb rate and bank angle commands.
5. Conclude your results.

Start date:	2015-01-05
Due date:	2015-06-15

Thesis performed at:	Department of Engineering Cybernetics, NTNU
Supervisor:	Professor Thor I. Fossen, Dept. of Eng. Cybernetics, NTNU
Co-Supervisor:	João Fortuna, Dept. of Eng. Cybernetics, NTNU

Abstract

This thesis presents the system design of an automatic ship landing system for a fixed-wing unmanned aerial vehicle (UAV) intended to land in a net. It involves path planning, guidance and control, accurate navigation and ship motion prediction. Fully functional, the automatic landing system is intended to remove the need for a pilot when operating UAVs from a ship or other platforms with confined space.

Dynamic flight paths for landing were developed using both piece-wise continuous interpolation (for simplicity) and Dubins path (for feasibility). The system uses the Pixhawk autopilot and a software toolchain from Underwater Systems and Technology Laboratory (LSTS), Porto, for vehicle control, mission review and communication. Either waypoints or climb rate and bank angle commands can be calculated in the LSTS toolchain and sent to Pixhawk. Real-Time Kinematic GPS (RTK-GPS) is used for accurate navigation. To ensure safe landing on a ship influenced by wind, waves and current, artificial neural network (ANN), an artificial intelligence (AI) method, is used for ship motion prediction.

Software-in-the-loop (SIL) simulations demonstrated successful performance of the automatic landing system, where the UAV hit the landing target with reasonable accuracy. This was done by developing a decoupled guidance system to send climb rate and bank angle commands to the Pixhawk. Using simulated data from a ship influenced by waves and current, the ANN ship motion prediction system sufficiently performed predictions of heading and heave displacement for up to 60 seconds. A novel approach to ANN pre- and post-processing was found to increase heading prediction accuracy compared to conventional approaches, while ANN data fusion increased the performance and robustness of heave prediction compared to using a single data type.

Keywords: Unmanned Aerial Vehicle (UAV), Guidance Navigation and Control (GNC), Net Landing, Real-Time Kinematic GPS (RTK-GPS), Artificial Neural Network (ANN), Artificial Intelligence (AI), Ship Motion Prediction, Data Fusion

Sammendrag

Denne avhandlingen presenterer systemdesignet til et automatisk landingssystem for et ubemannet fly (drone) til bruk ved landing i nett ombord på skip. Det innebærer baneplanlegging, kontrollsystemer, nøyaktig navigering og bevegelsesprediksjon. Ferdig utviklet er det automatiske landingssystemet ment å kunne fjerne behovet for en pilot ved droneopperasjoner fra skip eller andre plattformer med begrenset plass.

Dynamisk flygebane for landing er utviklet ved bruk av stykkevis kontinuerlig interpolasjon (rette baner, for enkelhets skyld), samt Dubins bane (for økt gjennomførbarhet). Systemet bruker autopiloten Pixhawk og Underwater Systems and Technology Laboratory (LSTS), Porto, sin programvarekjede for fartøyskontroll, oppdragsanalye og kommunikasjon. Enten rutepunkt eller høyderate- og rullvinkel-kommandoer kan bli utregnet i LSTS programvarekjeden og sendt til Pixhawken. Real-Time Kinematic GPS (RTK-GPS) brukes for nøyaktig posisjonering. For å sikre en trygg landing ombord på et skip påvirket av vind, bølger og strømninger, blir kunstig nevrale nettverk (ANN), en metode inne kunstig intelligens (AI), brukt for bevegelsesprediksjon av skipet.

Programvare-i-sløyfen (SIL) simuleringer demonstrerte suksessfulle tester for det automatiske landingssystemet, hvor dronen traff målet med relativt god presisjon. Dette ble gjort ved å utvikle et dekoblet overordnet kontrollsystem som sendte høyderate- og rullvinkel-kommandoer til Pixhawken. Ved bruk av simulert data fra et skip påvirket av bølger og strømninger, utførte systemet for ANN bevegelsesprediksjon vellykkede tester for prediksjon av kurs og hiv opp til 60 sekunder frem i tid. En ny tilnærming til ANN pre- og postprosessering viste seg å øke nøyaktigheten for kursprediksjon sammenlignet med ordinære metoder, mens ANN datafusjon økte nøyaktigheten og robustheten til hivprediksjon sammenlignet med bruk av bare en type data.

Acknowledgments

I would like to thank my supervisor Professor Thor I. Fossen for guidance during this project and valuable feedback on the thesis. I would also like to thank PhD candidate João Fortuna for helping out with everything related to the LSTS toolchain, and all the other guys at the NTNU UAV-Lab and at the office. Moreover, a thank to Bjørn A. Spockeli for cooperation on a preliminary report used as a basis for parts of this thesis.

Marcus Frølich
Trondheim, June 2015

Contents

MSc Thesis Description Sheet . . . ii
Abstract . . . iii
Sammendrag . . . iv
Acknowledgments . . . vi
List of Figures . . . xiii
List of Tables . . . xvii
Abbreviations . . . xix

1 Introduction 1
1.1 Background and Motivation . . . 1
1.2 Previous Work . . . 2
1.3 Contribution and Goal of the Thesis . . . 3
1.4 Organization of the Thesis . . . 3

2 Background Theory 5
2.1 Path Planning . . . 5
2.1.1 Piece-Wise Continuous Interpolation . . . 6
2.1.2 Dubins Path . . . 6
2.1.3 Clothoids . . . 7
2.1.4 Fermat's Spiral Path . . . 8
2.1.5 Monotone Cubic Hermite Spline Interpolation . . . 8
2.2 Guidance . . . 9
2.2.1 Beam-Rider . . . 10
2.2.2 Pure-Pursuit . . . 10
2.2.3 Constant Bearing . . . 11
2.2.4 Proportional Navigation . . . 12
2.2.5 Proportional-Integral-Derivative Controller . . . 13
2.3 Artificial Neural Network (ANN) . . . 13
2.3.1 Overview of ANNs . . . 14
2.3.1.1 Interconnection Pattern . . . 16
2.3.1.2 Activation Function . . . 16
2.3.1.3 Number of Hidden Layers . . . 18
2.3.1.4 Training . . . 19
2.3.2 Data Fusion Using ANNs . . . 22
2.4 Time Series Forecasting . . . 22
2.4.1 Autoregressive Integrated Moving Average . . . 23
2.4.2 Kalman Filter . . . 23
2.4.3 Artificial Neural Network . . . 23
2.4.4 Hybrid Approaches . . . 25

3 System Components 27
3.1 Software . . . 27

3.1.1 LSTS Toolchain 27
3.1.1.1 GLUED 27
3.1.1.2 IMC 28
3.1.1.3 DUNE 28
3.1.1.4 Neptus 29
3.1.2 Mission Planner 30
3.1.3 MAVProxy 30
3.1.4 APM:Plane 30
3.2 Hardware 30
3.2.1 UAV 31
3.2.2 Embedded Computer 32
3.2.3 Autopilot 33

4 Method **35**
4.1 System Overview 35
4.2 Ship Motion Prediction 37
4.2.1 Performance Measurements 37
4.2.2 ANN Time Series Forecasting 38
4.2.3 ANN Forecasting for Oscillations With Varying Mean 39
4.2.3.1 Problem Review 40
4.2.3.2 Suggested Solutions 41
4.2.4 Estimated Time of Arrival 48
4.3 Landing Path 48
4.3.1 Lateral Path 49
4.3.2 Longitudinal Path 50
4.3.3 Dynamic path 54
4.3.4 Evasive Maneuver 55
4.3.4.1 Evasive Maneuver Path 57
4.4 Guidance and Control System 58
4.4.1 Lateral Guidance 59
4.4.2 Longitudinal Guidance 60
4.4.3 APM:Plane 62
4.5 Landing Simulation, Software-In-the-Loop 65
4.5.1 Simulation Softwares 65
4.5.2 Neptus Automatic UAV Landing Plug-In 66
4.5.3 Landing Path Implementation 69
4.6 Accurate Navigation 72

5 Experimental Testing **75**
5.1 Time Series Forecasting 75
5.1.1 Forecasting Criteria 75
5.1.2 Oscillations Around a Varying Mean 76
5.1.3 Discussion - Oscillations Around a Varying Mean 79
5.1.4 Ship Simulation 79
5.1.5 Ship Heading Prediction 81

5.1.6 Ship Heading Prediction - Supply Vessel 84
5.1.7 Discussion - Ship Heading Prediction 87
5.1.8 Heave Displacement Prediction 87
5.1.9 Heave Displacement Prediction - Data Fusion 91
5.1.10 Heave Displacement Prediction - Added Noise 92
5.1.11 Discussion - Heave Displacement Prediction 94
5.2 UAV Landing - Software-In-the-Loop (SIL) 95
5.2.1 Landing Criteria 96
5.2.2 SIL Simulation - Static Path, No Wind 96
5.2.3 SIL Simulation - Static Path, Wind and Turbulence 100
5.2.4 Discussion - Static Path 102
5.2.5 SIL Simulation - Evasive Maneuver 102
5.2.6 SIL Simulation - Dynamic Path, No Wind 106
5.2.7 SIL Simulation - Dynamic Path, Wind and Turbulence 111
5.2.8 Discussion - Dynamic Path 113

6 Closing Discussion and Conclusion 115
6.1 Closing Discussion and Conclusion 115
6.2 Future Work 117

Appendix A Guidance and Control System 119
A.1 Preliminary Experiments 119
A.1.1 Waypoint Guidance 119
A.1.2 Aerosonde Lateral Controller 122
A.1.3 Sliding Mode Height Controller 123
A.2 ILOS Controller Tuning Values 124
A.3 Bank-To-Turn Controller Tuning Values 124
A.4 PID Height Controller Tuning Values 126

Appendix B Time Series Forecasting 127
B.1 Oscillations Around a Varying Mean 127
B.2 Ship Simulation 127
B.3 Ship Heading Prediction 130
B.3.1 Methods and Activation Functions 130
B.3.2 Methods and Hidden Neurons 131
B.4 Heave Displacement Prediction 134
B.5 Heave Displacement Prediction - Data Fusion 137

Appendix C Landing Simulation, Software-In-the-Loop 139
C.1 Waypoint Path Calculation 139
C.2 Loiter Alignment Path 140
C.3 Static Path, No Wind 140
C.4 Static Path, Wind and Turbulence 141
C.5 Dynamic Path, Wind and Turbulence 144

Bibliography 147

List of Figures

2.1 Piece-wise continuous interpolated path 6
2.2 Dubins path 7
2.3 Clothoid path 7
2.4 Monotone cubic and regular cubic path 8
2.5 Beam-rider guidance 10
2.6 Pure-pursuit guidance 11
2.7 Constant Bearing 11
2.8 Proportional Navigation 12
2.9 A single artificial neuron with three inputs 15
2.10 A three layer feed forward ANN with two inputs, three hidden neurons and one output. A circle represents an artificial neuron and an arrow a direct connection 16
2.11 Two activation functions: binary threshold and sigmoid 17
2.12 Two activation functions: linear and Gaussian 18
2.13 Simplified flow chart of a general training process for ANN, from [35] 20
2.14 Feature-based data fusion using ANNs in cascade 22
2.15 Time series forecasting using ANNs 24

3.1 IMC message bus serving DUNE and Neptus 29
3.2 X8 Skywalker 31
3.3 Pandaboard and BeagleBone Black dimensions 32
3.4 BeagleBone Black 33
3.5 Pixhawk from 3DRobotics 34

4.1 Software and hardware architecture of the automatic landing system with the base station to the left and UAV to the right 36
4.2 Simple example of yaw motion for a wave influenced ship 39
4.3 Pre- and post-processing for an ANN 41
4.4 *Unchanged* example 43
4.5 *Mean* processing example 44
4.6 *Median* processing example 45
4.7 *Current* processing example 46
4.8 *Normalization* processing example 47
4.9 Lateral path example 50
4.10 Longitudinal straight line path. Description of variables in Table 4.1 . 51
4.11 Example of a Dubins path for climb rate guidance 52
4.12 Example of the derivative of a Dubins path, giving desired climb rate ratio 53
4.13 An example of a dynamic path with an update requested during the approach phase 54
4.14 Course over ground vs. heading - NE plane, from [73] 56
4.15 Lateral path after an evasive maneuver is ordered during FA 58

4.16 Simplified successive loop closure for lateral guidance 60
4.17 Demonstration of the lookahead distance for the proactive PID controller 62
4.18 Software architecture for climb rate and roll angle guidance 64
4.19 System architecture of the APM:Plane SIL simulations 66
4.20 Neptus plug-in context menu (right-click) and net arrow icon 67
4.21 Plug-in menu with land icon marked in a red circle 67
4.22 Neptus plug-in parameter settings menu 68
4.23 Overview of the landing path generation process 70
4.24 The Neptus tool *IMC Message Sender* 71
4.25 Overview of the dynamic landing path generation process 71
4.26 Flowchart of RTK-GPS-Pixhawk HIL simulation. Software from the desktop computer (desktop) is connected to the external Pixhawk autopilot (external) 73
4.27 RTK-GPS-Pixhawk integration 74

5.1 Simple yaw motion, split into training set (300 seconds) and test set (200 seconds) 77
5.2 ANN test on a simple yaw motion, demonstrating performance of different methods. *Mean*, *median*, *current* and *normalization* are almost completely overlapping on top of the target function 78
5.3 Simulated sea state realization, generated from MSS [25] 80
5.4 Yaw motion of the simulated ship S175 81
5.5 Ship heading prediction with different methods and numbers of hidden nodes. Only results for *mean*, *median* and *current* are presented . 82
5.6 Training time for different methods and numbers of hidden nodes . . 83
5.7 Ship heading prediction test using processing method *median* 84
5.8 Yaw motion of the simulated supply vessel 85
5.9 Ship heading prediction test for supply vessel using processing method *current* 86
5.10 Heave motion of the simulated ship S175 87
5.11 Heave displacement prediction MSE for different numbers of hidden neurons and different activation functions 88
5.12 Heave displacement prediction 10 seconds ahead 90
5.13 Heave displacement prediction 60 seconds ahead 90
5.14 Heave data plus added noise 93
5.15 Example of lateral landing path with the added loiter circle 96
5.16 Altitude plot of one flight, static path - no wind 97
5.17 Lateral trajectory plot of one flight, static path - no wind 98
5.18 Altitude error of 5 flights, 10 last seconds, static path - no wind . . . 98
5.19 Cross-track error of 5 flights, static path - no wind 99
5.20 Cross-track error of 5 flights, 10 last seconds, static path - no wind . 99
5.21 Altitude error of 5 flights, 10 last seconds, static path - wind and turbulence 100

5.22 Cross-track error of 5 flights, 10 last seconds, static path - wind and turbulence . . . 101
5.23 Difference between COG and heading, static path - wind and turbulence 101
5.24 Lateral trajectory plot of one flight with net marked as green circle, evasive maneuver . . . 103
5.25 Relative lateral trajectory of 5 flights right after an evasive maneuver is requested. 3.5 m safety radius around the net is marked as a black circle . . . 104
5.26 Relative altitude of 5 flights, where request for evasive maneuver is at time 0 s and altitude 0 m . . . 105
5.27 Relative altitude of 5 flights, where request for evasive maneuver is at time 0 s and altitude 0 m. Zoomed in view . . . 105
5.28 Altitude plot of one flight, dynamic path - no wind . . . 107
5.29 Lateral trajectory plot of one flight, dynamic path - no wind . . . 108
5.30 Altitude error of 5 flights - starting from path update, dynamic path - no wind . . . 109
5.31 Altitude error of 5 flights, 10 last seconds, dynamic path - no wind . . 109
5.32 Cross-track error of 5 flights, from path update, dynamic path - no wind . . . 110
5.33 Cross-track error of 5 flights, 10 last seconds, dynamic path - no wind 110
5.34 Altitude error of 5 flights, 10 last seconds, dynamic path - wind and turbulence . . . 111
5.35 Cross-track error of 5 flights, 10 last seconds, dynamic path - wind and turbulence . . . 112
5.36 Difference between COG and heading, static path - wind and turbulence 112

A.1 SIL simulation with waypoints sent directly to APM:Plane . . . 120
A.2 SIL simulation with waypoints sent directly to APM:Plane, 15 last seconds . . . 121
A.3 Cross-track error from SIL simulation with Aerosonde lateral controller 122
A.4 Cross-track error from SIL simulation with Aerosonde lateral controller, last 30 seconds . . . 123
A.5 Height plot from SIL simulation with sliding mode height controller . 124
A.6 Straight line test of BTT response, from $-120°$ . . . 125
A.7 Straight line test of BTT response, from $-50°$ . . . 125

B.1 Wave settings for the ship simulation . . . 128
B.2 PSD plot of the heave motion with ShipX: S175 . . . 129
B.3 Peaks in a heave displacement plot . . . 130
B.4 MSE for different methods and numbers of hidden nodes . . . 133
B.5 MSE for different prediction intervals and number of hidden neurons . 134
B.6 Percentage of errors withing 0.25 m at peaks for different prediction intervals and number of hidden neurons . . . 135
B.7 Percentage of errors withing 0.25 m for different prediction intervals and number of hidden neurons . . . 135

B.8 60 seconds prediction step performance with combinations of input time range/sampling frequency 136
B.9 60 seconds prediction step performance, data fusion, with combinations of input time range/sampling frequency 137

C.1 Altitude error of 5 flights, static path - no wind 141
C.2 Wind direction during static path landing - wind and turbulence . . . 142
C.3 Wind speed during static path landing - wind and turbulence 142
C.4 Altitude error of 5 flights, static path - wind and turbulence 143
C.5 Cross-track error of 5 flights, static path - wind and turbulence 143
C.6 Altitude error of 5 flights, from path update, dynamic path - wind and turbulence 144
C.7 Cross-track error of 5 flights, from path update, dynamic path - wind and turbulence 145

List of Tables

4.1 Description of variables mentioned in Figure 4.10 52

5.1 Results from simple yaw motion test 79
5.2 Ship heading prediction for 60 seconds 84
5.3 Ship heading prediction for 60 seconds, supply vessel 86
5.4 Prediction steps with their necessary combinations of input time range/sampling frequency 89
5.5 Final test with different steps for heave displacement prediction . . . 91
5.6 Prediction steps, data fusion, with their necessary combinations of input time range/sampling frequency 91
5.7 Final test, data fusion, with different steps for heave displacement prediction 92
5.8 Heave displacement prediction, 30 seconds prediction step. With and without added noise 94
5.9 Net pose change, sent as *IMC::DeviceState* 107

A.1 Tuning values for the ILOS controller implemented in DUNE 124
A.2 Tuning values for the BTT controller implemented in DUNE 126
A.3 Tuning values for the PID height controller 126

B.1 ShipX main dimensions 127
B.2 MSE [°] for all methods and activation functions 131
B.3 Training time [s] for all methods and activation functions 131
B.4 Method - *Unchanged* 131
B.5 Method - *Mean* 132
B.6 Method - *Median* 132
B.7 Method - *Current* 132
B.8 Method - *Normalization* 132

C.1 Values defining the landing path from Figure 4.10. The UAV height is assumed 50.0 m (h2), but varies with every individual flight 139
C.2 Calculated waypoints for the landing path described in Table C.1 . . 140
C.3 Values defining the loiter path used before initiating landing 140

Abbreviations

ADALINE Adaptive Linear Element Network. 25

AI Artificial Intelligence. iii, v, 1, 2, 13

ANN Artificial Neural Network. iii, v, xiii, xiv, 1, 2, 13–25, 37–41, 47, 48, 75–78, 81–84, 87–89, 91–95, 115–117

APM ArduPilot Mega. 30, 58, 65, 119

APM:Plane ArduPilot Mega Plane. xiv, xv, 30, 35, 50, 58–60, 62, 63, 65, 66, 72, 106, 116, 119–121

AR Ambiguity Resolution. 56

ARIMA AutoRegressive Integrated Moving Average. 23, 25

BP Back-Propagation. 20, 22, 25, 40, 77

BTT Bank-To-Turn. xv, xvii, 59, 60, 102, 116, 124–126

CCD Charge-Coupled Device. 22

CG Conjugate Gradient. 21, 22

COG Course Over Ground. xv, 56, 97, 100–102, 111–113, 124

CPU Central Processing Unit. 30, 66

DGPS Differential GPS. 2

DOF Degree Of Freedom. 65, 79, 80

DUNE DUNE: Unified Navigation Environment. xiii, xvii, 27–29, 35, 48, 58–60, 63, 65, 69, 71–73, 75, 95, 106, 116–119, 122–124, 126, 141

ETA Estimated Time of Arrival. 37, 48, 62, 88, 102, 106, 118

FA Final Approach. xiii, 2, 49, 50, 52, 55, 58, 62, 96, 102, 116, 117

FBWB Fly-By-Wire B. 62, 63

FeNN Feature Neural Network. 22

GLUED GNU/Linux Uniform Environment Distribution. 27, 28, 35

GNSS Global Navigation Satellite System. 1, 72

GPS Global Positioning System. 1, 31, 33, 35, 39, 72, 73, 117

HIL Hardware-In-the-Loop. xiv, 29, 65, 72, 73

I^{2}C Inter-Integrated Circuit. 32, 33, 65

ILOS Integral Line-Of-Sight. xvii, 59, 60, 102, 116, 119, 124

IMC Inter-Module Communication. xiii, xiv, 27–30, 65, 69, 71, 73, 106

IMU Inertial Measurement Unit. 35, 72

INS Inertial Navigation System. 35

ITTC International Towing Tank Conference. 79

JSBSim Jon S. Berndt Simulator. 65, 71–73, 116, 141

LOS Line-of-Sight. 9–12, 59, 102

LSTS Underwater Systems and Technology Laboratory. iii, v, vii, 3, 27, 28, 32

MAE Mean Absolute Error. 37, 38, 79, 83–86, 91, 92, 94, 131, 132

MAVLink Micro Air Vehicle Link. 30, 65, 66

MB Megabyte. 27

MLP Multi-Layer Perceptron. 19, 21, 24, 37

MRA Mission Review and Analysis. 30, 95

MSE Mean Squared Error. xiv, xv, xvii, 37, 79, 83, 88, 91, 92, 94, 130–134

MSS Marine Systems Simulator. xiv, 79, 80, 84

NMEA National Marine Electronics Association. 72, 73

NP Nondeterministic Polynomial time. 5

NTNU Norwegian University of Science and Technology. vii, 2, 31, 32, 55, 72, 73, 117

PID Proportional-Integral-Derivative. xiv, xvii, 2, 13, 59, 61, 62, 102, 116, 119, 126

PN Proportional Navigation. 12

PSD Power Spectral Density. xv, 80, 129

PWM Pulse-Width Modulation. 63

RBF Radial Basis Function. 18, 25

RTK-GPS Real-Time Kinematic GPS. iii, v, xiv, 1–3, 35, 55, 56, 72–74, 115, 117

RTKLIB Real-Time Kinematic Library. 35, 72, 73, 117

SIL Software-In-the-Loop. iii, v, xiv, xv, 29, 58–60, 65, 66, 72, 95, 116, 117, 119–124

TECS Total Energy Control System. 62

UART Universal Asynchronous Receiver/Transmitter. 33, 65, 72, 73

UAV Unmanned Aerial Vehicle. iii, vii, xvii, 1–3, 6, 12, 31, 32, 35, 37, 38, 48–51, 54–63, 65, 66, 69, 71–73, 75, 76, 81, 83, 87, 88, 95, 96, 100, 102, 104, 106, 111, 113, 115–117, 119, 139–141

UDP User Datagram Protocol. 29, 65, 69

XOR Exclusive Or. 19

Chapter 1

Introduction

As an introduction, Section 1.1 presents the background and motivation for this thesis, Section 1.2 presents some previous work of relevance, Section 1.3 presents the contribution and goal of this thesis and Section 1.4 presents its organization.

1.1 Background and Motivation

Unmanned aerial vehicles (UAVs) have for long primarily been used for military purposes, but have in the recent years seen increased civilian use with applications like aerial photography and power line inspections.

UAVs are also predicted to have a significant influence in the maritime sector, with possible future applications including search and rescue missions, ice management in the Arctic, environmental monitoring of oil spills and maritime traffic monitoring.

Operating fixed-wing UAVs from ships is considerably more challenging compared to land based operations. Launching UAVs from maritime platforms can be done with catapults, like those of an aircraft carrier. However, landing on a ship without a flight deck poses several problems. Firstly, the UAV must be brought to a halt in a confined space without damaging the UAV. Most UAV landing methods on ships involves either net or cable-gripping systems, that catches and slows down the UAV allowing it to be recovered undamaged. Secondly, the ship is a moving platform affected by environmental forces like wind, waves and current. Successfully landing a UAV in such a challenging environment requires a high-precision navigation system and the ability to deal with possibly large and fast ship movements.

Although Global Navigation Satellite System (GNSS), e.g. Global Positioning System (GPS), is widely used for navigation, it lacks the accuracy needed for shipboard landings, especially when it comes to vertical positioning. Errors of just over a meter may cause the UAV to miss and crash into the ship or water.

This thesis considers the problem of designing an automatic net-landing system using Real-Time Kinematic GPS (RTK-GPS) to provide positioning with centimeter accuracy. To deal with ship motions, an artificial intelligence (AI) approach to ship motion prediction is designed using artificial neural networks (ANNs).

1.2 Previous Work

An autonomous landing can be divided into the two sub problems path planning and guidance, where each field have been subject to extensive research. This is also the case for forecasting, with both AI methods and statistical approaches being researched.

A commercially available autonomous landing system is the ScanEagle UAV and SkyHook recovery system [31]. It features fully autonomous aircraft approach and capture using RTK-GPS for consistently accurate positioning. The downside of this system is that it is closed source with a small selection of UAVs at a high price.

As part of a NTNU MSc thesis from 2014, Skulstad and Syversen [73] successfully performed net landing of a fixed-wing UAV using a low-cost RTK-GPS system. The real life test was with a static net, while additional simulations were performed for a target moving with a constant heading. A decoupled guidance system was used, consisting of two Proportional-Integral-Derivative (PID) controllers for longitudinal guidance and a nonlinear controller for lateral guidance.

Autonomous landing of a fixed-wing UAV using Differential GPS (DGPS) was tested by Smith [74]. A model of the aircraft was used to create decoupled linear controllers. An inner loop controlled the high-bandwidth specific accelerations and roll rate, while an outer loop controlled the low-bandwidth point-mass dynamics.

By integrating a vision system into a UAV platform already consisting of various avionic sensors and a flight control system, Kim et al. [40] managed to safely land a small fixed-wing UAV into a net. This visual-based net-recovery system does not require a clean runway, complex instrumented landing systems or expensive sensors such as a DGPS.

A spiral landing path and associated guidance laws for UAV net-recovery is proposed by Yoon et al. [88]. Landing is divided into the two following phases: a spiral descent phase to reach both an altitude and horizontal position sufficiently close to the net, and a final approach (FA) phase leading into the net.

The heave motion of a ship was predicted in [84] and for up to 50 seconds in [52] by using ANNs. Similarly, the roll motion was predicted in all of [34–36, 58, 86] for up to 10 seconds, while [57] predicted the yaw motion for 3 seconds. [83] used ANN to predict the ship trajectory. In all cases the prediction method only required time series of the ship motion, and without the need for a model of the ship.

1.3 Contribution and Goal of the Thesis

The goal of this project is to design and verify the performance of an automatic landing system for a fixed-wing UAV to be used with a net on board a moving ship. This involves:

- Developing a modular payload for the X8 Skywalker UAV using the LSTS software toolchain to command either waypoints or bank angle and climb rate to the Pixhawk autopilot (Chapter 4).
- Preparing the payload module for integration with RTK-GPS for accurate navigation (Section 4.6).
- Developing a feasible and dynamic flight path for automatic landing (Section 4.3).
- Developing a system for ship motion prediction to ensure a safe landing on a moving ship (Section 4.2).
- Performing ship motion prediction on simulated ships (Section 5.1).
- Performing software-in-the-loop simulations for the guidance and control system developed (Section 5.2).

1.4 Organization of the Thesis

This thesis is organized in six chapters and three appendices. The chapters are indexed 1-6 and the appendices are indexed A-C. A short description of the chapters and appendices are given below.

Chapters

- Chapter 2 contains a general description of the path planning and guidance problems that are discussed in this thesis, in addition to an introduction to artificial neural networks and time series forecasting.
- Chapter 3 describes the software and hardware components of the automatic landing system.
- Chapter 4 describes the overall system architecture as well as the design decisions and setup of all parts of the automatic landing system and the later experiments.
- Chapter 5 presents the experiments performed and results obtained through simulations.
- Chapter 6 summarizes the conclusions drawn from the results and provides recommendations for further work.

Appendices

- Appendix A contains the preliminary experiments and tuning values for the guidance and control system.
- Appendix B contains supplementary information and results for the time series forecasting experiments.
- Appendix C contains supplementary information and results for the software-in-the-loop landing simulations.

Chapter 2

Background Theory

This chapter contains a brief introduction to the theory behind path planning in Section 2.1, guidance in Section 2.2, artificial neural networks in Section 2.3 and time series forecasting in Section 2.4. It is the basis for the discussions in the rest of this thesis.

2.1 Path Planning

Path planning is the task of finding a way, or path, to go from location A to B. This path is optimized with respect to one or more criteria such as distance, travel time, energy consumption, simplicity etc. As opposed to motion planning and trajectory planning, path planning results in a purely geometrical path and does not include a time law for the vehicle to follow [41]. Some literature differentiate *path planning* as the task of finding waypoints and *path generation* as the task of making a path from those waypoints [70], but path planning will be used as a collective term henceforth.

An essential division when performing path planning is regarding the presence of obstacles. When obstacles need to be taken into account, the path planning problem can be very complex. Especially when working with a dynamic environment. Safety regions around the obstacles need to be defined, both because of the size of the moving vehicle and due to uncertainty in the movements. Planning the optimal path with obstacles present is a Nondeterministic Polynomial time (NP)-complete problem [14].

Another division of path planning is between *deliberative* path planning, when global world knowledge is assumed, and *reactive* path planning, when local sensor information is used [7]. When there are no obstacles to take into account, or the positions of all obstacles are known prior to execution, deliberative path planning can be used.

After a path is planned, the guidance controller assures the vehicle follows the path. To make this task easier, it is desirable to make a *feasible path*, a path for which there exists a sequence of inputs that executes the path. This means the path needs to be continuous, and in most cases also in its first and second derivative ($\mathbb{C}^2$ continuity) [80]. This is to avoid instantaneous changes in position/angle, speed/rotation or acceleration/torsion, which may not be physically possible to execute.

The path should also satisfy the dynamic and kinematic constraints, such as minimum curvature, minimum torsion, maximum climb/dive angle and saturation of every individual actuator. For a UAV, this is called a *flyable path* [80]. This is especially important in the presence of obstacles to ensure a collision free travel. The rest of this section is devoted to exploring the most common path planning methods.

2.1.1 Piece-Wise Continuous Interpolation

The most basic way of generating a path, and probably the most widely used, is to plot waypoints and draw straight lines between them [24]. A lot of care needs to be taken on the placement of the waypoints in the presence of obstacles. These kind of paths work well for vehicles with medium or low demand on precision, and a large variety guidance systems have been created to deal with them. A big problem is that the path is not feasible or flyable, as it at every waypoint has an instantaneous change of direction when going from one straight line to another.

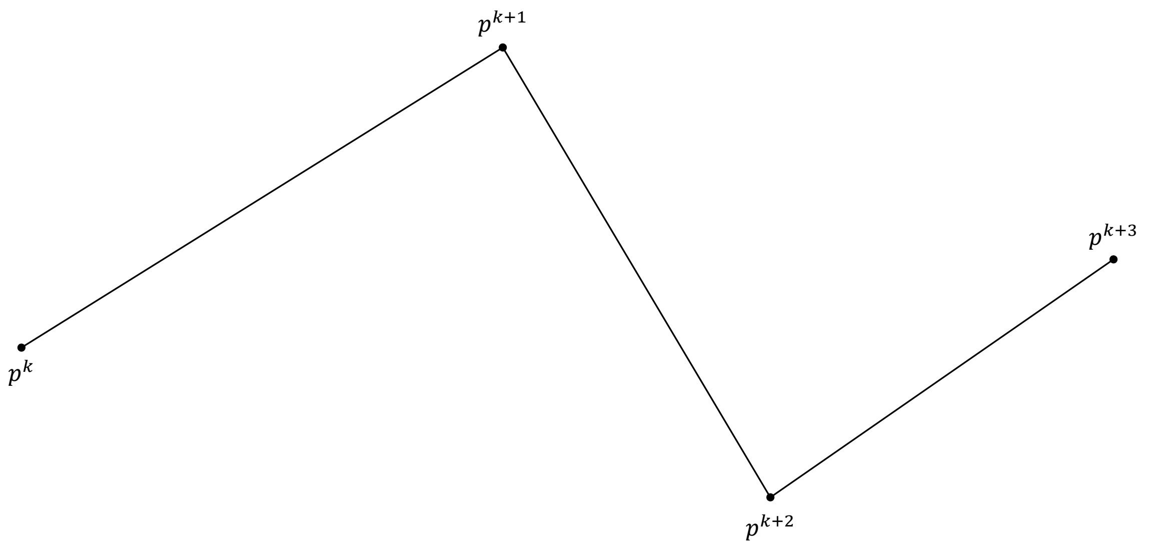


Figure 2.1: Piece-wise continuous interpolated path

2.1.2 Dubins Path

A Dubins path is the shortest path connecting two poses, when a maximum curvature bound is considered. This is proven mathematically by Lester Eli Dubins in 1957 [21]. The path consists exclusively of linear segments and constant-curvature segments. A problem with this path is that, even though there is a smooth change of direction around the waypoints, it is still not entirely feasible for most vehicles. This is because it includes curvature discontinuities in the transition between the arc and the straight line. One solution to this problem is to include clothoids.

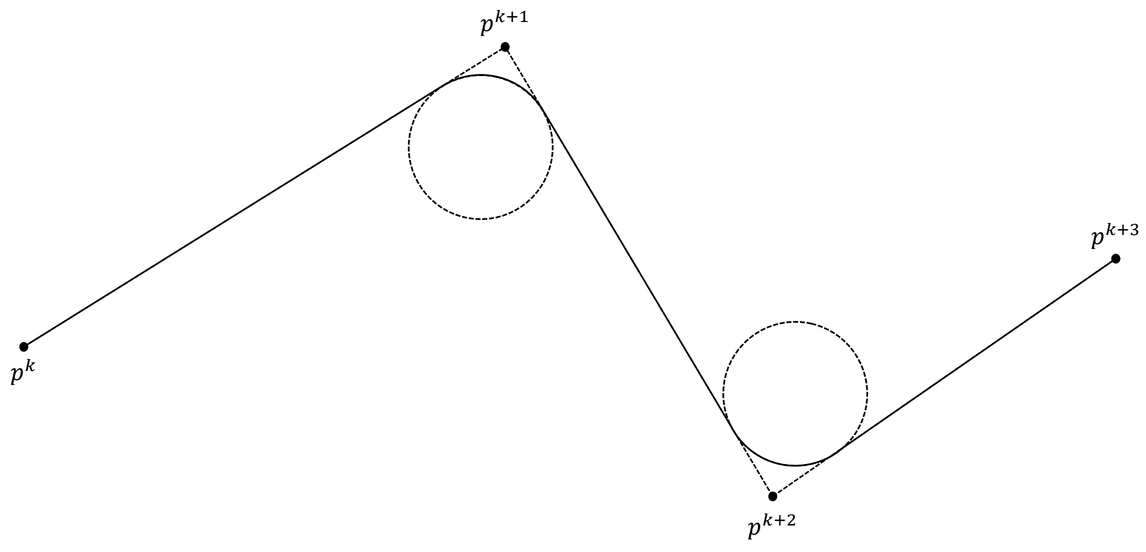


Figure 2.2: Dubins path

2.1.3 Clothoids

For a clothoid arc there are no curvature discontinuities [80]. It has a spiral shape with linearly increasing curvature. This property can be used in a variation of Dubins path where it either replaces the circular arcs or works as a transition between the arcs and the straight lines. A downside of using clothoids is that it has no analytic solution and needs to be calculated numerically.

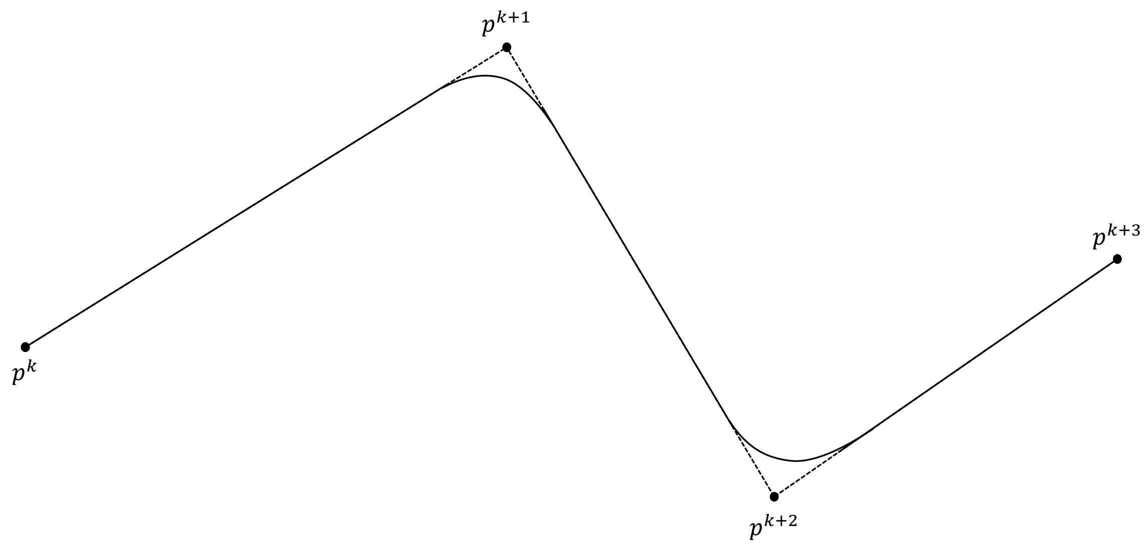


Figure 2.3: Clothoid path

2.1.4 Fermat's Spiral Path

Fermat's spiral is also known as a parabolic spiral and follows the equation $r^2 = k^2\theta$ in polar coordinates [42]. One advantage of using this kind of spiral for path planning is that the curvature is zero at the origin, and thus can seamlessly be connected to a straight line. Using this property, the discontinuity of Dubins path is avoided. Another advantage is that it is described by a very simple parametric equations. It needs to be computed numerically, but is much less computationally demanding than clothoids. In appearance, a Fermat's spiral path often looks very much like the clothoid path (Figure 2.3).

2.1.5 Monotone Cubic Hermite Spline Interpolation

Cubic Hermite spline interpolation is a spline with piecewise cubic polynomials that interpolates the given waypoints [26, 43]. Including the given waypoint locations, it also needs to know the first derivatives at these points. These values may be estimated if not provided. The result is a smooth continuous function with a continuous first derivative. One downside of this method is that it will not have a continuous second derivative.

Monotone cubic Hermite spline interpolation is a method to ensure monotonicity of the resulting Hermite spline. By knowingly selecting the interpolating tangents, overshoot is prevented, meaning that the monotonicity of the path is not violated. This is of special interest when there are obstacles present, and an overshoot can have fatal consequences as objects may be hit.

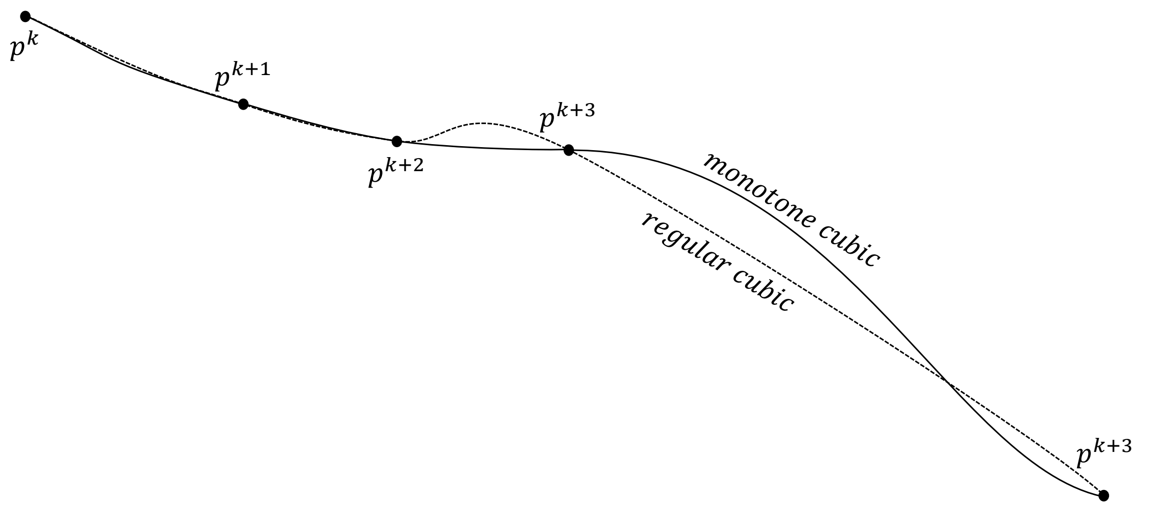


Figure 2.4: Monotone cubic and regular cubic path

2.2 Guidance

Guidance can be seen as the process of guiding an object towards a given point that may be stationary or moving [85]. Guidance is described by Breivik and Fossen as the basic methodology concerned with the transient motion behavior associated with the achievement of motion control objectives [11].

It is clear that the guidance of a vehicle is closely connected to the control system. Fossen [24] describes the different motion control objectives related to guidance as

- *Set-point regulation* is a special case where desired position and attitude are chosen to be constant.
- *Trajectory-tracking*, where the objective is to force the system output $y(t) \in \Re^m$ to track a desired output $y_d(t) \in \Re^m$. Feasible trajectories can be generated in the presence of both spatial and temporal constraints.
- *Path following* is following a predefined path independent of time. No restrictions are placed on the temporal propagation along the path. Spatial constraints can, however, be added to represent obstacles and other positional constraints if they are known in advance.

Thus, the purpose of a guidance system, seen from a control point of view, is to generate the control commands necessary to achieve said control objectives. In fact, Yanushevsky defines the guidance law of a system as an algorithm that determines the required commanded acceleration [85]. The rest of this section will be devoted to exploring the most common guidance strategies related to the path following control objective, based on Siouris' book *Missile Guidance and Control System* [72].

As mentioned above, the guidance system and the control system are closely connected and often implemented together in a guidance and control system. There are two different approaches in designing such a system. The first separates the vehicle guidance and control problem into two separate loops. The inner control loop follows acceleration commands generated by the outer guidance loop. The second is an integrated approach where the inner and outer loops are designed simultaneously. This method allows the use of control design techniques like receding horizon [67] and sliding mode [81] controllers.

The first approach is commonly favored due to its simplicity and the fact that it utilizes the well-established classical control design techniques for the inner loop. The guidance strategies used in the outer loop are heavily influenced by the guidance laws used to control missiles. According to Park et al. [62], several of the missile guidance laws may be modified to perform trajectory/path following by using an imaginary point moving along the desired flight path as a pseudo target. The majority of these guidance laws are of the Line-of-Sight (LOS)-type, i.e. the LOS vector and its rates are the primary source of guidance information. A few of the most relevant strategies will be briefly outlined in the following subsections.

2.2.1 Beam-Rider

Beam-rider is a three point guidance strategy involving a stationary reference point together with the interceptor and target. The vehicle is constrained to follow the straight line between the reference point and the target. This makes it suitable for following paths defined by straight lines between waypoints. A lookahead based approach is given by Fossen [24], where the desired course angle is given by

$$\chi_d = \alpha_k + arctan\left(\frac{-e}{\Delta}\right) \tag{2.1}$$

where α_k is the path tangential angle, Δ is the lookahead distance along the path and e is the cross-track error, seen in Figure 2.5.

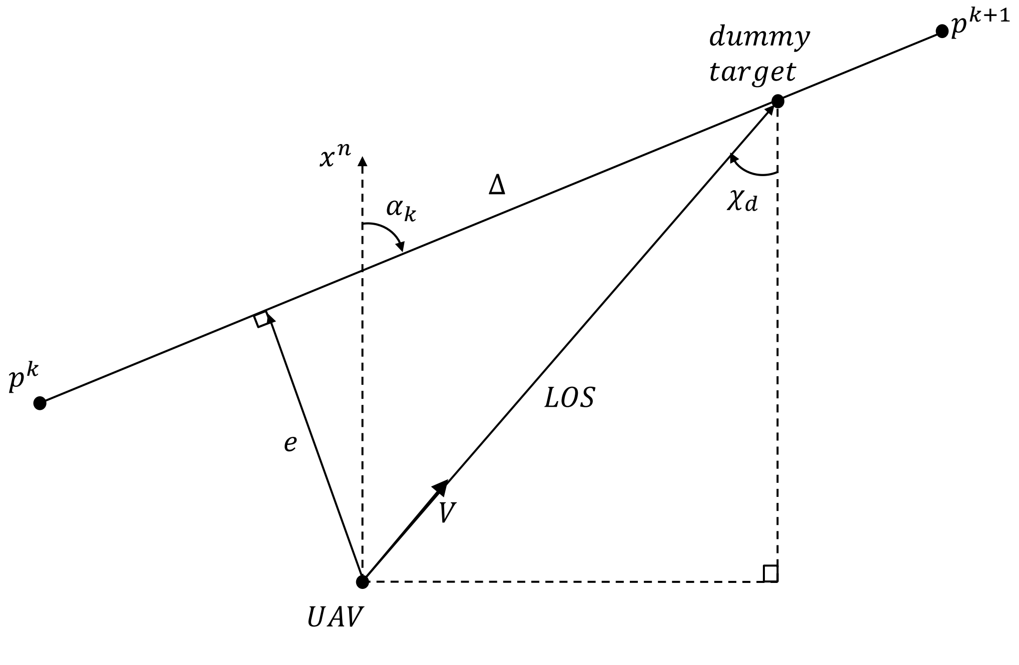


Figure 2.5: Beam-rider guidance

2.2.2 Pure-Pursuit

In pure-pursuit guidance the velocity of the vehicle is aligned with the LOS vector, i.e. the vehicle will steer directly towards the target at all times, seen in Figure 2.6. Fossen [24] performs the velocity assignment by defining it as

$$V = -\kappa \frac{p - p_t}{||p - p_t||} \tag{2.2}$$

where p is the vehicle position, p_t is the target position and $\kappa > 0$ is a design parameter.

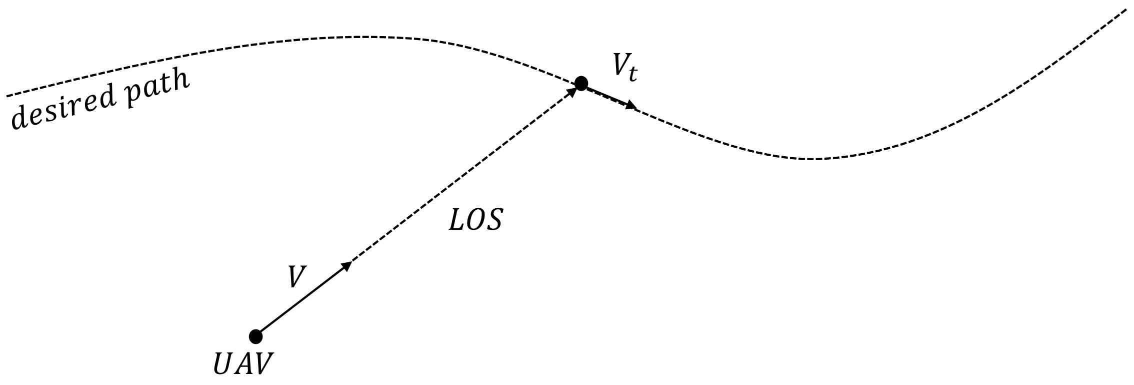


Figure 2.6: Pure-pursuit guidance

2.2.3 Constant Bearing

Constant bearing guidance involves keeping the LOS rate at zero. This leads to the LOS vector being parallel to the initial LOS vector, seen in Figure 2.7. Constant bearing guidance is therefore often referred to as parallel navigation.

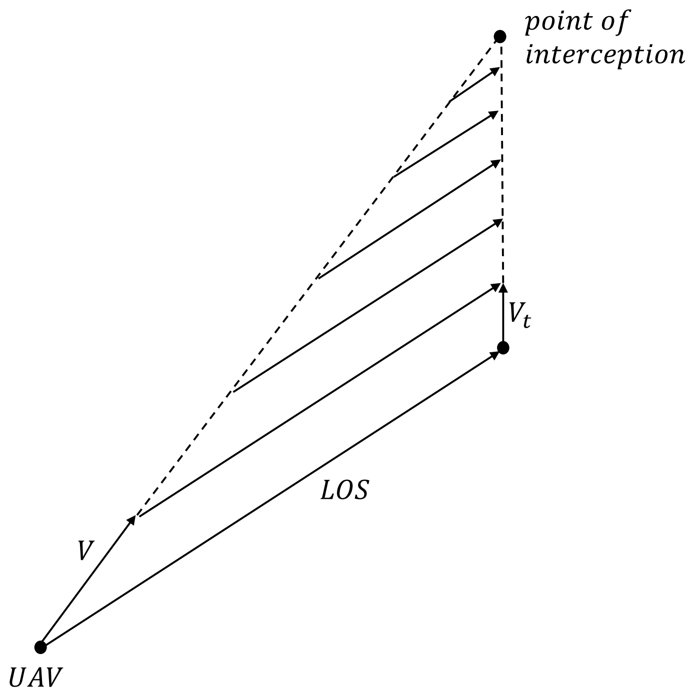


Figure 2.7: Constant Bearing

Constant bearing is usually implemented by rotating the velocity vector proportional to the rate of the LOS vector. This is known as proportional navigation which will be described in the next section.

2.2.4 Proportional Navigation

The by far most popular guidance law is *Proportional Navigation (PN)* [72, 85]. It involves measuring the rate of rotation of the line of sight angle with respect to fixed space coordinates, and commands a lateral acceleration proportional to that of the LOS rate, seen in Figure 2.8. This can be expressed mathematically as

$$a_n = NV_c\left(\frac{d\lambda}{dt}\right) \tag{2.3}$$

where

$$\begin{aligned} a_n &= \text{commanded lateral acceleration} \\ N &= \text{navigation constant} \\ V_c &= \text{closing velocity} \\ \frac{d\lambda}{dt} &= \text{LOS rate of change} \end{aligned}$$

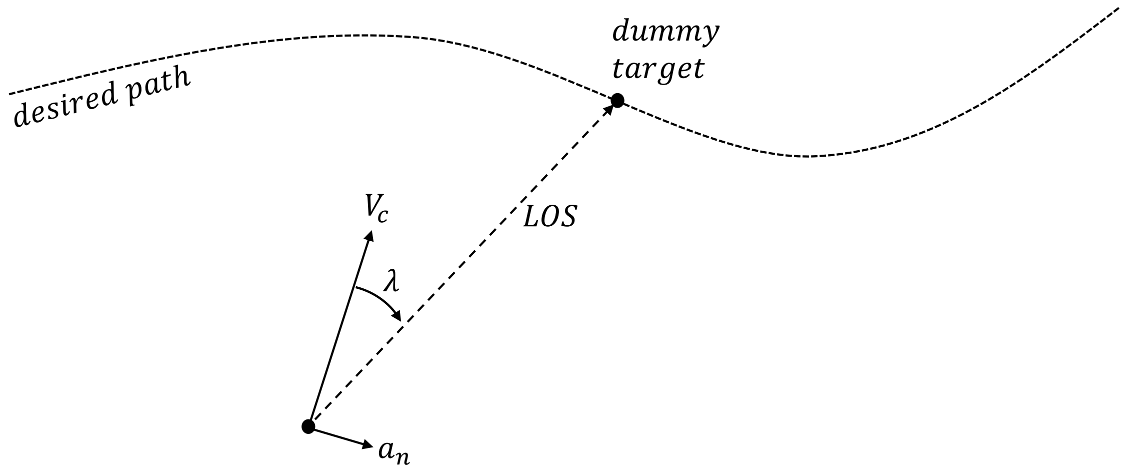


Figure 2.8: Proportional Navigation

When dealing with vehicles in all three dimensions (3D), like for UAVs, it is common to decouple the longitudinal and lateral motions. As a consequence, PN is often implemented as two separate 2D-guidance loops. However, the PN guidance problem has also been formulated as a 3D control problem [7, 72, 85].

2.2.5 Proportional-Integral-Derivative Controller

The proportional-integral-derivative (PID) controller is often used in industrial systems, but can also be used for guidance, as described by Beard and McLain [7]. By using loop feedback from the difference between a desired and a measured value, an input either directly into an actuator or to a second controller can be computed to reduce the error. The value controlled can for example be the distance to a path, and the output can be the vehicles orientation towards it. This control law can be expressed mathematically as

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{d}{dt} e(t) \tag{2.4}$$

where

$$\begin{aligned} u(t) &= \text{commanded input} \\ K_p &= \text{proportional gain} \\ K_i &= \text{integral gain} \\ K_d &= \text{derivative gain} \\ e(t) &= \text{error (desired - measured value)} \\ t &= \text{time} \end{aligned}$$

The proportional part is there to adjust the output depending on the magnitude of the present error. The integral part is the memory term adjusting the output based on the accumulation of past errors, e.g. it increases if the error is positive persistently. The derivative part is predicting future errors and dampens the output.

Integral windup is a situation where the PID controller experiences a large change in the set-point, making the integral term accumulate a significant error while the set-point is being reached (windup). Once the new set-point is reached, the huge integral term leads to an excess overshoot while the accumulated error unwinds. Several solutions to this problem has been proposed, called *anti-windup.*

A simple anti-windup solution is to limit the value of the integral term. If the saturation of the actuator is known, the integral part of the PID controller can be limited to this value. Another option is to reset the integral part to zero when the PD-effect alone is enough to saturate the output.

2.3 Artificial Neural Network (ANN)

This section introduces the artificial intelligence (AI) class of systems called artificial neural network (ANN), some of its varieties and a few applications.

2.3.1 Overview of ANNs

An artificial neural network (ANN) is a learning algorithm inspired by the biological neural network, like the one found in the human brain. By giving it a series of inputs where the output is known, it can learn complex input-output relationships. It is a highly parallel system consisting of interconnected simple processing units, called nodes or neurons. Each neuron receives a series of weighted inputs, sums them up, processes this resulting value through an activation function and sends out a single transformed output. There is connectivity between the individual neurons in a network, in addition to external inputs and outputs connected to this network.

An external input can for example be pixels of a facial image, and the output, after the input has passed through the entire ANN, can be activation of an output neuron representing a specific person. This is called classification. The network can also be used for function approximation on the general vector form $\vec{\hat{y}} = \hat{f}(\vec{x})$, where the input to the ANN is $\vec{x}$, and $\hat{f}$ is an approximation of the real function f, giving the network output $\vec{\hat{y}}$.

A human brain has in the order of 10^{10} neurons and about 60×10^{12} interconnections. A single neuron in the brain is measured to operate at 10^{-3} sec [78], compared to a silicon gate operating at 10^{-9} sec. Because of the extreme high degree of parallelism in the brain, it still outperforms the fastest computers in many tasks like pattern recognition and perception.

McCulloch and Pitts were in 1943 the first to propose an artificial neuron model [54]. They based it on summation and a binary threshold activation function. This special case of artificial neurons with a threshold activation function is what Rosenblatt in 1957 called a *perceptron* [65]. More on activation functions later. Mathematically, the output of an artificial neuron is given as [36]:

$$out = f(net) = f\left(\sum_{i=0}^{n-1} I_i w_i + w_n\right) \tag{2.5}$$

where $\{I_i | i = 0, ..., n-1\}$ is the input to the neuron, $\{w_i | i = 0, ..., n-1\}$ is the weight associated with its respective input I_i, and w_n is a bias weight. $f()$ is the activation function. Equation 2.5 is illustrated in Figure 2.9 below. The bias neuron itself is usually a constant, often set to 1, letting its associated weight adjust the bias entering the summation.

When building an artificial neural network, several neurons are combined into layers. There is always an input layer with the same number of neurons as inputs, an output layer with the same number of neurons as outputs and in between those there can be any number of additional layers, called hidden layers. The neurons in these hidden layers are called hidden neurons.

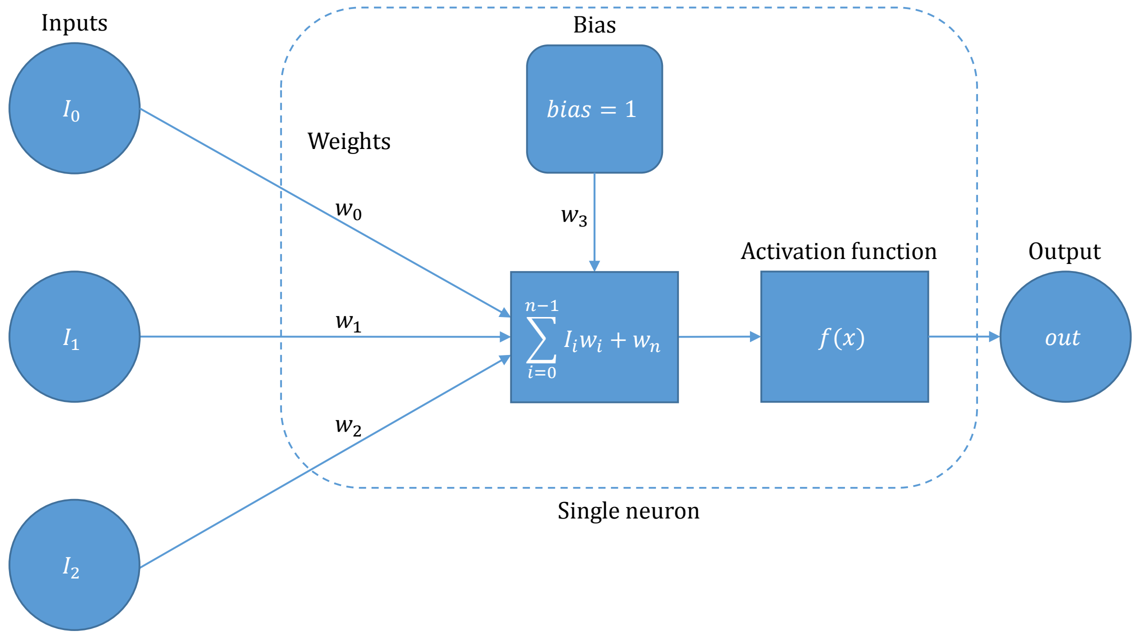


Figure 2.9: A single artificial neuron with three inputs

The structure of the neural network makes it hard to get an intuitive understanding of how it works, and it cannot be used directly to say anything about the model parameters of the approximated function. In fact, there does not seem to be any strict mathematical verification of all the neural networks capabilities [35]. Despite this, a mathematical demonstration of ANN as a universal approximator is presented by Funahashi [27], but is considered outside the scope of this thesis. A trained ANN is often regarded as a "black box", although some people work on finding meanings in the different network weights [46].

One of the great strengths of neural networks is its generalization and approximation capabilities. This means, despite both noisy input and target values, a well designed and properly trained neural network is capable of finding a good approximation of the underlying function for the training data.

On the other hand, since the neural network does not find a model of the actual function, merely an approximation of the training data, it has problems with extrapolation. This means an ANN often performs poorly on test data beyond the range of its training data, emphasizing the importance of a properly trained network. Training on a data set from a too specific range can lead to extrapolation, while a too wide range may lead to an overly generalized approximation that fails to capture important details in the data.

Categorization of ANNs can be defined by four parameters, which will each be discussed below:

1. The interconnection pattern between the different layers and neurons

2. The activation functions of the neurons
3. The number of hidden layers
4. The learning process of updating the weights

2.3.1.1 Interconnection Pattern

The most common interconnection pattern is a feed forward network structure where the connections between the neurons do not form a directed cycle, like the example depicted in Figure 2.10 below. In the opposite case, a network structure with cycles is called a recurrent network. Depending on the degree of connectivity between the layers, the network may also be classified as either fully or sparsely connected.

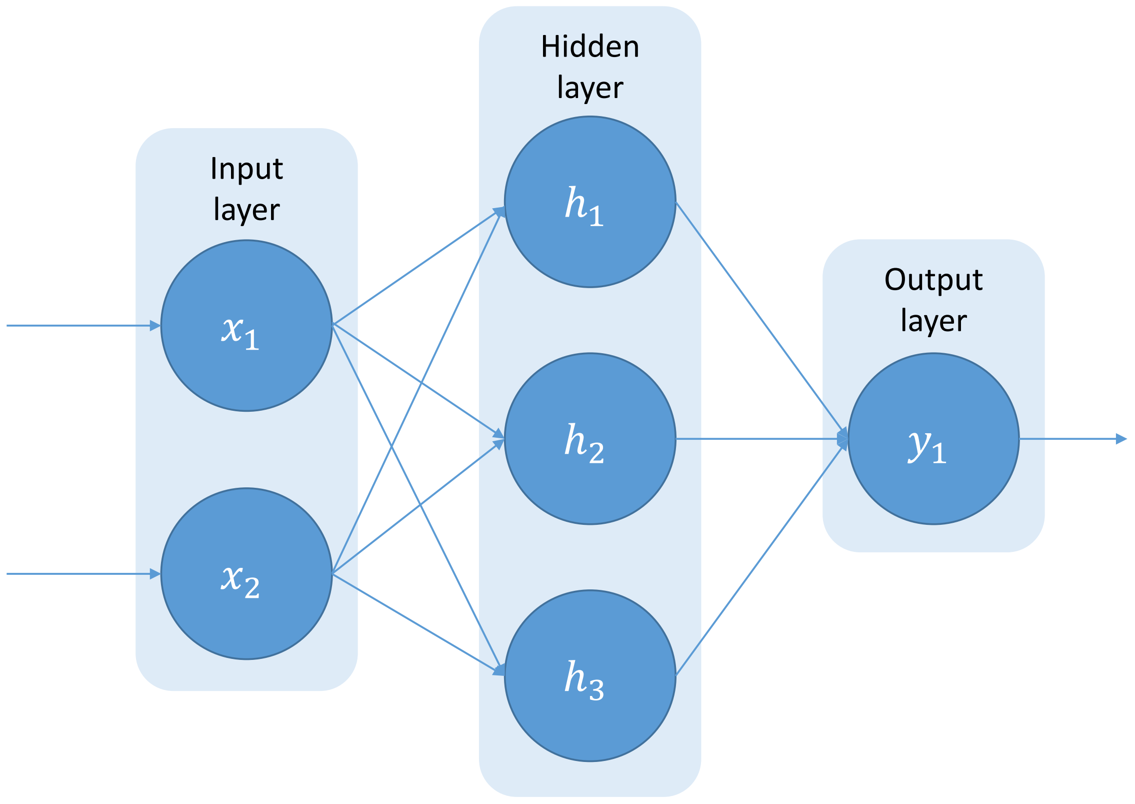


Figure 2.10: A three layer feed forward ANN with two inputs, three hidden neurons and one output. A circle represents an artificial neuron and an arrow a direct connection

2.3.1.2 Activation Function

The activation functions are important choices depending on the application, like function approximation or classification, and the training method. All the neurons of a network may have individual activation functions, but it's rare to have different activation functions within the same layer. Below follows brief descriptions of some commonly used function types.

Binary threshold function

As previously discussed, the first neuron model used a binary threshold activation function [54], as seen in Figure 2.11. This means the output can only be one of two values, usually $\{0, 1\}$ or $\{-1, 1\}$. When performing classification this is a desired property, but it has limitations with regards to training. Because it can not be differentiated, training methods like the later discussed back-propagation algorithm will fail.

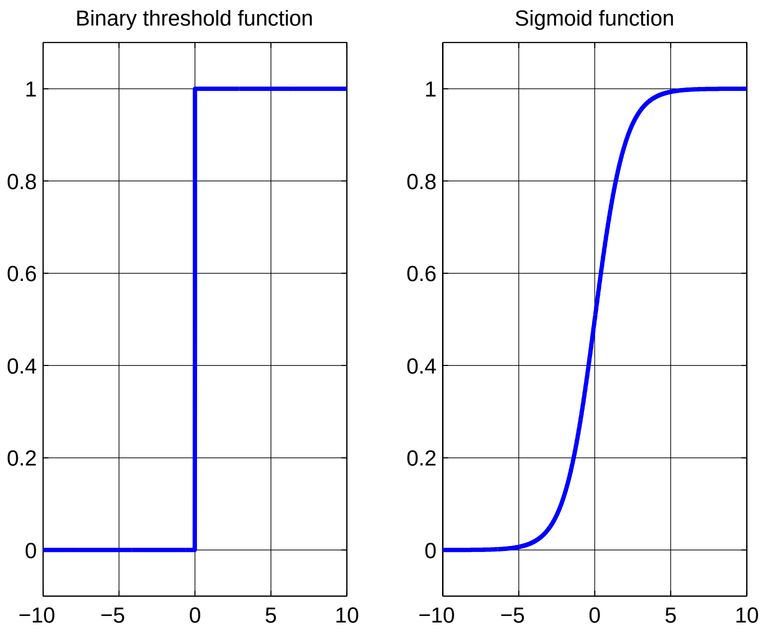


Figure 2.11: Two activation functions: binary threshold and sigmoid

Sigmoid function

The most commonly used activation functions in ANN applications are varieties of the sigmoid function. An example of such a function is seen depicted in Figure 2.11. The steepness of this function can be varied, thus it can be adjusted to resemble a binary threshold function by setting the steepness factor high enough. One great advantage with sigmoid functions is that they are often easy to differentiate. A commonly used sigmoid function is the hyperbolic tangent, transforming neuron Equation 2.5 into:

$$out = \tanh\left(\sum_{i=0}^{n-1} I_i w_i + w_n\right) \tag{2.6}$$

Linear function

When an ANN is used for function approximation, and not classification, the output needs to be continuous and able to take any value. A linear activation function, as seen in Figure 2.12, provides this by not truncating what's summed up in the neuron. A linear activation function is thus often used in the output layer when performing function approximation.

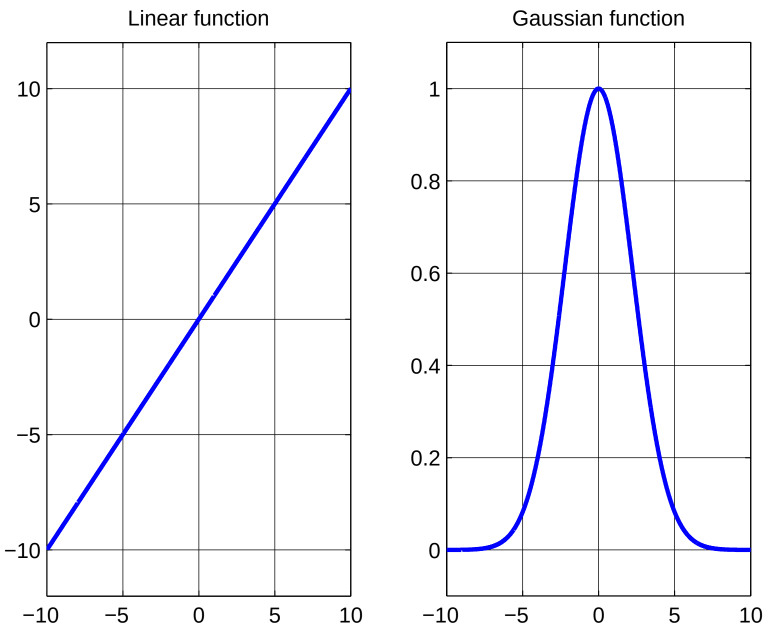


Figure 2.12: Two activation functions: linear and Gaussian

Gaussian function

The Gaussian function has a bell shape, seen in Figure 2.12. This is the most popular and widely used function by radial basis function (RBF) networks [68], which is a variety of single-layer ANN first formulated by Broomhead and Lowe in 1988 [12].

2.3.1.3 Number of Hidden Layers

A feed forward network with no hidden layers is called a single-layer perceptron [65]. Minsky and Papert famously demonstrated in a book published in 1969 how this network is unable to learn an XOR function [55]. The single-layer perceptron will only be able to perform a linear operation on the input data.

To handle all non-linear problems, one or more hidden layers are needed in addition to a non-linear activation function. A feed forward network with one or more hidden layers is called a multi-layer perceptron (MLP). Although it's called a multi-layer *perceptron*, the neurons may have other activation functions than the threshold function used in the strict definition of a perceptron. In 1989 Cybenko published an article on function approximation with the use of neural networks and sigmoidal activation functions [18]. He there proved what's known as Cybenko's theorem:

> *A MLP with only one hidden layer is capable of approximating any function, provided there are enough number of hidden neurons whose activation functions are non-linear.*

While this theorem states that only one hidden layer is enough to approximate any function, there may be reasons to have additional hidden layers. Experienced designers may adjust the number of hidden layers and neurons to reduce the training time, as the computational burden is dependent on the number of neuron interconnections. The network structure may also be adjusted to exploit a partially known input-output relationship.

2.3.1.4 Training

Training of a neural network can be seen as a minimization problem where the weights of the network are adjusted to minimize the error between the output of the ANN and the desired output (target). As with most optimization tasks, an ANN may contain several minima, thus obtaining the global minimum is often very time consuming and practically impossible. Luckily, finding the global minimum of the network's error function is in many cases not needed or even desired. This is due to the problem of over-fitting.

Over-fitting is when the network approximates the learning data too perfectly and fails to generalize, leading to wrong outputs when presented with inputs different from the training data. In the opposite case, under-fitting is when the network generalizes too much and fails to capture important details in the underlying function. This trade-off makes it hard to know exactly when to stop the training process. One common trick is to validate the accuracy of the trained network on a separate data set, a validation set independent of the training set. This way training can halt when generalization stops improving. A flow chart of this training process is presented in Figure 2.13 below (from Khan et al. [35]). Below follows only a selected few of the most widely used training methods.

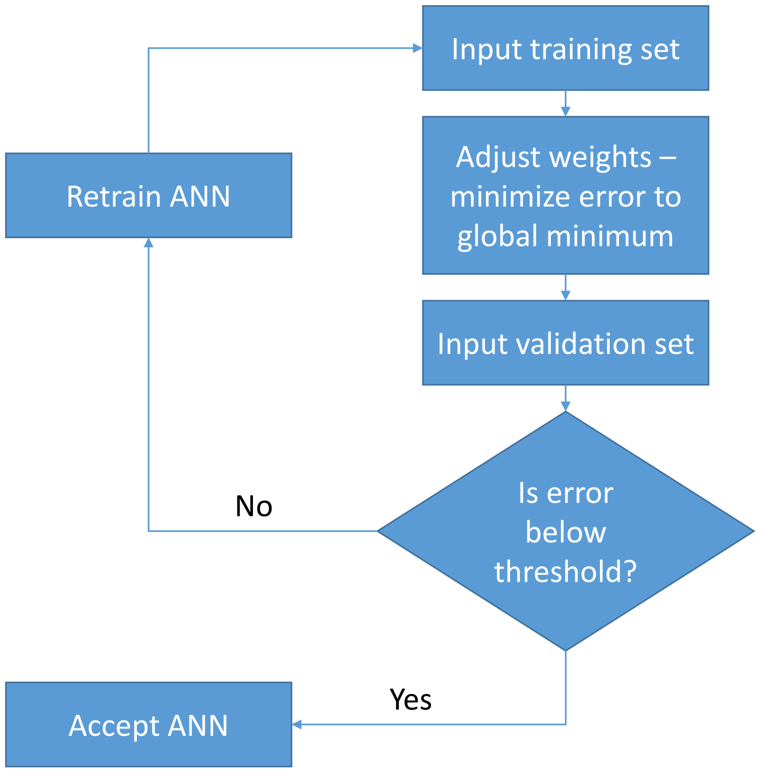

Input training set
Adjust weights – minimize error to global minimum
Input validation set
Is error below threshold?
No
Retrain ANN
Yes
Accept ANN

Figure 2.13: Simplified flow chart of a general training process for ANN, from [35]

Back-propagation

The back-propagation (BP) algorithm was originally discovered by Werbos in his PhD thesis in 1974 [82], but not made famous until it was rediscovered in 1986 by Rummelhart et al. [66]. Although several other algorithms for neural network training have come along since then, it remains a widely used and effective training method.

The algorithm is a form of gradient descent where the error between the ANNs output and target value is propagated backwards through the network, from the output of the network to the input layer. Along the way, the weights of each layer is altered in the direction of the negative gradient, the weights of one layer at a time.

As with the regular gradient descent algorithm there is the problem of local minima, in addition to the design choice trade-off of step size, known as learning rate η. A large step size ensures a fast algorithm with the chance of missing the minimum, while a small step size ensures a minimum is found at the expense of computation time. A common technique to compensate for this trade-off is to introduce a momentum term, where the previous search direction is included in the new search direction. This may also help avoid getting trapped in local minima.

Using this algorithm on an ANN requires that the activation functions of the artificial neurons are differentiable. This means that the binary threshold function does not work with this algorithm, while the sigmoid function is a popular choice because of its easy and efficient differentiation.

```
input  : A training set (and optionaly a validation set)
output: A trained ANN

initialize the network weights (to small random values);
while stopping criteria not met do
    // forward propagation
    out = ANN_prediction(input_sample);
    target = ANN_target_value(input_sample);
    error = target - out;

    // backward propagation
    compute $\Delta w_h$ for all weights from hidden layer to output layer;
    // further backward propagation
    compute $\Delta w_i$ for all weights from input layer to hidden layer;

    update all weights based on $\Delta w$;
end
```

Algorithm 1: Back-propagation algorithm for a three-layer MLP, based on [29]

The general function for the back-propagation algorithm outlined in Algorithm 1 can be written as

$$w_{ji}^{(s)}(n+1) = w_{ji}^{(s)}(n) + \eta \delta_j^{(s)}(n) x_{out,i}^{(s-1)}(n) \tag{2.7}$$

where each weight w is described by the neuron it's connected from, neuron i in layer $s-1$, and the neuron it's connected to, j in layer s. η is the learning rate and $\delta_j^{(s)}$ is calculated as a function of the back-propagated error at neuron j in layer s. Further details of the learning algorithm is considered beyond the scope of this thesis, and interested readers are directed to Haykins book on neural networks [29].

Conjugate gradient algorithm

The conjugate gradient (CG) algorithm was developed by Hestenes and Stiefel in 1952 [30] as an iterative algorithm for solving systems of linear equations. It was first used for training of neural networks many decades later, like Møller (1990) [56] and Charalambous (1992) [15].

As the original paper from 1952 demonstrates, CG is able to obtain the exact minimum in n steps if the n-dimensional function can be expressed as quadratic. The error function of ANNs is quadratic close to the minimum, hence it is expected that once close to a minimum, convergence to the local minimum will be very rapid [36].

The fundamentals of the CG algorithm shares commonality with the BP algorithm, but is often regarded as superior. It does not move straight down the new gradient, but the search direction is instead conjugate to the old gradient and to previous directions traversed.

2.3.2 Data Fusion Using ANNs

Because neural networks don't rely on any prior knowledge about neither the underlying function nor the inputs to the network, it can easily be used for data fusion. The simplest approach is to collect all the data that is to be fused into the input vector of the network. During training, the ANN will by itself find the relationship between the different inputs without knowing which data set or sensors the different inputs originate from.

Another approach is feature-based data fusion. Different kinds of data are either separately or in groups sent through different ANNs for feature extraction, before the output of those ANNs are fused in a final network, like the example network depicted in Figure 2.14. A group at Carnegie Mellon University successfully implemented a network like this, called feature neural network (FeNN), for autonomous outdoor navigation with the use of a CCD camera and laser scanner [19].

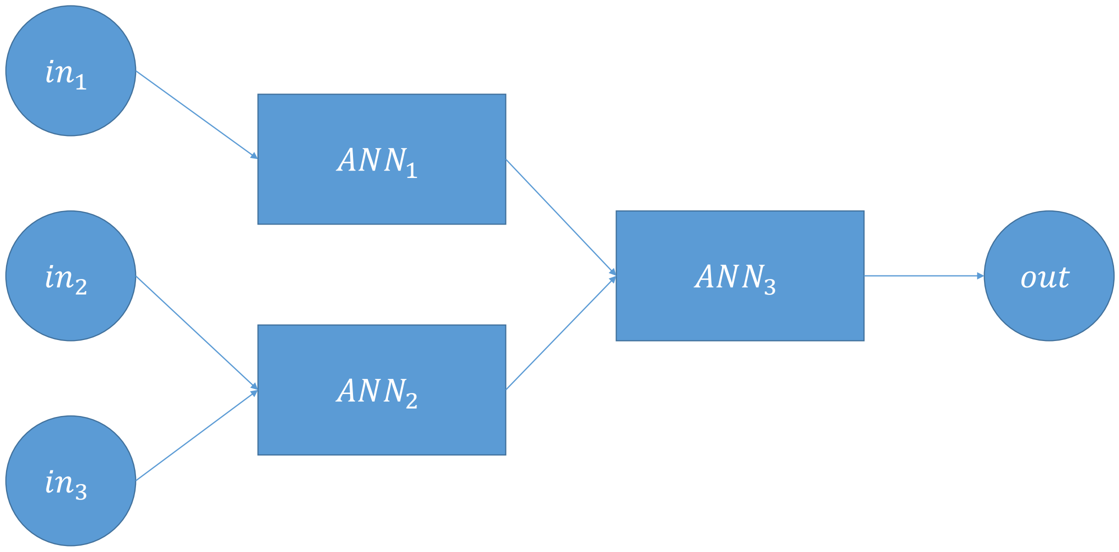


Figure 2.14: Feature-based data fusion using ANNs in cascade

2.4 Time Series Forecasting

Forecasting is the process of predicting the future, while a time series is a sequence of data points collected over a time interval. Khashei and Bijari offers the following definition to time series forecasting:

With the time series approach to forecasting, historical observations of the same variable are analyzed to develop a model describing the underlying relationship. Then the established model is used in order to extrapolate the time series into the future. [38]

This approach is particularly useful when there is too little knowledge of the underlying function to build a reliable model. Time series forecasting is an active research area with a large variety in both forecasting methods and application areas. Below follows a brief introduction to a small selection of the most common methods.

2.4.1 Autoregressive Integrated Moving Average

The autoregressive integrated moving average (ARIMA) model is a statistical method used for time series analysis and forecasting. Box and Jenkins popularized the use of ARIMA in the 1970's [9, 10], and their approach is known as the Box-Jenkins method.

For ARIMA forecasting the predicted value is assumed to be a linear function of past samples and random errors [39]. The assumed linearity is a strong limitation for this method. Compared to non-linear methods like ANN, multiple attempts at time series forecasting have demonstrated that ARIMA is inferior [34, 37, 90].

2.4.2 Kalman Filter

The Kalman filter is a recursive filter able to estimate linear and nonlinear dynamics from noisy measurements [24]. It is based on a model of the dynamics. The first version, a discrete-time linear filter, was proposed by Kalman in 1960 [33].

In its essence the Kalman filter uses the current state estimates and system model to predict the next step, updates the prediction based on measurements, outputs new state estimates and repeats this process recursively. If the filter does not receive new measurements, it will work as a predictor. By rapidly iterate multiple steps forward in time, this can be utilized for forecasting.

When used for ship motion forecasting in high sea states, past attempts [16, 17, 71] have demonstrated that Kalman filters are unable to maintain a high prediction accuracy beyond 3-4 seconds [35].

2.4.3 Artificial Neural Network

ANNs are able to deal with highly non-linear systems without the need of a system model, are noise tolerant, has a natural high degree of parallelism and can adapt to the circumstances. This makes it a good choice for forecasting. Time series forecasting is a special case of the function approximation problem, where the inputs and outputs of the network are samples of the same observed data set [58].

By training the network on a time series the ANN can learn the data patterns to find the continuation of the time series [28]. Figure 2.15 below illustrates such a MLP network designed to output the estimated value at time $k + m$ according to Equation 2.8:

$$\hat{y}(k+m) = f\left(y(k), y(k-1), ..., y(k-n+1)\right) \tag{2.8}$$

where k is the current time, m is the prediction step, n is the number of input samples to the network and $f()$ is the ANN network giving the predicted value $\hat{y}$.

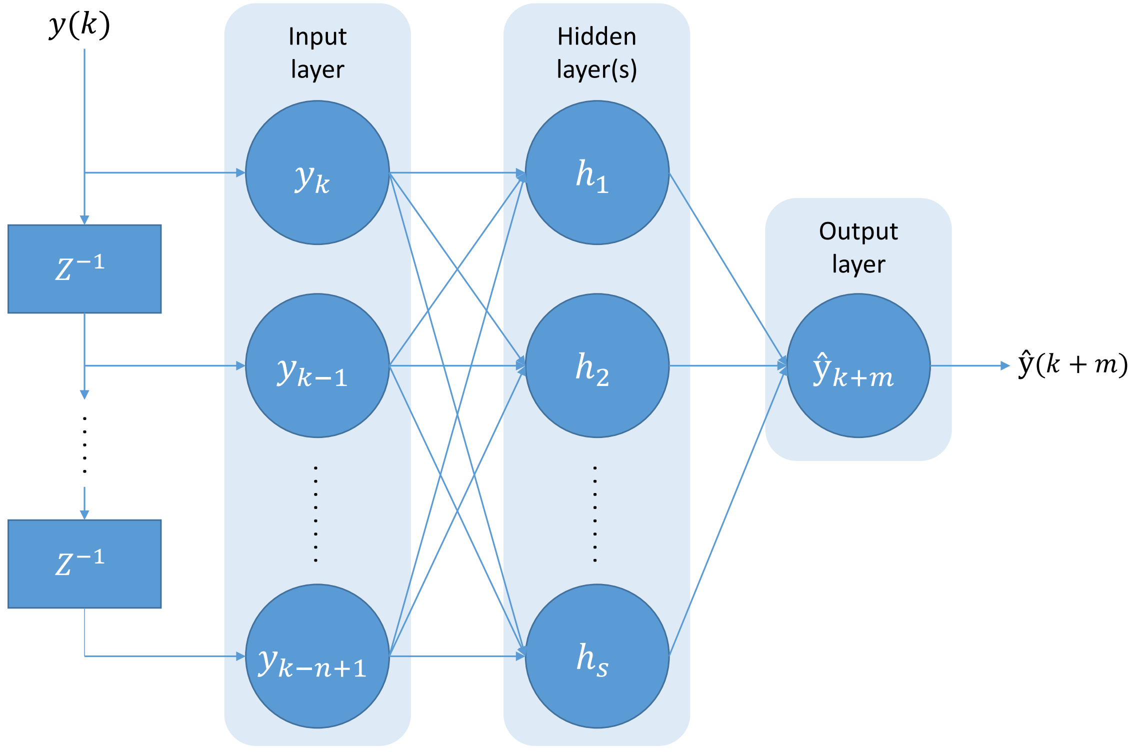


Figure 2.15: Time series forecasting using ANNs

There is only one output in the figure, corresponding to only one prediction step. It is advised to use a single ANN for every prediction step [34]. If predictions at multiple time steps ahead are required, a separate ANN should be used for each time step even though a single ANN could be equipped with multiple outputs. This is based on the reasoning that the optimal weights in all layers vary depending on the desired prediction step. The optimal network architecture may also be different.

It is important to notice that when using ANNs for forecasting, it is not guaranteed that the network will perfectly learn the patterns of the data, even though it theoretically is capable of it. This depends on variables like network architecture, training method and size of the data set used for training. In addition, while an

ANN is good at handling noise, the degree of randomness in the time series affects how correct the predictions will be.

The more complex the time series is, the more information about the past is needed, and the size of the input layer and the corresponding number of weights are increased [28]. The size of the forecasting step, given by the variable m, will also affect the number of input samples n needed. For a more accurate approximation of the time series, the number of hidden nodes, denoted s, may be increased.

However, when the data sets are affected by noise, determination of the number of hidden units becomes more complicated [44]. A too low number of hidden nodes will lead the ANN to fail on approximating the underlying function of the time series, i.e. under-fitting. In the opposite case of over-fitting, a too high number of hidden nodes will result in a very good approximation on the training data, but fail to capture the generality of the time series, hence lead to poor estimations on new test data.

2.4.4 Hybrid Approaches

Several hybrid approaches to time series forecasting has been proposed. It can be homogeneous, e.g. different configurations of ANNs, or heterogeneous, e.g. both linear and nonlinear models, [37]. For the most common forecasting models, none of them are universally superior to the other. This may give hybrid approaches an edge. The disadvantage is that it is much more time consuming to implement as all the models need to be tuned individually before combined together in a hybrid model.

Khashei et al. have proposed three hybrid models for time series forecasting using ARIMA and ANNs [37–39]. The great linear capabilities of ARIMA is utilized together with the nonlinear capabilities of the ANN. Li et al. have presented a wind speed forecasting methodology based on a Bayesian combination algorithm [45]. It combines the three artificial neural network models adaptive linear element network (ADALINE), back-propagation (BP) network (the most common ANN) and radial basis function (RBF) network. This provides an adaptive, reliable and comparatively accurate forecast result.

Chapter 3

System Components

This chapter describes the selection of individual components for the automatic landing system, with software presented in Section 3.1 and hardware presented in Section 3.2.

3.1 Software

This section presents the software components of the automatic landing system.

3.1.1 LSTS Toolchain

This is an open-source control architecture and software toolchain developed by the Underwater Systems and Technology Laboratory (LSTS), which is an interdisciplinary research laboratory established in 1997 at the Faculty of Engineering, University of Porto [47]. The toolchain supports networked heterogeneous air and ocean vehicle systems [64], and is composed of the embedded operating system GLUED, on-board software DUNE, shore-side control software Neptus and IMC, the communication protocol which is shared by all components.

3.1.1.1 GLUED

GNU/Linux Uniform Environment Distribution (GLUED) is a minimal Linux distribution targeted at embedded systems developed by LSTS. GLUED has the following features:

- Cross compilation ready
- Small footprint - 10 MB
- Lightweight
- Fast boot time - 2-5 seconds depending on target system
- Easy to configure
- Portable to Intel x86, ARM and MIPS architectures

GLUED is configured for a target system in a configuration file that is created for each specific system. The configuration file specifies the system architecture, network settings and which packages that will be a part of the target distribution. The standard GLUED distribution contains only the packages necessary to run on an embedded system, adding or removing packages is done through the configuration file.

Finally, the distribution is usually cross-compiled for the target distribution, i.e. building the binaries for a platform other than the one on which the compiler is running. This makes it possible to develop the software systems on powerful desktop computers instead of working on the embedded system itself. This greatly decreases the build time and completely removes the need for boot-loaders and kernel customization.

3.1.1.2 IMC

Inter-Module Communication (IMC) is a communication protocol developed by LSTS to be used in their toolchain. It defines a common control message set understood by all the vehicles, sensors and consoles [49]. The IMC protocol does not impose or assume a specific software architecture for client applications. IMC already contains many different message types, but also has the ability for users to easily add new and customized message types.

3.1.1.3 DUNE

DUNE: Unified Navigation Environment (DUNE) is an open-source on-board software solution for unmanned vehicles [48]. It is a runtime environment used to write generic embedded software in C++, and is responsible not only for every interaction with sensors, payload and actuators, but also for communication, navigation, control, maneuvering, plan execution and vehicle supervision. It is designed to contain multiple relatively small independent tasks running in separate threads of execution. They are all connected to an IMC bus where they can publish and subscribe to messages for communication in between them, using the commands *dispatch* and *consume*, respectively. This leads to a high degree of modularity, where new tasks, like new sensors, actuators or controllers can be easily added, and old tasks can be enabled and disabled freely. Messages passed to and from the bus are specified by the communication protocol IMC. DUNE provides an operating-system and architecture independent platform abstraction layer.

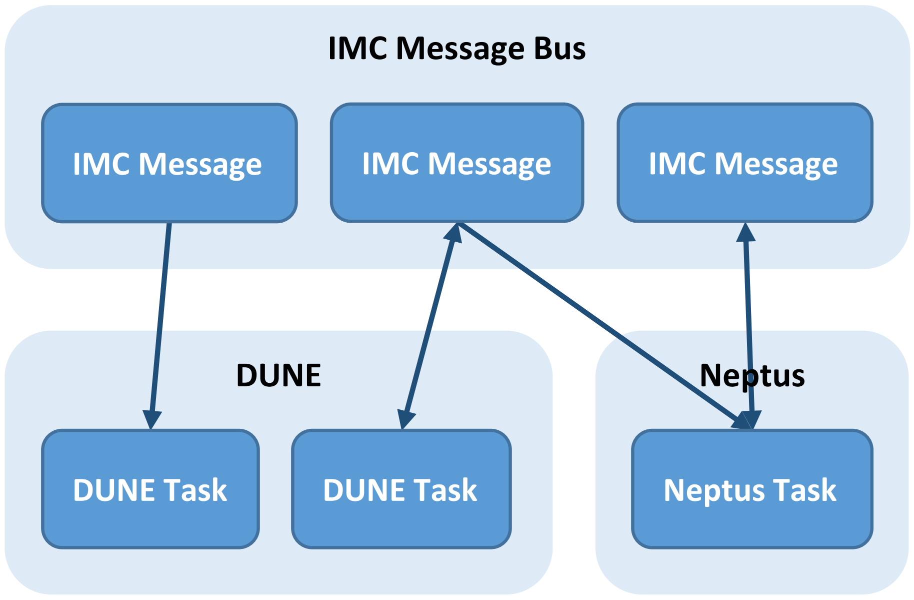


Figure 3.1: IMC message bus serving DUNE and Neptus

3.1.1.4 Neptus

Neptus is an open-source command and control software for a single or fleets of unmanned vehicles and different types of non-actuated sensors [50]. Operators using Neptus can observe real-time data of networked vehicles, review previous missions and plan and simulate future missions. Independently of the main Neptus source code, plug-ins can be developed using the Java programming language. This way new components can be added without affecting the main program or having to share the source code among developers. The main Neptus communication interface is IMC.

A mission in Neptus is specified as a set of maneuvers and transitions between them, forming a graph. This can for example be a set of waypoints, each containing parameters such as three-dimensional position and speed. When connected with a transition between those waypoints, they form a path. Missions can either be sent to Neptus from DUNE using IMC over User Datagram Protocol (UDP), or made within the program itself using the user interface.

Neptus provides three different types of simulations: software-in-the-loop (SIL) simulation, hardware-in-the-loop (HIL) simulation and behavior prediction. SIL simulation is where sensor values and actuations are simulated with the help of one or more simulated vehicles. To perform a test of actual vehicles, real sensors and/or actuators, HIL simulation may be used. While a mission is being executed, Neptus provides a rough behavior simulation of the vehicles currently disconnected from the base station. Their states are predicted to aid the operator, especially in fleets of vehicles with complex behavior.

Mission execution in Neptus' consoles enables operators to monitor vehicle execution, change or create new plan specifications, send plans for execution, operate vehicles, etc. The user interface can be customized according to type or phase of mission, type or number of vehicles, or other operator demands. By displaying only the information the operator needs at every phase of the mission, it makes executing even complex missions fairly simple.

After a mission is completed, it can be reviewed with Neptus Mission Review and Analysis (MRA). Vehicles store mission data of both generated and received IMC messages. Neptus MRA provides several visualizations and utilities to process and export data, and decompress data into text files for later importation to programs like Excel$^{\text{TM}}$and Matlab$^{\text{TM}}$. Replays of entire missions can also be visualized.

3.1.2 Mission Planner

Mission Planner is part of the ArduPilot Mega (APM) open source autopilot project, sponsored by 3DRobotics [3]. It is a ground control station application with similarities to Neptus, but is specialized to work with the APM autopilots like APM:Plane.

3.1.3 MAVProxy

MAVProxy is a UAV ground station software package for MAVLink based systems (such as APM) [79]. It differs mostly from Mission Planner by being only for UAVs and that it is a command-line, console based app. Because of this it is much lighter on CPU and memory usage.

3.1.4 APM:Plane

ArduPilot Mega Plane (APM:Plane) (formerly ArduPlane), is an autopilot firmware running on Pixhawk [1]. The firmware gives a fixed-wing aircraft fully autonomous capabilities with its functions such as three-dimensional waypoint handler, mission planning and automatic take-off and landing on runways. It also provides different possibilities for assisted flying with a combination of manual and automatic control of the plane.

3.2 Hardware

This section presents the hardware components of the automatic landing system.

3.2.1 UAV

The UAV chosen for the project is the X8 Skywalker seen in Figure 3.2, from Skywalker Technology. It is based on a flying-wing design, giving it excellent glide performance as well as fast low-power cruise speed. The airframe is constructed from molded expanded polyolefin (EPO) foam, making the frame very robust. It has a large space within the fuselage, making it ideal for experimental payloads.

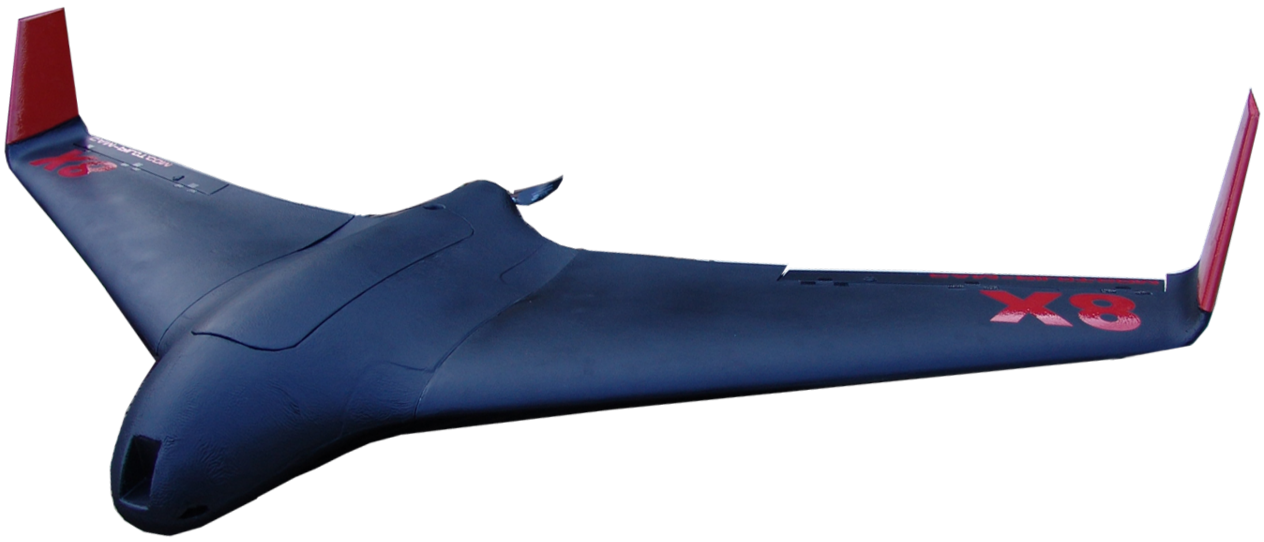


Figure 3.2: X8 Skywalker

The wingspan of 2120 mm allows for a maximum aircraft gross weight (AUW) of 3500 g. The UAV has to be fitted with the following components by the NTNU UAV-Lab in order to be flyable:

TX	Spektrum DX7s
RX	Spektrum Remote Receiver SPM9645
Servo	Hitec HS-5085MG
Motor	Hacker A40-12S V2 14-Pole
ESC	Master Spin 66 Pro
Propeller	Aeronaut 13x8 folding prop.
Battery	2x Zippy Compact 4S 5000 mAh 25C
Autopilot	3DRobotics Pixhawk w/3DR uBlox GPS with compass kit and airspeed sensor.
Telemetry link	3DRobotics radio (433 MHz) (VLOS operations) or Ubiquiti Rocket M5 radio (5,8 GHz)(BLOS operations)

3.2.2 Embedded Computer

The embedded computer in the UAV is the core of the automatic landing system, and there are multiple requirements that have to be satisfied. Firstly, the embedded computer has to be able to run the LSTS toolchain and RTK positioning algorithms in real-time. Secondly, weight and size are key factors in terms of flight endurance and payload capabilities, respectively. Furthermore, it must support the communication interfaces used by the peripheral components of the landing system.

The Pandaboard ES has been used in previous projects and theses at NTNU [73]. The Pandaboard has an OMAP4430 Dual-core ARM Cortex-A9 MPCore with Symmetric Multiprocessing (SMP) at 1 GHz each. It features 1 GB of low-power DDR2 RAM and a card reader for high-speed and capacity SD-cards.

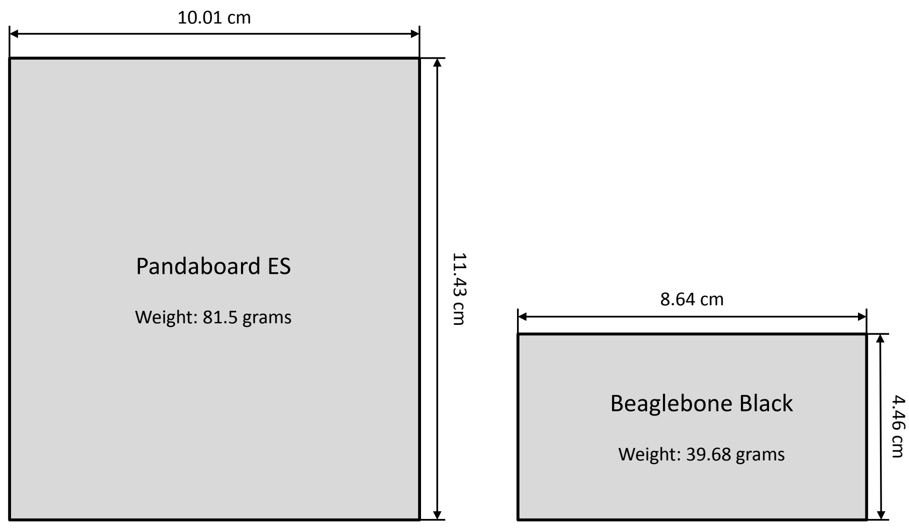


Figure 3.3: Pandaboard and BeagleBone Black dimensions

An alternative to the Pandaboard ES is the BeagleBone Black seen in Figure 3.4. It is a low-cost, open source community-supported development platform built around the 1 GHz Sitara AM335x ARM Cortex-A8 processor from Texas Instruments. The single Cortex-A8 processor is weaker in terms of computing power when compared to the Pandaboard. It is however more than enough to run the software needed for this project. It also features 69 GPIO pins, 5 serial ports and an I^2C-bus, making it ideal for experimental and prototyping applications. Another advantage is the dimensions, seen in Figure 3.3, shaving of half the weight and freeing up valuable space inside the fuselage compared to the Pandaboard.

Further information and complete specifications can be found at the BeagleBone Black [6] and Pandaboard [61] specification pages.

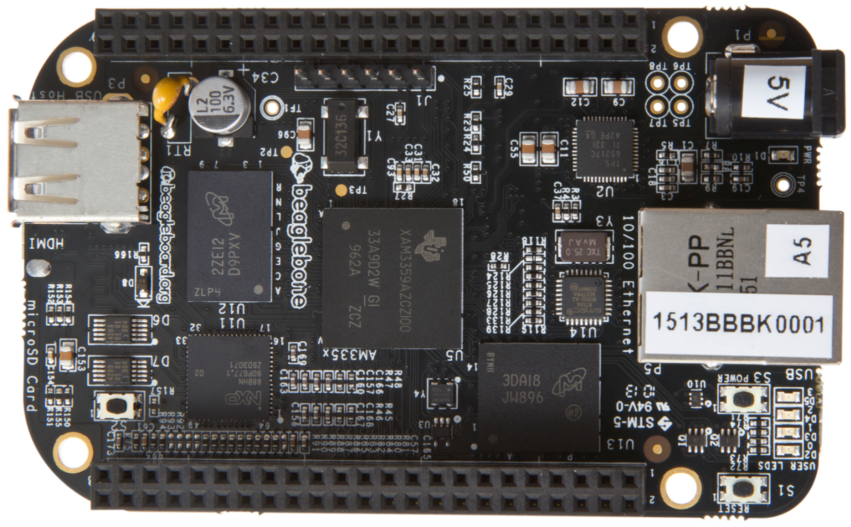


Figure 3.4: BeagleBone Black

3.2.3 Autopilot

The Pixhawk, seen in Figure 3.5, is an advanced autopilot system designed by an open-hardware project and manufactured by 3DRobotics. It is based around a 32-bit STM32F427 ARM Cortex M4-Processor and has an additional 32-bit STM32F103 co-processor as failsafe. The Pixhawk features a powerful internal sensor packet

- ST Micro L3GD20 3-axis 16-bit gyroscope
- ST Micro LSM303D 3-axis 14-bit accelerometer / magnetometer
- Invensense MPU 6000 3-axis accelerometer/gyroscope
- MEAS MS5611 barometer

in addition to an external airspeed sensor and GPS with integrated compass. The Pixhawk also features connectivity options for additional peripherals through UART, I^2C and CAN interfaces. Complete specifications can be found at the Pixhawk autopilot specification page [4].

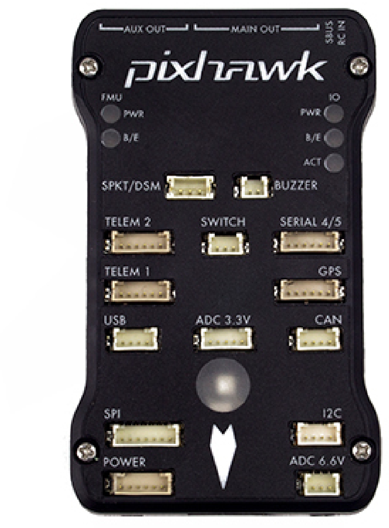


Figure 3.5: Pixhawk from 3DRobotics

Chapter 4

Method

In this chapter follows a description of the overall system architecture as well as design decisions and setup of all parts of the automatic landing system and later experiments. Section 4.1 describes briefly how the whole system is connected, Section 4.2 discusses ship motion prediction, Section 4.3 describes the designed landing path, Section 4.4 describes the designed guidance and control system, Section 4.5 describes the setup for the software-in-the-loop landing simulations and Section 4.6 describes a system designed to facilitate for accurate navigation.

4.1 System Overview

The complete system overview is described pictorially in Figure 4.1. It is divided into two main modules, the base station and the UAV. Communication between the modules is done over a long range WiFi link provided by two Ubiquiti M5 Rocket radios.

The base station placed on the landing ship consists of a base station computer, a GPS receiver (uBlox EVK-6T) with a Novatel GPS-701-GG GPS antenna and a telemetry link. All main data processing is done on the base station computer, where Real-Time Kinematic Library (RTKLIB) (str2str) is used to interface the RTK-GPS receiver and Neptus is used as the command and control software. To allow ship motion prediction, the Euler angles of the ship needs to be calculated in addition to the position obtained from the RTK-GPS. An inertial measurement unit (IMU) or inertial navigation system (INS) can be utilized, e.g. a Pixhawk like the one used for the UAV can be connected. DUNE is meant to perform the forecasting at the base station.

The UAV has slightly different software and hardware compared to the base station. The module is centered around the on-board embedded computer BeagleBone Black running GLUED. DUNE is responsible for forwarding and calculating the RTK-GPS solution using RTKLIB (rtkrcv). It also sends guidance commands to the Pixhawk and arranges communication with the base station through the telemetry link. The Pixhawk autopilot, containing both sensors and the autopilot software APM:Plane, is responsible for control of the UAV, i.e. controlling the servos and engine. The GPS receiver of the UAV is identical to that of the basestation, the uBlox EVK-6T, but is connected to the smaller M1227HTC-A-SMA GPS antenna.

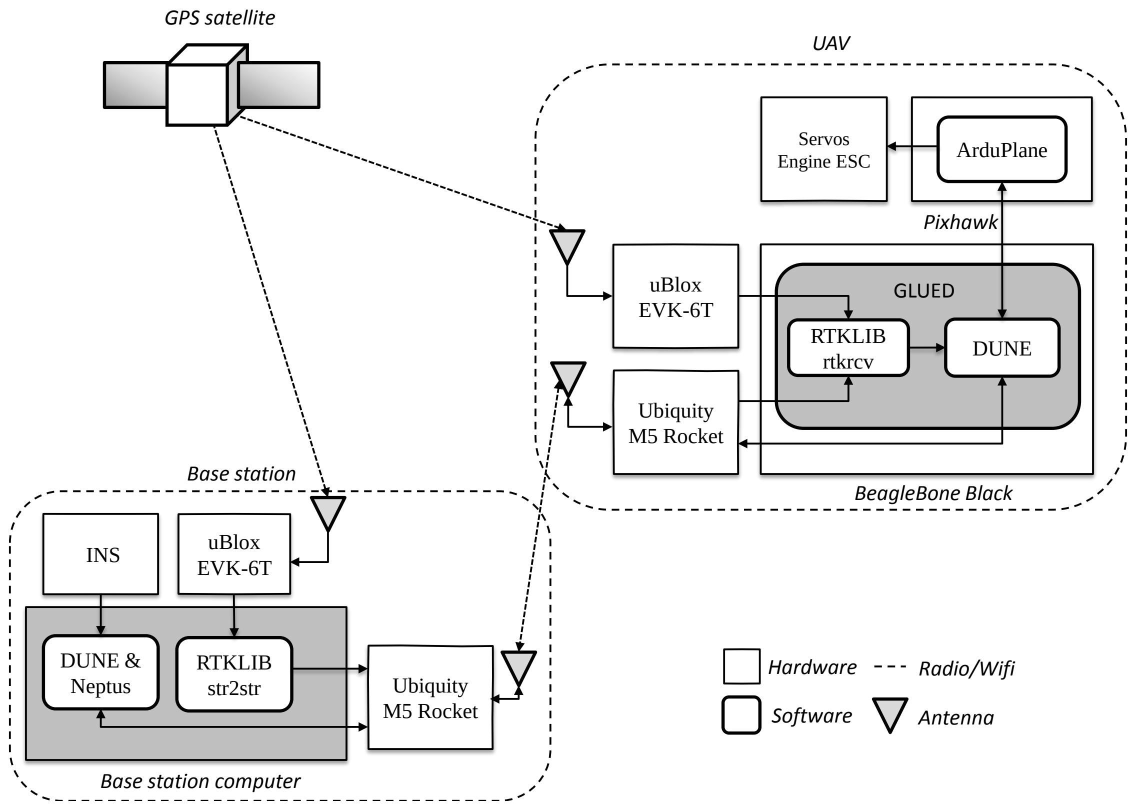


Figure 4.1: Software and hardware architecture of the automatic landing system with the base station to the left and UAV to the right

Below follows sections describing the design decisions and setup of all parts of the automatic landing system, including the simulations for the experiments.

4.2 Ship Motion Prediction

The reason for performing ship motion prediction in a UAV ship landing system is to increase the chance that the planned position and orientation of the landing target coincides with the actual case upon impact. If this is not the case, the UAV might miss the target. Constantly updating the path according to the target's current pose may lead to oscillations for the UAV. This increases wear and tear. The UAV will also lag behind the always updating path, hence the net will not be hit at the optimal position and angle.

It is not expected that an algorithm will predict the ship motion exactly, but it is expected that it is capable of predicting the motion to an accuracy where the prediction will be of practical importance. When deciding which method to use for ship motion prediction, several criteria are considered. It should be adaptable to any ship or sea state. This means a method involving a specific system model is undesirable. In addition, the system needs to handle nonlinearities. Due to the roughness of the sea and unknown sensor specifications, the forecasting method should also handle noise well.

Accurate prediction steps of up to 60 seconds for yaw and heave displacement is advantageous for a safe and smooth transient to the updated path, with 30 seconds as a recommended minimum. When performing forecasting during flight, the system needs to be fast enough to not cause significant delays. This also applies to training time if this is done on-line during flight.

The forecasting method considered to best meet the criteria is (MLP) ANN time series forecasting. Based on collected time series, the system will predict certain parts of the ship's pose on impact. This involves a calculation of the estimated time of arrival (ETA) to the net, which is briefly explained later.

4.2.1 Performance Measurements

When comparing different forecasting techniques, like varying network architectures or other approaches, several performance measurements are used.

Performance measurements

- Training time
- Percentage of forecasting errors within a defined range
- Percentage of forecasting errors at peaks within a defined range
- Mean absolute error (MAE)
- Mean squared error (MSE)

Mean squared error (MSE) is the most commonly used performance measurement for time series forecasting (like in [20, 37–39, 87]). It can be calculated using the formula given in Equation 4.1, where N represents the total number of vectors in the data set. Similarly, the mean absolute error (MAE) can be calculated according to Equation 4.2 (like in [37–39, 45]).

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(out_n - target_n)^2 \tag{4.1}$$

$$MAE = \frac{1}{N}\sum_{n=1}^{N}|out_n - target_n| \tag{4.2}$$

Another useful measurement is the percentage of errors within a certain limit (like in [34–36, 89]), both for the whole test set and at all the peaks (local minima and maxima). Looking specifically at the errors at all the peaks may be important because these points are often the most critical and hard to predict correctly. Figure B.3 in Appendix B.2 illustrates all the calculated peaks of a heave displacement plot. The training time is also used to compare the forecasting methods (like in [36]). This is the time it takes to completely train the forecasting method. It is especially important when the system is trained onboard a computer with limited computation power and when it needs to be trained/retrained on-line/inflight.

The data set used for calculating the forecasting performance related to forecasting errors is new to the network; it is not used for neither training nor validation. This way the generalization of the network is tested, instead of only how well it can learn the training set.

4.2.2 ANN Time Series Forecasting

Because the UAV landing system is meant to work on any ship, independent on ship design or changing sea states, on-line training of the ANN is preferred. This means the network is not static, but gets retrained as new data is collected during runtime. The reasoning behind this is that every ship moves differently in the sea, and a system prepared for every wave, wind and current condition would either be very big and complicated or too generalized. In addition, it is safe to assume that the ship used for net landing will lie in the water for some time before performing the landing, leaving time to collect enough samples for training.

For this thesis, a simple yet effective approach to on-line training is used. Instead of retraining the network at every new sample, the system collects a batch of data samples on-line before training the network with this data set. After some time, the sea state may change due to a change in weather and/or the ship may have moved. It is therefore wise to retrain the network periodically. The *training window*, meaning the period of time where the training samples are collected from, is chosen to be the last five minutes. Retraining can be performed every one minute, using a sliding training window.

The data that must be collected by the system is at least the same data to be forecasted. For this thesis it means a minimum of heave and yaw motion. In addition, by using data fusion, supplementary sensor information can be utilized. An experiment with data fusion of heave, roll and pitch will later be presented. The forecasting system needs to be connected to a position/orientation estimation unit, like the Pixhawk with an attached GPS.

4.2.3 ANN Forecasting for Oscillations With Varying Mean

A large number of time series used in artificial neural network (ANN) forecasting contains some kinds of oscillation. The mean around which it oscillates is often a constant, like zero, but in some cases it is not known prior to execution or it can even vary. One example is the yaw motion of a ship influenced by waves and wind; it is an oscillatory motion, but around an unknown mean $(0 - 360°)$. This may cause problems for the ANN when performing both training and execution of time series forecasting, as it prefers the data to be centered around zero and to be evenly scaled. In the following subsections an introduction to the problems will be presented, followed by a novel set of solutions.

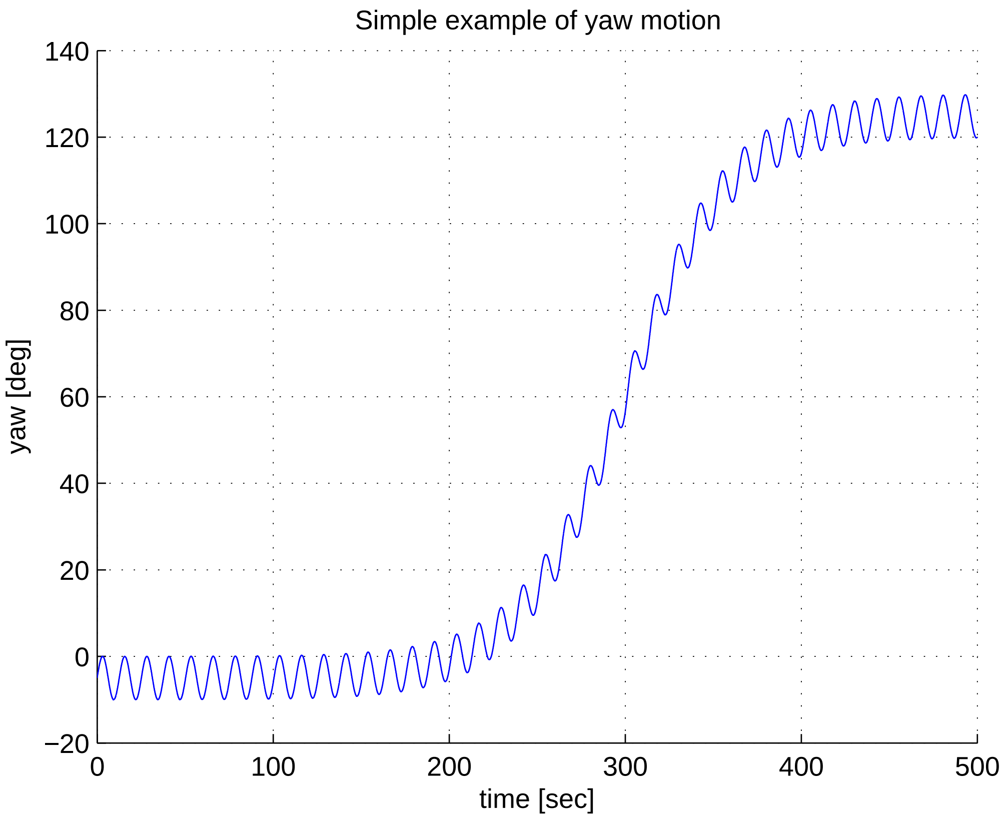


Figure 4.2: Simple example of yaw motion for a wave influenced ship

4.2.3.1 Problem Review

A shortcoming of ANNs is the problem of extrapolation described briefly in Subsection 2.3.1. ANN is a good tool for function approximation of the data given to the network during training, but may fail if tested on data too different from what it was trained for. For example, in the case of yaw motion of a ship, the network may be perfectly trained for the yaw range $0 - 20°$. When this network receives inputs in the range of $100 - 120°$ instead, the network is not prepared for inputs this big.

As described in the background theory chapter, time series prediction using ANNs is performed by collecting a given number of previous time samples into an input vector, and training the network to predict the value m time steps ahead. The general function for the back-propagation (BP) algorithm presented in Equation 2.7 gives us the following equation for weight adjustments in the first hidden layer (the equation and following reasoning is based on Haykin's book *Neural Networks and Learning Machines* [29]):

$$\Delta w_{ji}^{(1)}(n) = \eta \delta_j^{(1)}(n) x_i(n) \tag{4.3}$$

where $\Delta w_{ji}^{(1)}$ is the weight adjustment for the weight in the j^{th} neuron coming from input i, x_i. η is the learning rate and $\delta_j^{(1)}$ is the back-propagated error at neuron j.

Focusing on all the weights connected to the j^{th} neuron in the first hidden layer, $\eta\delta_j^{(1)}(n)$ will be constant. This means the algebraic signs of the different weight changes, $\Delta w_{ji}^{(1)}(n)$, will only vary if the signs of the individual inputs $x_i(n)$ vary. Hence, if all the inputs are of the same sign, all weights connected to the same neuron j will decrease or increase together. The problem with this is that the weight adjustments loose freedom which leads to a less efficient updating process. When the average input is shifted away from zero, there will be introduced a bias in the update direction and thus slow down the learning. On the other hand, if the mean of the input data is close to zero, more freedom is given to the weight adjustments and consequentially faster learning is achieved.

From Equation 4.3 another conclusion about the inputs can be drawn. The bigger the input x_i, the larger the weight update Δw_{ji}. If the input vectors in the training set are unevenly scaled, meaning that it in some cases contains only small values while in other cases only big values, an unbalance in the weight's update rate occurs. Scaling the inputs to become more even helps balance out the learning rate and thus improves and speeds the learning.

In the general case, neural network training is often made more efficient by minimizing these mentioned problems with pre- and post-processing of the training data. This means the entire set of training data is processed before entering the ANN, and then reversed back to its original form as it is outputted by the network, as illustrated in Figure 4.3. After the training is done, new data entering and exiting the network will be processed in the similar way and with the same parameters as when the network was trained. The following normalization is often used for preprocessing of ANNs, and will result in a data set with zero mean and unit variance:

$$\overline{x}_i = \frac{\sum_{n=1}^{N} x_i(n)}{N}, \quad \sigma = \sqrt{\frac{\sum_{n=1}^{N} (x_i(n) - \overline{x}_i)^2}{N}}, \quad x_i'(n) = \frac{(x_i(n) - \overline{x}_i)}{\sigma} \tag{4.4}$$

where x_i is the original data, N is the number of samples and $x_i^{'}$ is the normalized data.

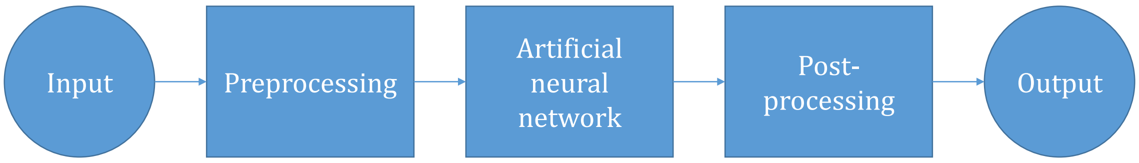


Figure 4.3: Pre- and post-processing for an ANN

The main problem with the example of yaw motion for a ship is that, although it oscillates around a mean, this mean may vary within the same training set, between different training sets and/or from the training set to the test set. For example (see Figure 4.2 above), an input set may consist of samples from a long period of oscillations between $-10°$ to $0°$ followed by a short period of similar oscillations between $120°$ to $130°$. This renders regular normalization or scaling useless as the input set, as a whole, may have zero mean and a well defined variance, but still suffer from the problems highlighted above.

In this example, all of the data in the input vector will have the same algebraic sign; all inputs are negative in the first period of $-10°$ to $0°$, but positive in the second period of $120°$ to $130°$. In addition, there is an unbalance in the scaling. In the second period the yaw oscillates around a mean ($125°$) that is 25 times larger than the absolute value of that in the first period ($5°$), while the amplitude remains the same ($5°$).

The same heuristic should also be applied to the outputs of all the hidden layers, and is therefore a reason to why the hyperbolic tangent is a preferred activation function for the hidden layers. Its output is evenly distributed around zero, between -1 and 1.

4.2.3.2 Suggested Solutions

The problems presented above can be summarized into three main problems:

- Extrapolation during execution
- Different scales on the input vectors during training
- Overweight of one algebraic sign in the input vector during training

To deal with these problems, originating from the fact that various input vectors may have different means, a novel approach is introduced. When preprocessing the data, instead of normalizing the entire data set or shifting all samples by the same amount, each individual input vector and its corresponding target value is treated individually. Five preprocessing methods are explained and compared in this thesis.

1. *Unchanged* - No individual processing applied
2. *Mean* - The mean of the input vector is subtracted
3. *Median* - The median of the input vector is subtracted
4. *Current* - The most current sample is subtracted
5. *Normalization* - Individual normalization is applied

The reasoning behind these methods is that by shifting all the individual input vectors to lie around zero, all three main problems are to a large degree removed. Concerning extrapolation, no matter what the original values were, the altered input vector will now always be close to zero. Hence, the extrapolation problem due to unexpected means of the input vectors is removed. If no individual preprocessing is applied, the network has to be trained for a very wide range of possible means. When the preprocessing methods presented above are applied, the network can specify its training for values around zero.

During training there will also be a more evenly distribution of positive and negative values for the inputs, and there will not be the possible huge scaling differences between the different input vectors. It is important to notice that after training of the network with preprocessing, the output will not be the final predicted value directly, but first needs the preprocessing "reversed" with post-processing.

Below follows explanations of the five methods with accompanying figures.

Unchanged

The *unchanged* method does not do any processing on the data. This is used for comparison purposes. Figure 4.4 illustrates a sine wave $y = 5\sin(x) + 5$ working as the unprocessed example of the values in a single input vector.

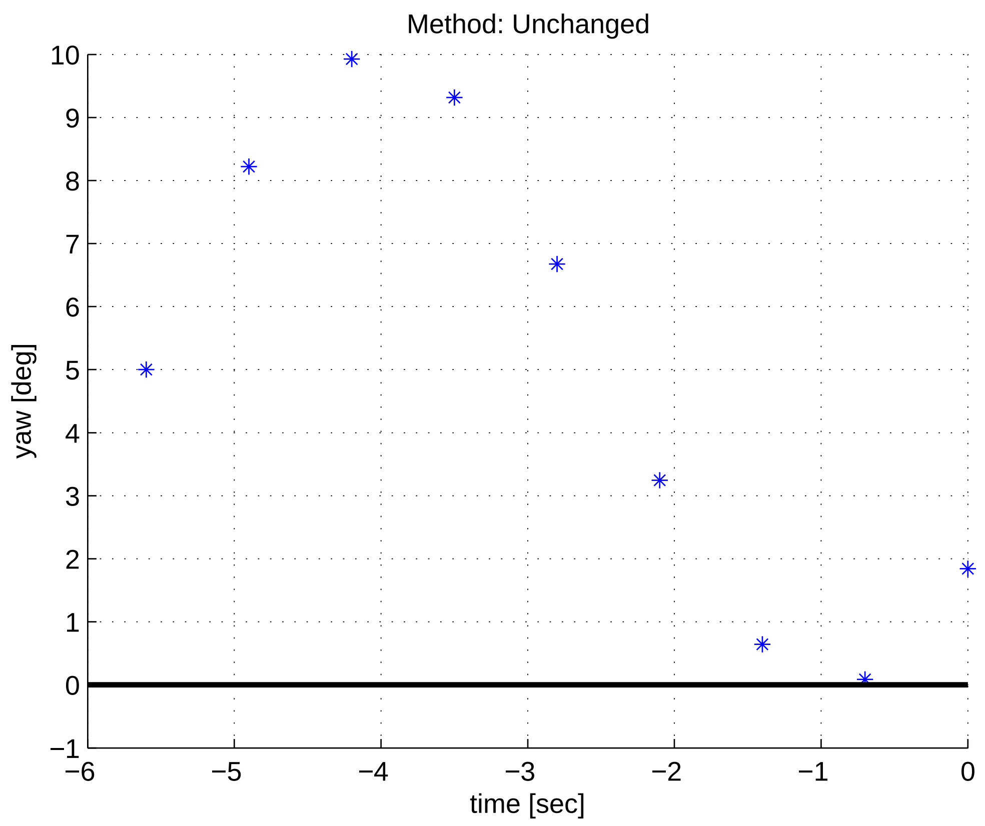


Figure 4.4: *Unchanged* example

Mean

When calculating the mean value of an input vector and then subtracting this from each element, the new mean value of the modified input vector becomes zero. The output of the network, meaning the predicted value m time steps ahead, will be the deviation from the input vector's mean.

This is a slightly indefinite concept for the neural network to learn, because the prediction is not a deviation from a fixed input. How the different inputs relate to the mean may vary. The most current sample may for example sometimes have the same value as the mean, or in other cases be far off. Figure 4.5 illustrates the same sine wave as displayed in Figure 4.4, but with the mean value subtracted.

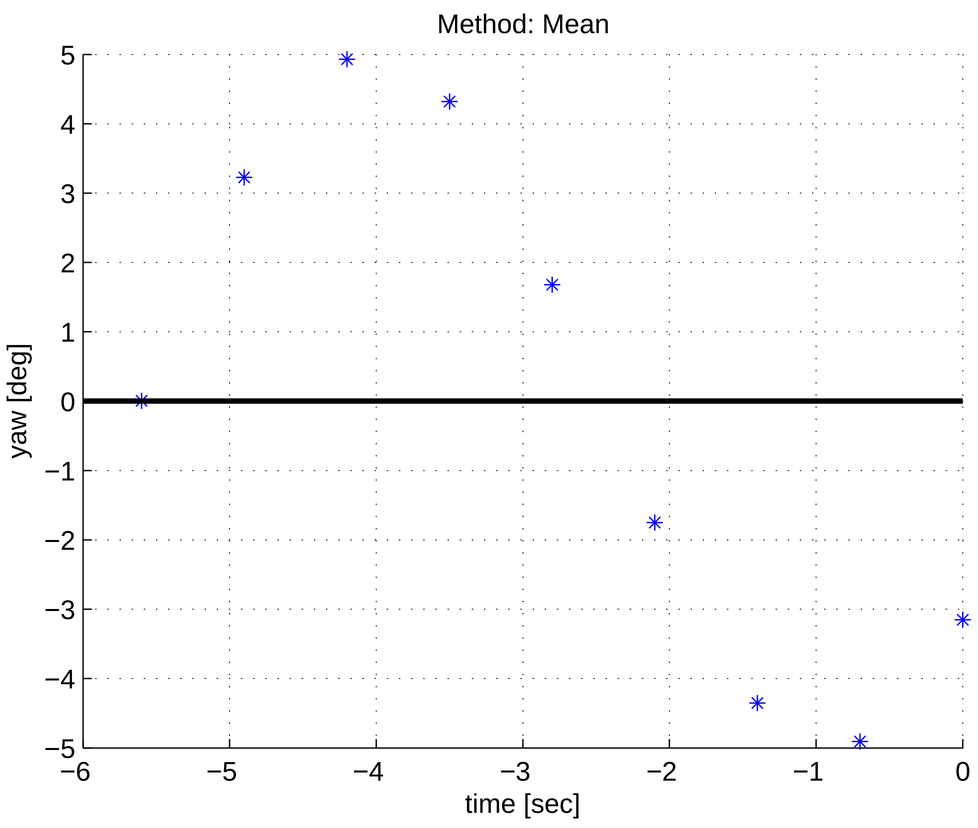


Figure 4.5: *Mean* processing example

Median

Subtracting the median value of the input vector guarantees the same amount of inputs above and below zero, thus maximum freedom in weight adjustments is achieved. There is no bias in the update direction that will slow down the learning. The output of the network will in this case be the deviation from the input vector's median.

This method will in many cases look very similar to *mean*. It also has the same problem of a changing relationship between the prediction and the different inputs, as which input that has the median value will vary. Figure 4.6 illustrates the aforementioned example sine wave with the value of the median input subtracted and marked in red.

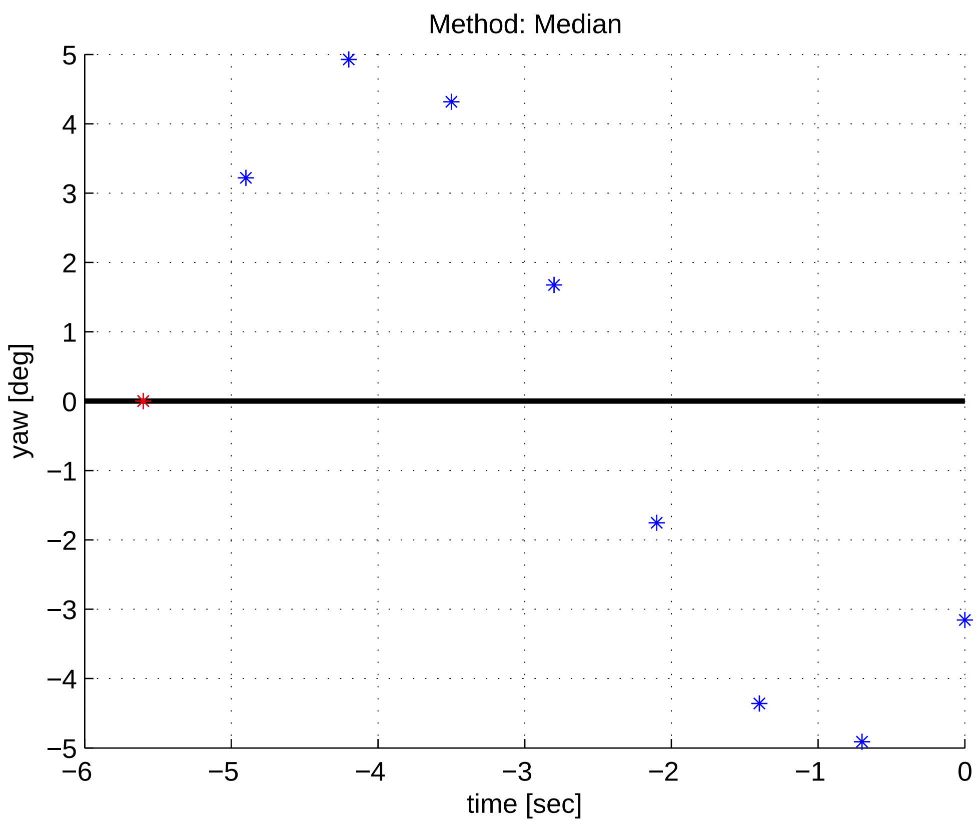


Figure 4.6: *Median* processing example

Current

The *current* processing method is invented to avoid the problems of the two aforementioned methods regarding which input becomes zero after subtraction. By subtracting the value of the most current sample from the input vector, the output of the network will always be the deviation from the currently newest sampled value. Hence, there is a more constant relationship between the inputs and the output.

When the most current sample is subtracted, the input corresponding to this sample will always be zero (usually the last element in the input vector). Hence, this element can be omitted as input to the network with no loss of information. This presupposes that no other processing is later done to the inputs, which is often done by default when using neural network packages like the one for Matlab. The same reasoning could have been used for *median*, but there is only guaranteed to be a zero input when the number of inputs is odd, and even then which input becomes zero varies according to where the median is.

The downside of the *current* processing method compared to the two aforementioned is that there is only a guarantee that the last element in the input vector becomes zero. The rest of the vector may for example contain only large all positive elements, leading to the problems previously described. In practice, this method will lead the input vectors to be more even in size than *unchanged*, as large biases are canceled out by the subtraction. There is also a higher chance that the modified input vector will be better distributed around zero than it originally was. Figure 4.7 illustrates the sine wave example with the value of the most current sample subtracted and marked in red.

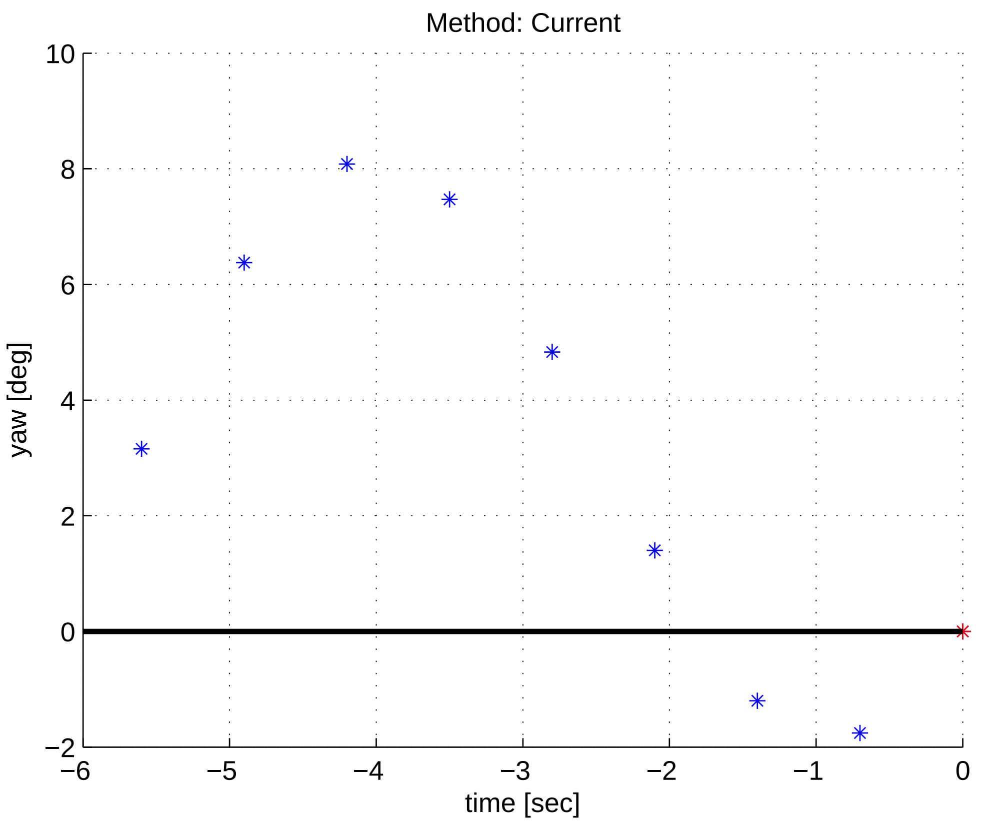


Figure 4.7: *Current* processing example

Normalization

Normalization is the processing method where each individual input vector is normalized, and its training target and the network output also needs to be normalized and de-normalized, respectively, using the same parameters. Unlike the other methods that only shifts the inputs by a certain amount, normalization also scales the inputs. This means not only a close to evenly distribution between positive and negative inputs, but also that all input vectors are scaled to lie in the same range. Depending on the normalization method, the input vector will either get a specific variance, e.g. the unit variance example in Equation 4.4, or be normalized to the same range, e.g. between -1 and 1.

Considering the theory discussed earlier, *normalization* processing method should yield the fastest training. On the other hand, when the individual normalization process has made all the input vectors this similar, it is very hard for the ANN to find the characteristics that gives a certain target output. Figure 4.8 illustrates the sine wave after normalization to $\{-1, 1\}$.

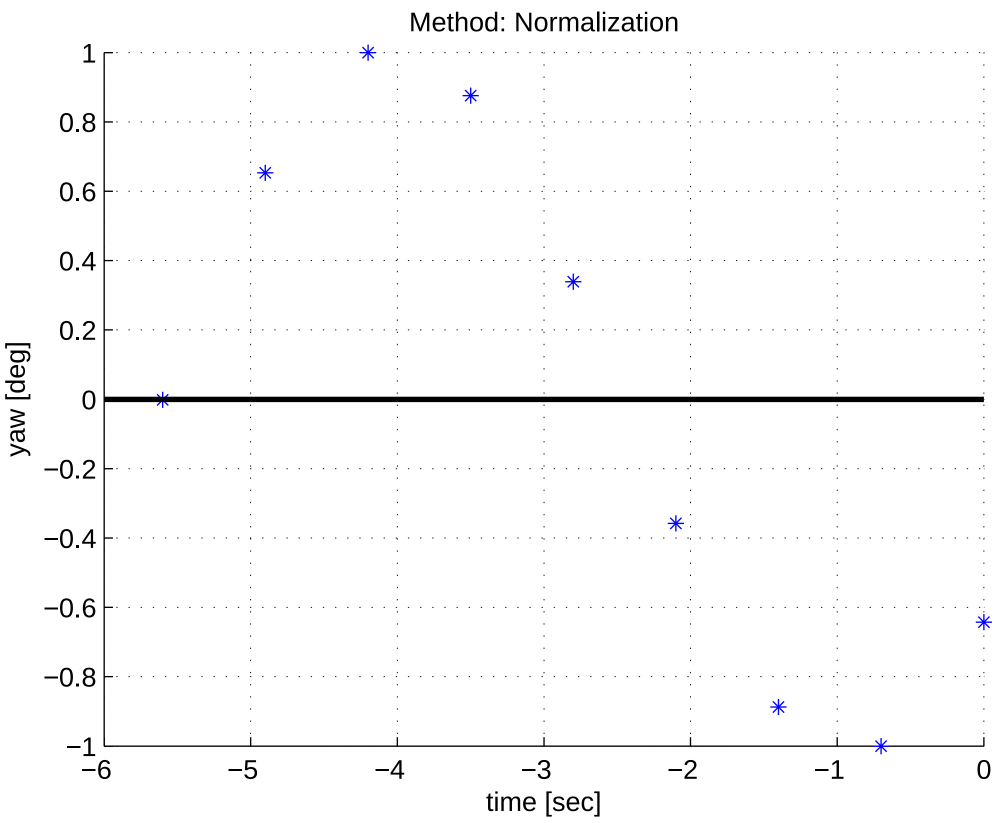


Figure 4.8: *Normalization* processing example

For all the last four methods, the extra computational time caused by the pre- and post-processing is not discussed, as it is neglectable compared to the training time. Another small additional cost of using these methods is the parameters needed to be saved from pre- to post-processing, input vector to prediction output. For *mean*, *median* and *current*, one parameter needs to be temporarily saved; what was subtracted from the input vector needs to be added to the predicted output. For *normalization*, two parameters must be temporarily saved to reversely perform the de-normalization of the output.

An interesting problem only discussed briefly here is the problem of heading measurements around 0/360°. This may lead some values in the unchanged input vector to be right above 0, while others are right below 360. In reality they belong to the same continuous motion spanning only a few degrees, but the discontinuity at 0/360° leads to problems for the ANN. A solution is to not normalize the heading when it passes 360°, but this may not be possible. Marinkovic and Markovic [51] proposes a solution for preprocessing of data entering an ANN that involves transformation to polar coordinates. The same problem occurs when using the range ±180°.

4.2.4 Estimated Time of Arrival

When using the technique of motion forecasting for ship-net landing it is important to know the estimated time of arrival (ETA) to the landing net. Uncertainty in the ETA may in the worst case render motion forecasting useless as ETA is the value used for the prediction step.

The simplest calculation of ETA is done using the linear distance to the target divided by the current speed towards it. This follows from the basic physics equation of time as a function of distance and speed [69]:

$$Total\ time = \frac{total\ distance}{average\ speed} \tag{4.5}$$

Instead of the linear distance, the length of the remaining path can be used when this is known. This stresses the importance of a flyable path, which increases the chances of the planned path equaling the actual trajectory being flown. The speed value will be more accurate if filtered, e.g. with a low-pass filter or averaging over a predefined time period. If the speed is planned to be altered along the path, like being lowered before net impact, this should be included into the average speed calculation.

The frame of reference for the speed should also be considered. Airspeed is relative to the air or wind, while ground speed is the horizontal speed relative to the ground. If ground speed is used, only the horizontal part of the path should be used for total distance. When using the total remaining path length for distance calculation, speed relative to the three dimensional path needs to be considered.

For reasons of simplicity, the rest of this thesis related to forecasting assumes a perfectly calculated ETA, while the ETA used during landing simulations is already implemented in DUNE as:

$$ETA = \frac{norm(\Delta x, \Delta y)}{norm(vx, vy)} \tag{4.6}$$

where Δx and Δy are the longitudinal error and cross-track error respectively, related to the straight line path from the previous to the next waypoint. vx and vy are the horizontal and vertical speed respectively.

4.3 Landing Path

The UAV is supposed to fly and land on open water. Hence, it is assumed no obstacles in the landing path other than the water below, i.e. path planning in obstacle free space. Since there are no obstacles to take into account, deliberative path planning is used. The only variables needed to know when planning the landing path is the location and orientation of the net, and the height of the UAV when landing is executed.

Waypoints will build the basis for the landing path. It is divided into three phases: approach, glideslope and final approach (FA). In the approach phase the goal is to place the UAV at a distance sufficiently far away from the net to ensure it has time to descend down to the net's height. It also makes sure the UAV is on a straight line perpendicular to the landing net. When entering the glideslope phase, the altitude decreases with a constant rate. After the altitude is sufficiently decreased and the UAV is close enough to the net, the FA phase begins. Its task is to make sure the UAV converges straight into the middle of the net at a certain vertical angle of attack and at a low speed.

A problem may arise if the guidance laws have a large acceptance radius around the waypoints or if there is a position error making the autopilot believe it has reached the last waypoint before it actually hits the net. To ensure the autopilot does not prematurely finishes but keeps aiming at the center of the net all the way through it, another waypoint is added on the other side of the net as an extrapolation of the path in the FA phase.

A decoupled approach to path planning is used where the lateral and longitudinal paths are described separately, but combined together they give a complete three-dimensional path.

4.3.1 Lateral Path

The main objective is to make sure the UAV lands safely in a net. It needs to be able to do this from any initial in-air position. The lateral path only consists of straight lines. This is because there are no obstacles or time constraints needed to be taken into account. A straight line into the net is the easiest for the UAV's guidance and control systems to follow. The straight lines are normal to the net, with the exception of the initial line from the current UAV position to the beginning of the lines leading into the net. Figure 4.9 illustrates an example path.

The length of the approach phase is chosen to be long enough for the UAV to converge onto the straight line path before glideslope starts. This is necessary because the lateral path is not flyable due to the sharp turn leading into the approach phase. There will be oscillations, but the approach phase assures they are eliminated before entering glideslope.

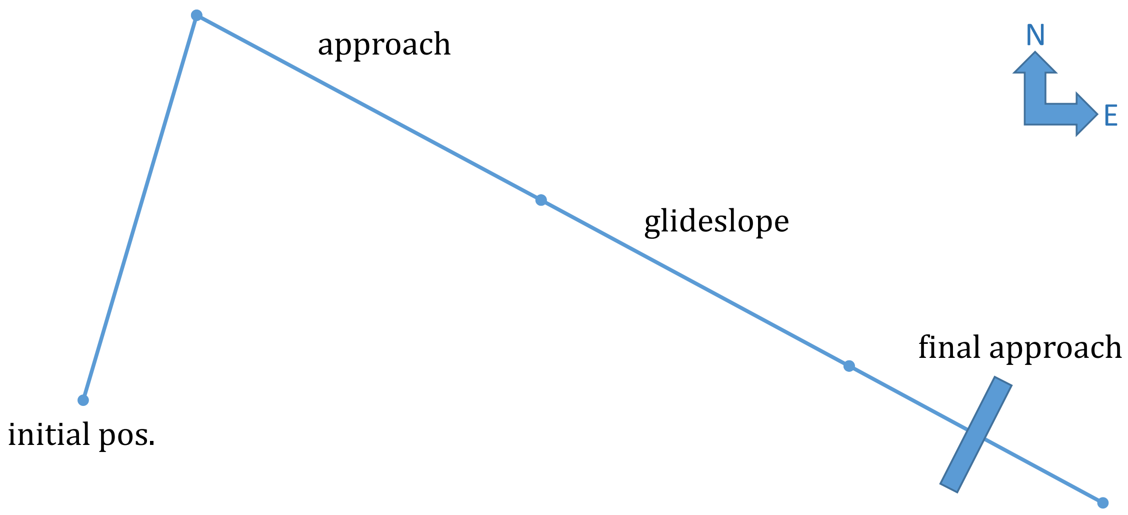


Figure 4.9: Lateral path example

4.3.2 Longitudinal Path

The longitudinal path defines the desired height along the lateral path. In the context of this path, all mentions of height are in the meaning of height above the ground level (ship deck) on which the net stands.

For simplicity, the desired height is constant from the initial position all the way through the approach phase. This is because the approach phase is only meant to align the UAV horizontally. The desired height is set to equal the actual height of the UAV at the moment the land command is requested by the operator, as long as this is above a safety height. During glideslope the goal is to loose altitude as fast as possible. The descend angle is a function of the maximum descend rate of the UAV. FA has a constant vertical angle of attack ensuring a safe landing into the net. Figure 4.10 and Table 4.1 gives a description of the longitudinal path consisting of straight lines.

Because the rate of the desired height changes multiple times along the path, and possibly changes close to the net when FA starts, it is important to ensure a flyable path. The straight line path in Figure 4.10 is not flyable as it has discontinuities in its first derivative, meaning there will be sudden jumps in the desired climb rate (height derivative). This will lead the UAV to overshoot the desired path as it does not manage to change desired climb rate instantaneously.

The APM:Plane autopilot system takes said desired climb rate as input. Hence, the longitudinal path should be continuous in its first derivative, resulting in no sudden changes in the desired climb rate. A Dubins path solves this problem by connecting the straight lines with constant-curved segments. This way the tangent of the path will change in a smooth manner. The only parameter needed for this design is the maximum curvature bound.

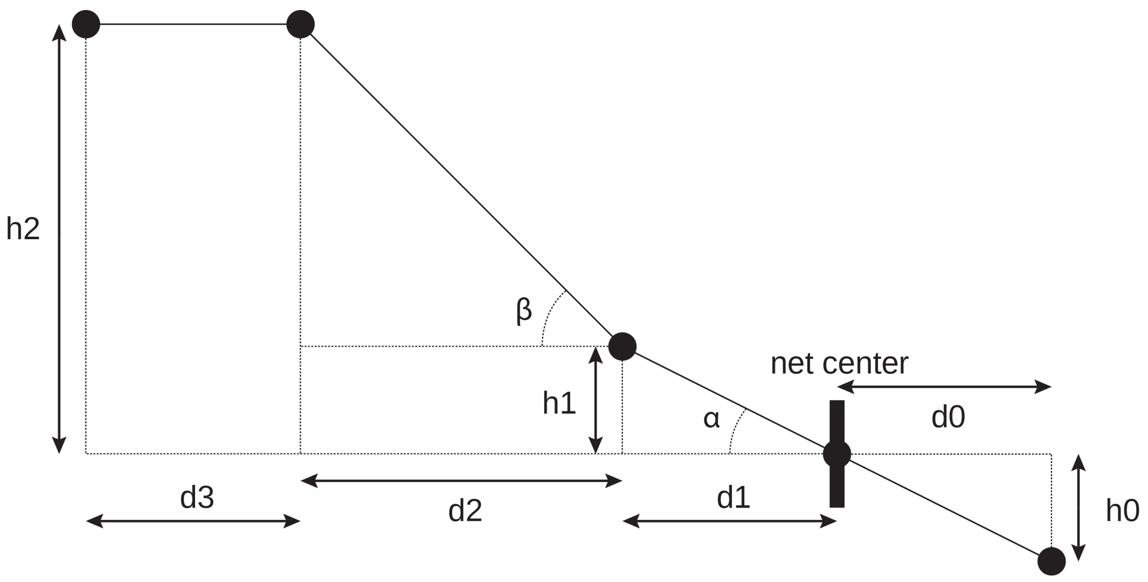


Figure 4.10: Longitudinal straight line path. Description of variables in Table 4.1

To make a path fully flyable, continuity in the second derivative is also required to avoid curvature discontinuities. Due to the rough conditions when flying a UAV outdoors and the fact that the low-level controller only takes climb rate as input, making the path continuous in the second derivative does not have any practical significance. Because of its simplicity and that it is proven to be the shortest path connecting two poses in the presence of a maximum curvature bound, Dubins path is chosen as the basis for the longitudinal path.

The radius of the circles connecting the straight lines is inversely proportional with its curvature, and the curvature is limited to not being greater than the maximum curvature bound of the UAV. This ensures that the desired change of climb rate is not happening faster than the physical limitations of the UAV. Figure 4.11 illustrates an exaggerated example of such a path. The UAV will follow the straight lines until it reaches a circle. It then switches to follow the circumference of this circle until a new straight line segment is reached. To find the desired climb rate required to follow this path, the derivative is calculated (Figure 4.12). This derivative represents the ratio between the horizontal ground speed and the vertical speed, meaning that the desired climb rate is ground speed multiplied with the Dubins path derivative.

The maximum curvature bound is not necessary constant as it in addition to the maximum aileron/elevator deflection also is a function of variables like airspeed and wind [22]. Therefore, the optimal/fastest switching between the straight lines requires on-line calculation of the circles radii. For this landing system there is no strong requirements for optimality around the intermediate waypoints. Since the distances between the waypoints are also controlled, a constant and common radius for all circles can be set. This radius needs to ensure that both the bound on maximum curvature is never breached and that the circles are small enough to switch between the straight line segments in time for the next waypoint.

Table 4.1: Description of variables mentioned in Figure 4.10

Variable	Brief description
h0	Vertical distance from center of the net to the aiming point
h1	Vertical distance from center of the net to where FA starts
h2	Relative height of the UAV above center of the net, measured at the execution of the land command
d0	Horizontal distance from the net to the aiming waypoint, making sure the UAV keeps flying all the way "through" the net
d1	Horizontal distance from the net to where FA starts
d2	Horizontal length of the glideslope phase
d3	Horizontal length of the approach phase, ensuring enough distance for the UAV to align properly before glideslope begins
α	Angle of attack on net impact
β	Descend angle during the glideslope phase

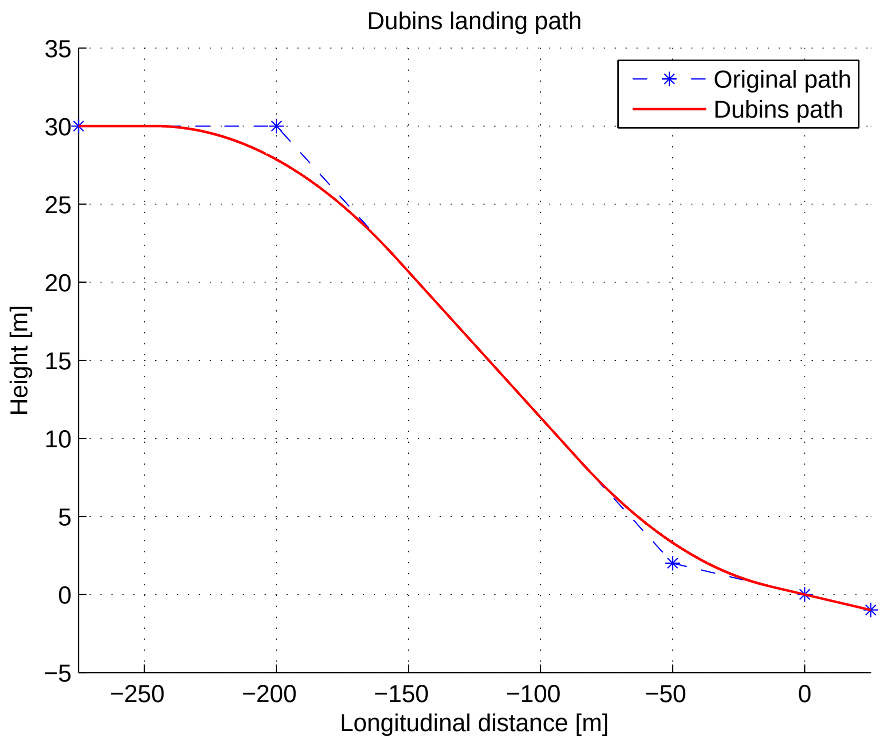


Figure 4.11: Example of a Dubins path for climb rate guidance

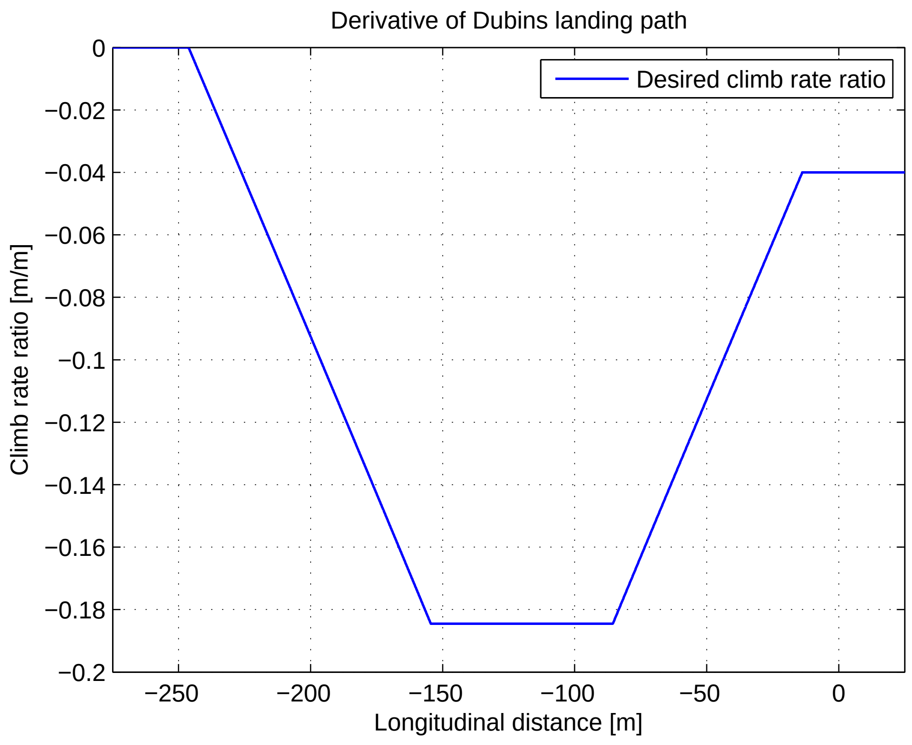


Figure 4.12: Example of the derivative of a Dubins path, giving desired climb rate ratio

4.3.3 Dynamic path

Because the UAV is supposed to land on a ship, the net can not be expected to be static. As the ship rotates, the path should rotate along with it. Movements in surge, sway and heave should also be followed by adjustments of the landing path. If this is not done, the net pose upon impact may vary enough from the original pose to make the UAV miss it and crash.

As discussed in details later, ship motion prediction may be used to predict where the ship will be when the UAV reaches the net. Together with a dead zone around movement changes, this ensures the path does not need constant updates for every small movement.

When an update of the path is desired, a new path is constructed based on the calculations from the old one. The waypoints already reached are omitted, while the remaining ones are rotated according to the change of ship heading, and translated according to the change of position. To make sure the UAV aligns on the straight line going through the updated waypoints, and new waypoint is added at almost the same distance from the net as the current position, but on the line perpendicular to the net.

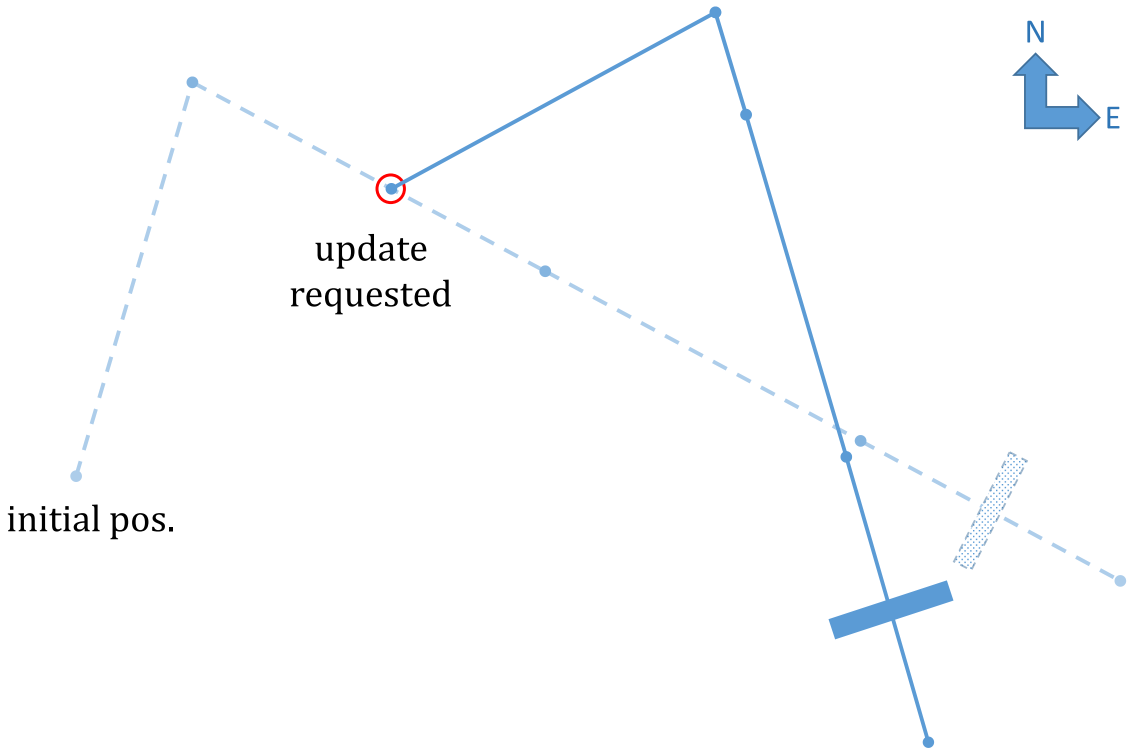


Figure 4.13: An example of a dynamic path with an update requested during the approach phase

Figure 4.13 demonstrates the new lateral path when a large update of the heading and position of the net is requested during the approach phase. The longitudinal path will be shifted up or down according to the heave displacement.

4.3.4 Evasive Maneuver

If there is a big risk the UAV will miss the target when approaching the landing net, the landing should be aborted in time to avoid a possible accident. Below follows the criteria used to evaluate it the risk of missing is too high. These criteria will be evaluated only from FA is entered, as the UAV may have time to correct itself further away from the net.

Criteria for evasive maneuver

- Cross-track error exceeds ±1 m
- Altitude error exceeds ±1 m
- Course over ground vs. heading exceeds ±45°
- RTK-GPS solution quality below 1
- Ignore evasive maneuver

The above criteria and following explanations are a further development of the criteria found in the NTNU MSc thesis by Skulstad and Syversen [73].

Cross-track error

The net used as landing target is 5 m wide, while the wing span of the X8 UAV is 2.1 m (as described in Subsection 3.2.1). This gives less than 1.5 m from the horizontal edges of the net to the tip of each wing. From this a cross-track error of less than ±1 m was chosen as a criterion for successful landing.

When deciding if the cross-track error criterion is violated, several samples are examined. An evasive maneuver is only executed after 5 consecutive violations at a sampling rate of 10 Hz. This is because it may be noise in the position data, like RTK-GPS "spikes" leading to short temporary "false" errors. Another reason for not ordering an evasive maneuver based on a single sample is to give the UAV time to correct the error.

Altitude error

The net height is 3 m, but the height of the X8 is minimal. This leads to slightly less than 1.5 m from the edges to the UAV, giving ±1 m as a criterion. Similar to the cross-track error, 5 consecutive violations are required for an evasive maneuver to be ordered. The reasoning for this is the same as previously described in "Cross-track error".

Course over ground vs. heading

When there is a strong crosswind, the course over ground (COG) will differ from the UAV's heading and the heading of the net. If the difference between the COG and UAV heading is larger than 45°, the UAV will hit the net wing first. To avoid stress on the UAV fuselage this should be avoided.

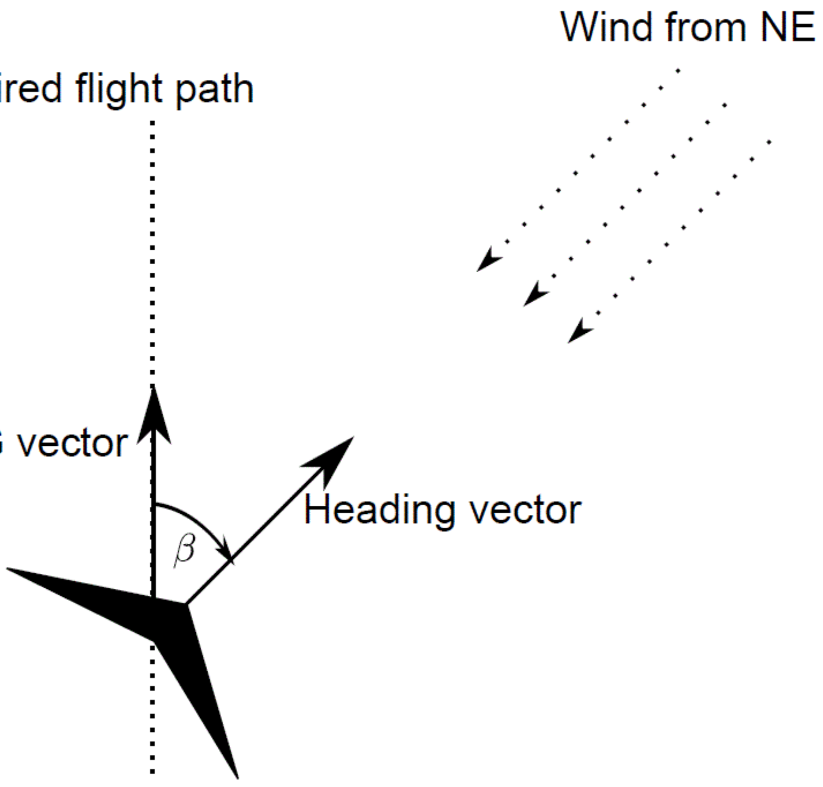


Figure 4.14: Course over ground vs. heading - NE plane, from [73]

The same sampling technique and reasoning as described in "Cross-track error" is used. A wind gust may cause a temporary large difference between COG and UAV heading, but the series of samples avoids an immediate execution of an evasive maneuver.

RTK-GPS solution quality

Quality of the RTK-GPS positioning solution is measured by the ambiguity resolution (AR) ratio. Any value above zero means it has sufficient information to start position estimation. Skulstad and Syversen [73] considers an AR ratio below one to give a too large standard deviation of the position estimate, concluding that the value should be above one. For the rest of this thesis, the RTK-GPS solution quality is not considered. More information on AR ratio can be found in said paper.

Ignore evasive maneuver

There may be reasons to ignore an evasive maneuver and force landing. If the UAV has a low battery level it may not be able to do an evasive maneuver and initiate a new landing. In this case it is better to land with a small violation of some of the evasive maneuver criteria and risk damaging the fuselage, than to loose power over open water and risk damaging valuable payload or possibly loose the UAV completely.

In situations with damaged equipment or extreme weather conditions there may be some criteria that are always violated. To ensure automatic landing is still possible and not constantly aborted by the system, it is important to have the option of ignoring a request for an evasive maneuver. The choice to ignore an evasive maneuver could be decided by the system, e.g. by monitoring the UAV battery level or after a number of consecutive failed landing attempts, but it is safer to have it as an option for the operator.

A last reason to ignore an evasive maneuver request is if the distance from the UAV to the net is too short to properly complete the maneuver. What may originally have been only a small deviation a second away from the net may lead the UAV too try a steep climb and sharp turn in an attempt to follow the evasive plan, but due to the limited time lead to a crash into the net edge or the boat. In these cases, it is better to continue aiming at the center of the net. To allow enough time for a proper evasive maneuver (including computational delay and servo delay), a request will be ignored if issued later than an estimated time of arrival of 4.0 seconds. This "time of arrival factor" is tested experimentally later in Subsection 5.2.5.

4.3.4.1 Evasive Maneuver Path

The purpose of the evasive maneuver is to abort/reset a landing sequence and avoid a possible crash with the boat or water. To ensure this, the requested path when an evasive maneuver is ordered involves both a steep climb and a quick turn at the same time.

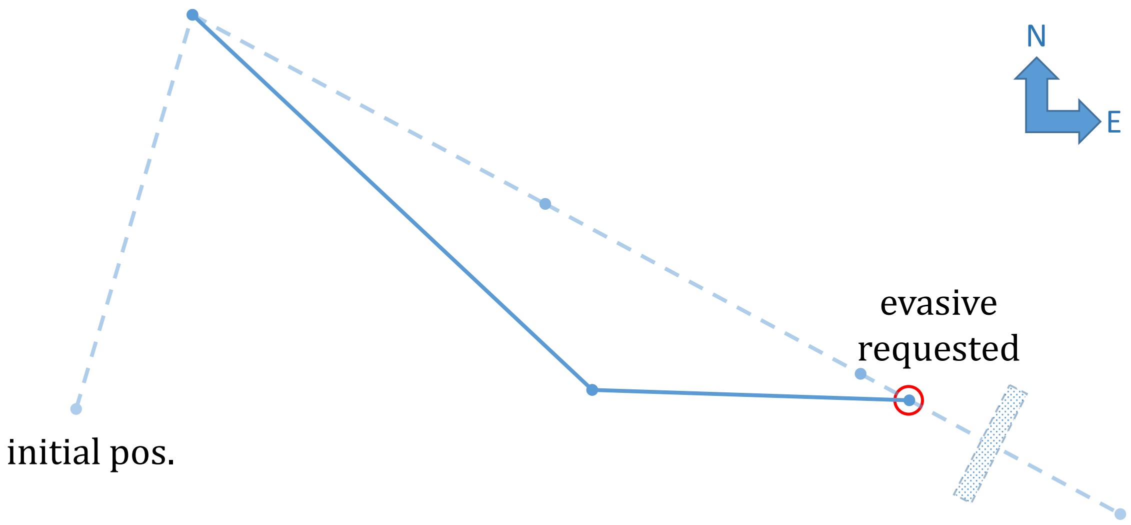


Figure 4.15: Lateral path after an evasive maneuver is ordered during FA

4.4 Guidance and Control System

This section presents all the guidance systems and controllers implemented to ensure a secure landing for the UAV. The autopilot software APM:Plane supports a decoupled approach where the longitudinal dynamics (forward speed and climb rate) can be commanded separate from the lateral dynamics (roll angle). By using this approach, two high-level guidance systems can be implemented independently in DUNE, while taking advantage of the built in low-level controllers in APM:Plane.

An easier option would be to send waypoint commands straight to APM:Plane, leaving it to deal with both guidance and control. Preliminary SIL experiments were performed to evaluate this option. Because this thesis focuses on the final UAV landing system, little focus is placed on the implementation and execution of the preliminary experiment, other then a brief explanation found in Appendix A.1.1. It also contains plots illustrating the performance of the APM:Plane height controller when it receives the five waypoints directly.

The conclusion from the waypoint test is that the UAV altitude is not controlled in a sufficiently good manner to be used for UAV net landing; it has a constant deviation from the desired path, and the desired path generated by APM is itself infeasible for the UAV to follow. The great advantage of controlling the desired climb rate and roll angle directly, as opposed to only demanding a set of waypoints, is that more control is gained over the actual trajectory taken by the vehicle. Hence, this is the method used for the rest of this thesis.

4.4.1 Lateral Guidance

The first lateral guidance system tested was a controller named *Aerosonde*, based on a paper by Niculescu [59]. It was implemented in DUNE by E. Marques, R. Bencatel and F. Ferreira from University of Porto, Portugal. The input to this controller is a set of waypoints, and the output is a bank angle. This is the desired input-output architecture for the lateral guidance system, but Aerosonde does not have any proper means to compensate for constant disturbances like wind or a small drift. This means it in some cases has a small deviation from the desired straight line path between two waypoints. Appendix A.1.2 illustrates a preliminary SIL simulation using this controller, and demonstrates how it in lateral direction leads to both oscillations and a constant deviation from the desired path. There was no wind during the simulation.

To ensure disturbances are compensated for and the lateral path is followed closely, integral action can be added. An integral line-of-sight (ILOS) guidance law based on a paper by Caharija et al. [13] was chosen. It is LOS guidance with an additional integral term. Originally it was designed and tested for ship control, but it can easily be used for UAVs after some tuning. The DUNE implementation was done by W. Caharija from University of Porto, Portugal. In Appendix A.2 the ILOS tuning values are listed.

A problem with the implemented ILOS guidance law is that it outputs desired course, while the low-level controller in APM:Plane expects desired bank. There is no direct conversion for this as it is for course rate to bank angle. Therefore, a new intermediate bank-to-turn (BTT) controller was designed to convert course into bank.

The BTT controller was designed using a PID. Appendix A.3 presents the PID tuning values in addition to a step response plot from a straight line test. The whole lateral guidance system is designed based on successive loop closure [7]. Figure 4.16 illustrates a simplified version of the successive loop closure for lateral guidance. The outer most loop is the ILOS guidance system that takes the path as input and outputs commanded heading based on UAV position updates. Commanded course enters the BTT-PID controller in the middle loop together with measured course angle. It outputs commanded roll angle to the APM:Plane low-level controllers in the inner most loop. Measured roll angle is also an input her, and the output is commanded aileron deflection for the UAV.

Separation of bandwidths is important when using successive-loop-closure design. This involves the inner loops having higher bandwidths than the outer. When the inner loop is adequately tuned with a high frequency, the outer loop may see it as a single unit gain. Hence, after tuning the inner loop, the outer loop can be designed and tuned independently. For the lateral guidance system used, this means that the PID controller can be tuned independent of the controllers in APM:Plane, considering the bandwidth separation is high enough. At the same time, the PID needs to be fast enough to not affect the ILOS.

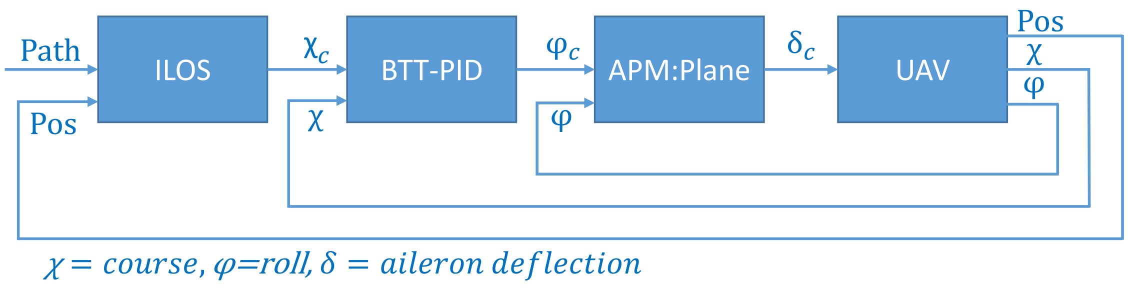


Figure 4.16: Simplified successive loop closure for lateral guidance

Because the lateral guidance system is designed as a modular system, where the ILOS and BTT controllers are designed and tuned independently, the outer guidance block can be changed with any controller that outputs desired course angle.

4.4.2 Longitudinal Guidance

When choosing the height controller, a preliminary SIL simulation was performed on a sliding mode controller already implemented in DUNE by R. Bencatel and J. Fortuna, called *Height*. It takes waypoints as input and sends climb rates to APM:Plane. A plot of the height from the simulation can be found in Appendix A.1.3.

The controller proved insufficient to make the UAV follow a predefined path. This is because it works like set-point regulation. Instead of following a smooth path between the waypoints, the UAV tries to reach the height defined by the next way-point as fast as possible, and keeps that height until the waypoint is reached. This makes it hard to ensure the net is reach at a specific vertical angle of attack. In addition, there will normally be an overshoot at every new set-point because the UAV enters the waypoints with a low vertical angle, demanding an instantaneous change of climb rate.

To gain more control over the vertical trajectory taken by the UAV and the angle at which it enters the net, a new height controller was implemented. Instead of using a guidance system to follow fixed set-points or straight lines between the waypoints, like for lateral guidance, a controller was implemented to be able to follow the feasible longitudinal Dubins path described in Section 4.3.

Since the input to APM:Plane should be commanded climb rate, and that climb rate is the time derivative of height, the new controller can use the derivative of the longitudinal path. The path is defined as desired height at specific distances from the net, not with time as the x-axis. Thus, to get desired climb rate from the path derivative, it needs to be multiplied with the current ground speed. In an ideal scenario where the UAV starts at the correct height given by its path, it should be sufficient for the height controller to always output the current path time derivative in order to follow the path perfectly. Mainly because of disturbances, this will not work in practice.

The new height controller has a novel proactive design based on the derivative of the desired path, but with a PID part in addition. Fundamentally, the regular PID controller is for set-point regulation. By letting the set-point vary with the desired height from the path, it can be used for path following. The derivative of the desired height lays the foundation for the controller output, but the PID makes additional adjustments based on the height error. Algorithm 2 presents a simplification of the height controller.

```
initialize PID tuning values $K_p$, $K_i$ and $K_d$;
loop
    [altitude, position, speed] = getUAVEstimates();
    altitudeDesired = getAltDes(position);
    error = altitudeDesired - altitude;
    integral = integral + error*$\Delta t$;
    derivative = (error - prevError)/$\Delta t$;
    prevError = error;

    lookahead = speed*lookaheadTime;
    climbRateRatio = getAltDesDerivative(position-lookahead);
    climbRateDesired = speed*climbRateRatio;

    climbRateDemanded = climbRateDesired +
    ($K_p$*error + $K_i$*integral + $K_d$*derivative);

    climbRateDemanded = saturate(climbRateDemanded);
    climbRateDemanded = antiWindup(climbRateDemanded);

    dispatch(climbRateDemanded);
end
```

Algorithm 2: Proactive PID controller for height

The benefit of not only basing the controller on PID is that it reacts faster when the path changes. Instead of waiting for the reaction from an increase in the height error, the desired climb rate will react directly to the change of the desired height derivative.

Nevertheless, the controller will still have limitations. Most prominent is probably the lag time [22]. This is a small time delay from the issuing of a commanded climb rate to the actual execution, caused by computational delay and physical constraints like the discussed curvature discontinuity. It will make the UAV lag behind at the circle segments of the Dubins path. The solution is to not use the current desired height derivative as the foundation for the controller output, but the desired height derivative at a forward sighted lookahead time, outweighing said time delay. This is illustrated in Figure 4.17. Thus, the PID controller becomes proactive. It intervenes to outweigh the lag time and prepares for the expected change in desired climb rate.

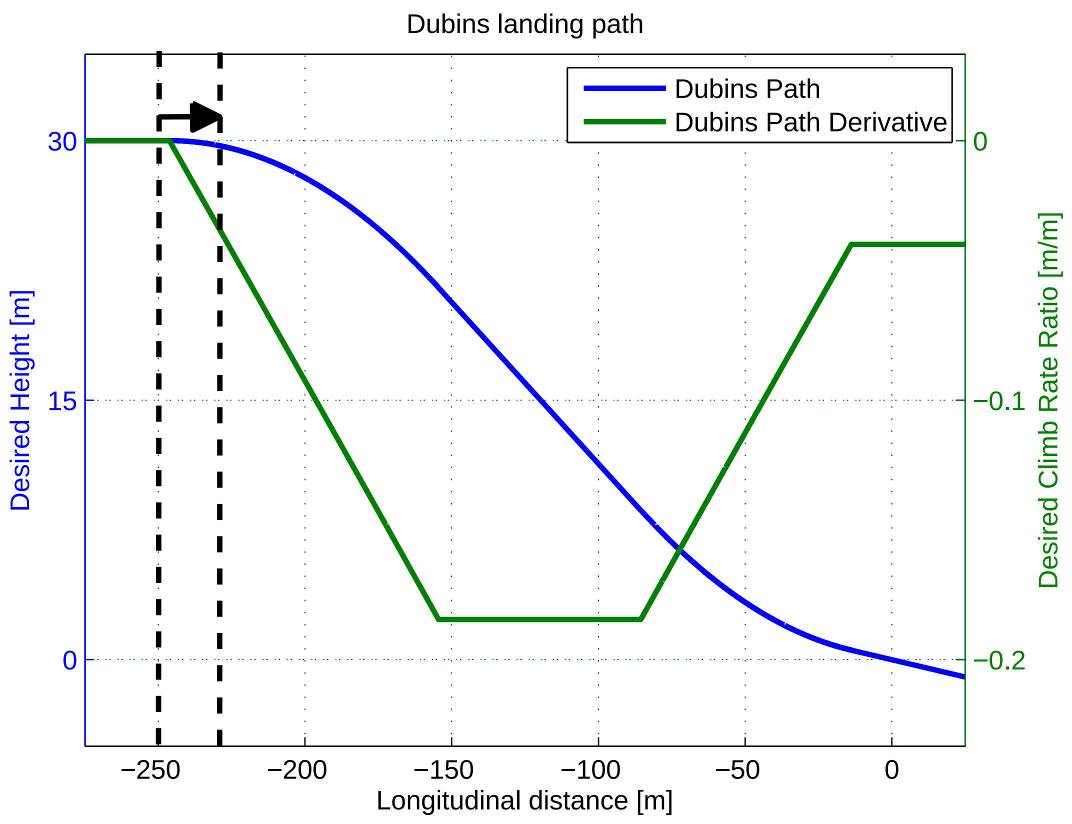


Figure 4.17: Demonstration of the lookahead distance for the proactive PID controller

Control of the UAV during the approach phase, with zero desired climb rate, compared to the glideslope/FA phase, with a negative desired climb rate, may require different tuning values. Appendix A.4 presents the PID tuning values for the height controller, with different gains for approach and descend, but with the same values for min/max climb rate, anti-windup and the PID lookahead time described above.

Speed control is done in APM:Plane using a controller called Total Energy Control System (TECS) [5]. It receives a new desired speed as a set-point at every waypoint. When the UAV hits the net, its propeller should already have quickly stopped and folded in order not to damage either the net or the UAV. This is ensured by designing the longitudinal guidance system to dispatch a brake message, called *IMC::Brake*, at a tunable ETA.

4.4.3 APM:Plane

The autopilot software for the Pixhawk, ArduPilot Mega Plane (APM:Plane), is used for low level control of the UAV. This is by using the flight mode called Fly-By-Wire B (FBWB). It is a mode for assisted flying where the UAV holds a roll angle and altitude/climb rate specified by the users control sticks when using a remote control transmitter [2]. In DUNE there is an *ArduPilot* task that converts roll angles and altitude rates into the pulse-width modulated (PWM) signals that resembles what APM:Plane would normally have received from the corresponding moves of the control sticks. All communication between DUNE and APM:Plane is done through the *ArduPilot* task, and the guidance and control system designed in this thesis is made to use the unaltered version of this task in order to ensure modularity.

By using the FBWB mode, low-level controllers in APM:Plane are utilized, while the high-level guidance laws can be implemented in DUNE for increased control. An overview of the software architecture is illustrated pictorially in Figure 4.18 below. The feedback needed from the Pixhawk is both the estimated position and attitude of the UAV. Based on the defined path and the feedback received from the Pixhawk, the guidance task calculates a desired climb rate and roll angle, in addition to sending desired speed.

The design of the automatic landing system is made modular to be able to use the unaltered version of APM:Plane. This means the Pixhawk can easily be updated to another version or changed with another hardware/software, as long as it can handle the desired roll angle and climb rate that the guidance system dispatches.

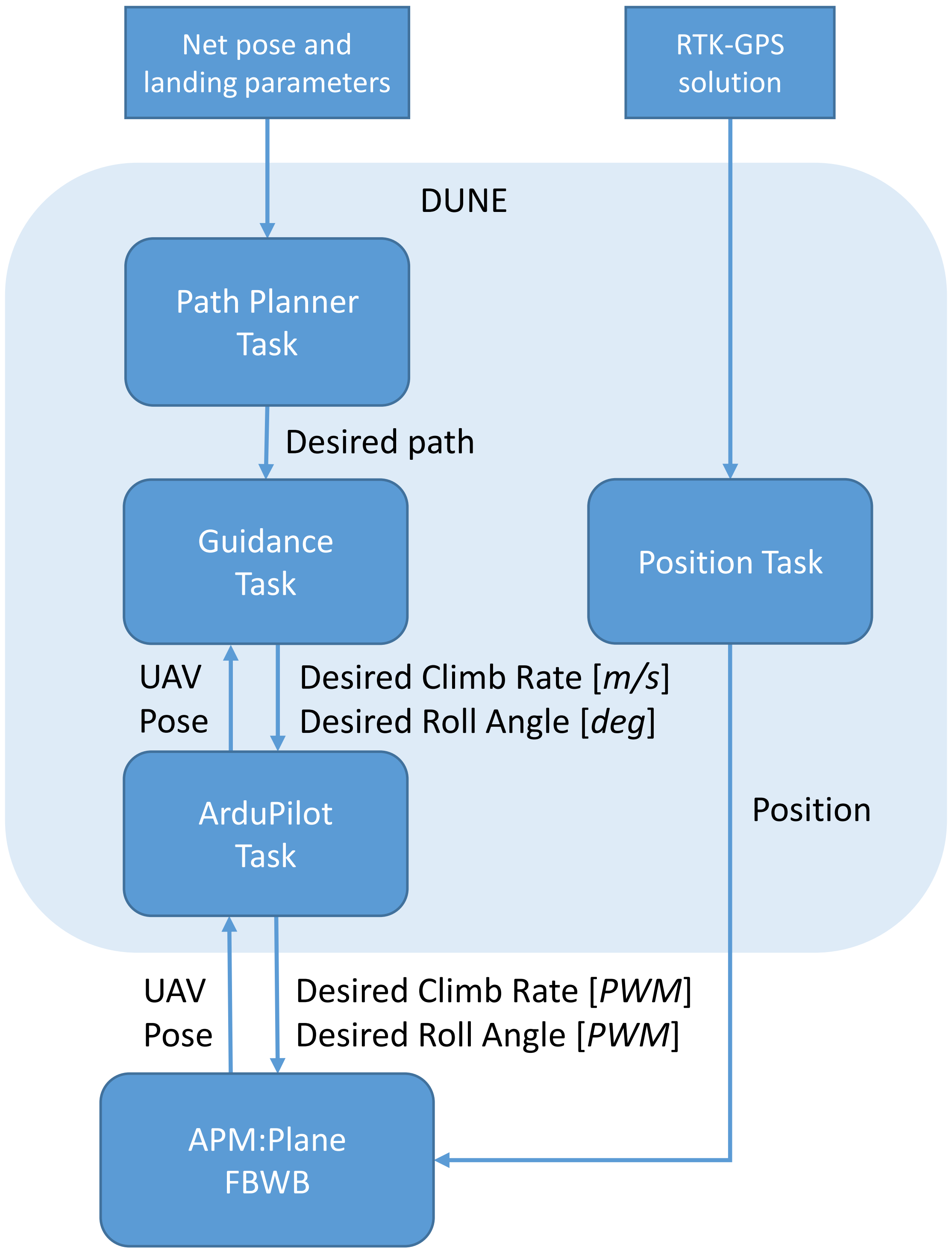

Net pose and landing parameters
RTK-GPS solution
DUNE
Path Planner Task
Desired path
Guidance Task
Position Task
UAV Pose
Desired Climb Rate [*m/s*]
Desired Roll Angle [*deg*]
ArduPilot Task
Position
UAV Pose
Desired Climb Rate [*PWM*]
Desired Roll Angle [*PWM*]
APM:Plane FBWB

Figure 4.18: Software architecture for climb rate and roll angle guidance

4.5 Landing Simulation, Software-In-the-Loop

Simulations allow extensive testing and tuning of the landing system without risk of damaging valuable equipment by prematurely testing in the real world. The UAV is meant to use the Pixhawk autopilot system. Therefore, tests of the controllers utilized by the Pixhawk, APM:Plane, are performed. Instead of doing a hardware-in-the-loop (HIL) simulation for UAV landing, software-in-the-loop (SIL) simulation was chosen. The reason for this is that a SIL simulation is much simpler to perform, while still using the same controllers as if a HIL simulation was performed. Thus, both the performance of the controllers and feasibility of the flight path can in large be concluded from this simpler test. In addition, there would not have been possible to perform a thorough enough HIL simulation to make a big difference. Sensor and actuator data would have had to be simulated and fed to the Pixhawk using the MAVLink protocol, instead of UART and I^2C as the real system uses. This implies additional time delay and processing for the Pixhawk, rendering the HIL simulation unrealistic.

4.5.1 Simulation Softwares

To perform SIL simulations, an additional software to the ones described in Section 3.1 is needed. The simulation software of choice is Jon S. Berndt Simulator (JSBSim).

JSBSim is an open source, platform-independent, software library conceived in 1996 [8]. Its purpose is to be a lightweight, non-linear, six-degree-of-freedom (6DOF) simulation application that is aimed at modeling flight dynamics and control for aircraft. This flight dynamics model is a good tool for preforming SIL and HIL simulations, as it defines the movements of the aircraft under the forces and movements applied to it from both control mechanisms and simulated nature.

Many different softwares need to work together to perform the APM:Plane SIL simulation. This is summarized pictorially in Figure 4.19. DUNE is the software responsible for handling of guidance, and feeds desired climb rate, roll angle and speed to the APM:Plane controller. This is done through a task called *ArduPilot Task* which includes low-level drivers for communication with the APM autopilot softwares. It is also responsible for forwarding of messages between APM:Plane and Neptus to display updated information about the UAV.

DUNE uses IMC messages both internally and when communicating with Neptus through UDP. This is similar to when DUNE is running on the payload computer and Neptus is running on the base station computer, with communication through a telemetry link. APM:Plane also uses UDP to communicate with JSBSim, the flight dynamics simulator. To initialize the UAV and give commands to Neptus, either MAVProxy or Mission Planner can be used. MAVProxy was used for this simulation because it is lighter on CPU than the alternative. It connects to APM:Plane using the MAVLink protocol.

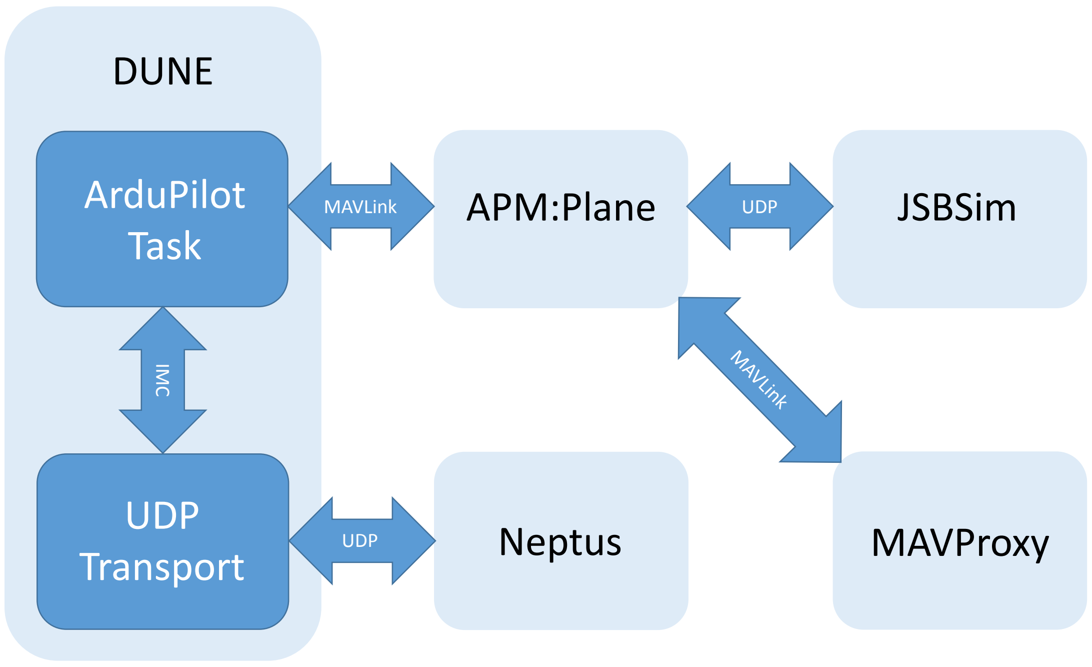


Figure 4.19: System architecture of the APM:Plane SIL simulations

4.5.2 Neptus Automatic UAV Landing Plug-In

To make it easier for an operator to request an automatic UAV landing, a new plug-in was developed for the command and control software Neptus. It allows the operator to define where the landing net is placed by either marking it directly on a map or by entering latitude and longitude coordinates for a more accurate placing. In addition, all the other settings defining the landing path can be changed by right clicking and selecting *Settings* from the appearing context menu, as showed in Figure 4.20. This includes net orientation, descend and attack angle, ignore evasive option etc. All the changeable parameters can be seen in Figure 4.22, showing the settings menu. When the net is placed, it appears as an arrow on the map that makes it easy to inspect both the position and orientation of the net, before and during landing. The net arrow is illustrated in Figure 4.20.

When the net is placed, the settings are changed and the operator wants the automatic landing to start, the operator right-clicks anywhere and selects *Start land plan* from the appearing context menu. Actions will then be taken to automatically generate and execute the landing path. If the net heading gets updated during landing, i.e. dynamic path, the arrow representing the net on the map will rotate correspondingly.

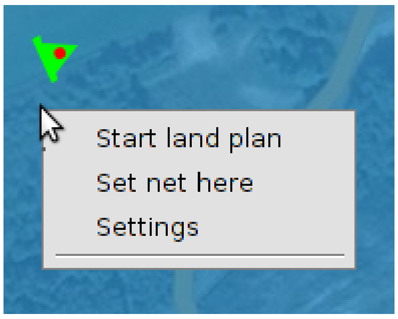


Figure 4.20: Neptus plug-in context menu (right-click) and net arrow icon

The plug-in is represented by an icon that is made to resemble the arrival signs seen at airports. This helps making it easy for new operators to recognize the landing plug-in. Figure 4.21 shows how the icon appears in the Neptus plug-in menu.

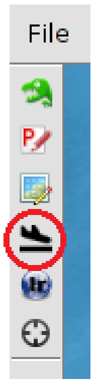
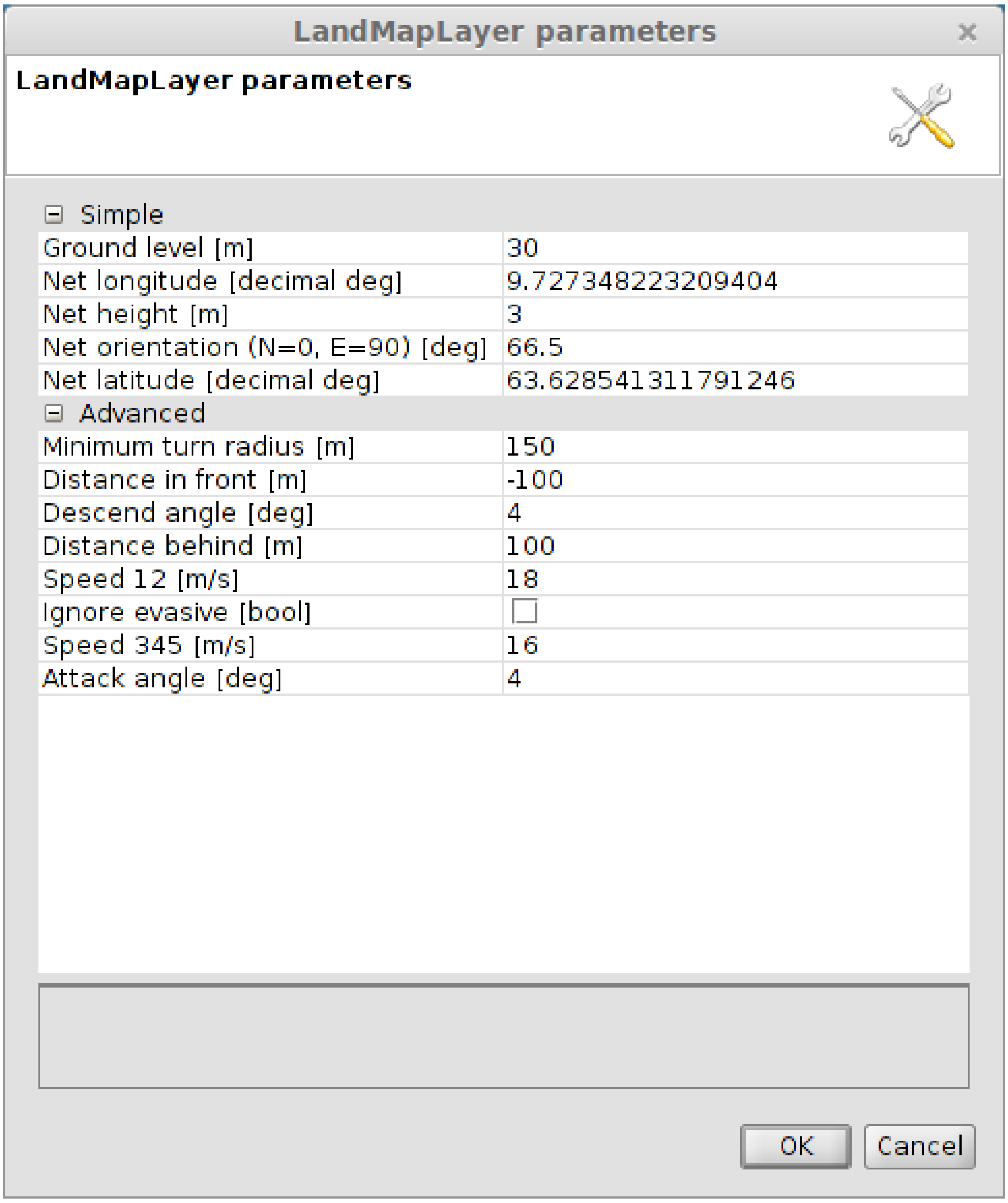


Figure 4.21: Plug-in menu with land icon marked in a red circle



Simple

Ground level [m]	30
Net longitude [decimal deg]	9.727348223209404
Net height [m]	3
Net orientation (N=0, E=90) [deg]	66.5
Net latitude [decimal deg]	63.628541311791246

Advanced

Minimum turn radius [m]	150
Distance in front [m]	-100
Descend angle [deg]	4
Distance behind [m]	100
Speed 12 [m/s]	18
Ignore evasive [bool]	☐
Speed 345 [m/s]	16
Attack angle [deg]	4

Figure 4.22: Neptus plug-in parameter settings menu

4.5.3 Landing Path Implementation

The landing path is calculated in DUNE, but receives the fundamental settings of the path from the Neptus plug-in. An operator only needs to be concerned with the Neptus plug-in. Below follows a description of the program flow for the new DUNE implementation made for processing of the landing path, which is illustrated in Figure 4.23.

All the landing parameter information is sent with an IMC message over UDP from Neptus to the DUNE task *Plan.Generator_DubinsPath*. The message type used is a standard message type called *IMC::PlanGeneration*, with the path settings added as parameters. When *Plan.Generator_DubinsPath* receives this message, it uses the parameters to calculate all the waypoints and the Dubins path.

The lateral path (and desired speeds) are defined only using waypoints, as it is a straight line path. Waypoints in DUNE are defined by the message type *IMC::Goto*. The longitudinal path, meaning the Dubins path, is calculated on the basis of waypoints, but with information about the position and size of the circle segments calculated in addition. All the Dubins path information is later retrieved with a *get*-function by the height controller, that utilizes it to calculate both the desired height and desired climb rate at every distance from the landing net.

To construct a landing path from individual waypoints, they are collected in a list of the type *IMC::MessageList<IMC::Maneuver>*, where *IMC::Maneuver* is a super-class that includes *IMC::Goto*. This maneuver list is transformed into an *IMC::PlanSpecification*, and in the end into an *IMC::PlanDB*. This is the waypoint path message being dispatched from the task *Plan.Generator_DubinsPath*. Other tasks in Neptus and DUNE consumes this message to make sure that the plan is displayed in Neptus and to guide the UAV.

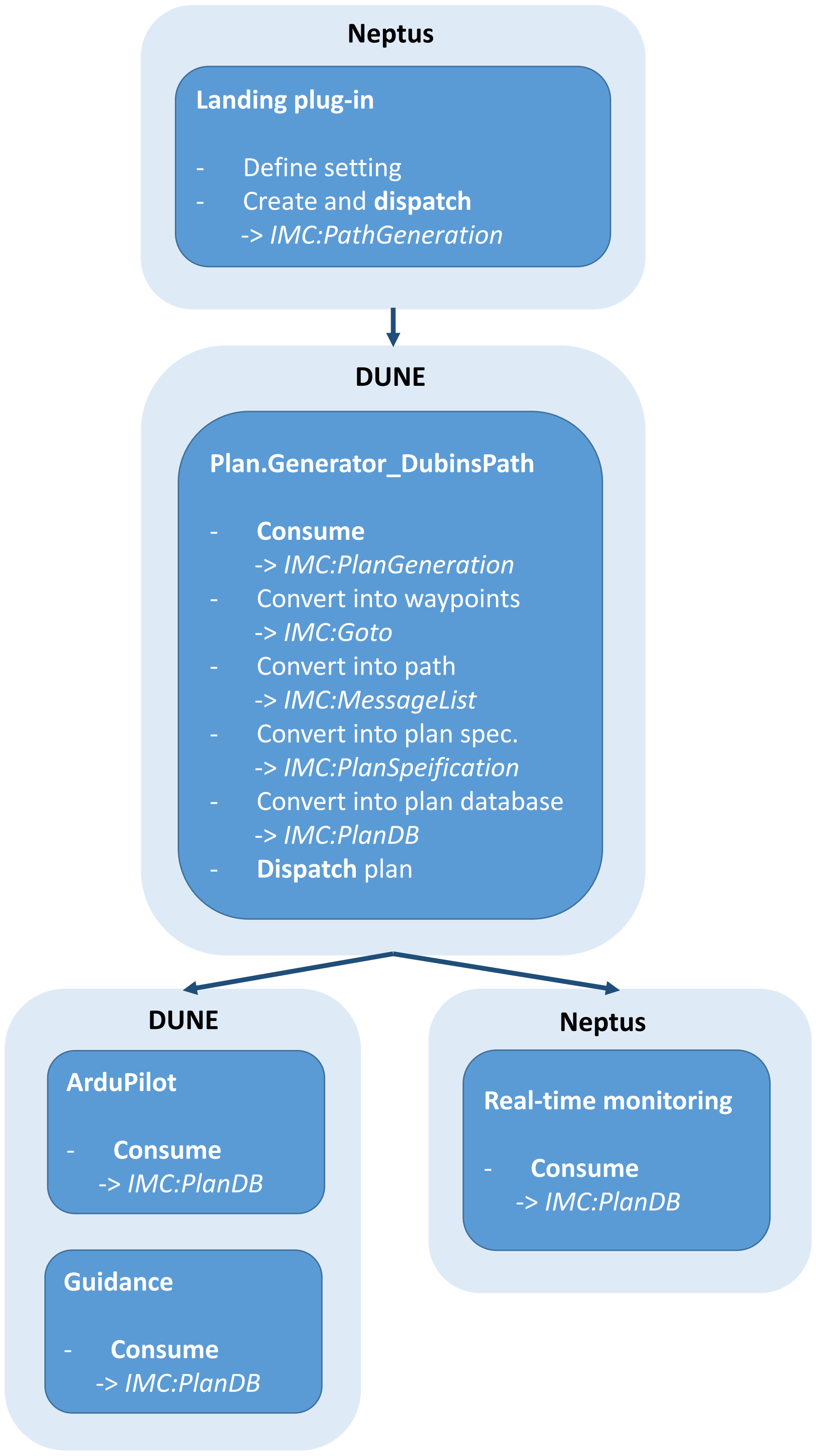
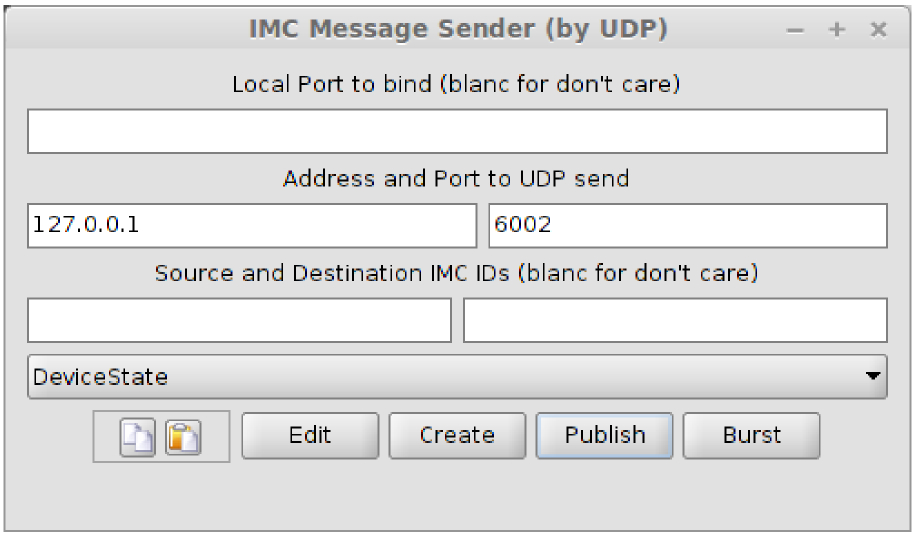


Figure 4.23: Overview of the landing path generation process

Dynamic path implementation

Since the ship motion simulator is in Matlab while the UAV flight simulator is in JSBSim, they are decoupled and can not be used together. To simulate a change in the predicted ship pose, leading to a change in the landing path, the new net pose message is sent manually. From Neptus it is sent as the IMC message type *IMC::DeviceState* using the already present Neptus tool *IMC Message Sender* illustrated in Figure 4.24. This message is received by the newly made DUNE task *Plan.DynamicLanding*. The message is defined to contain the predicted displacement in the net position and the predicted net orientation. Further, an *IMC::PlanGeneration* message with id *land_dynamic* is automatically dispatched from *Plan.DynamicLanding* to *Plan.Generator_DubinsPath*, where the old path is updated and processed as previously explained. Figure 4.25 demonstrates a simplistic view of the program flow.


IMC Message Sender (by UDP)

Local Port to bind (blanc for don't care)

Address and Port to UDP send: 127.0.0.1 | 6002

Source and Destination IMC IDs (blanc for don't care)

DeviceState

Figure 4.24: The Neptus tool *IMC Message Sender*

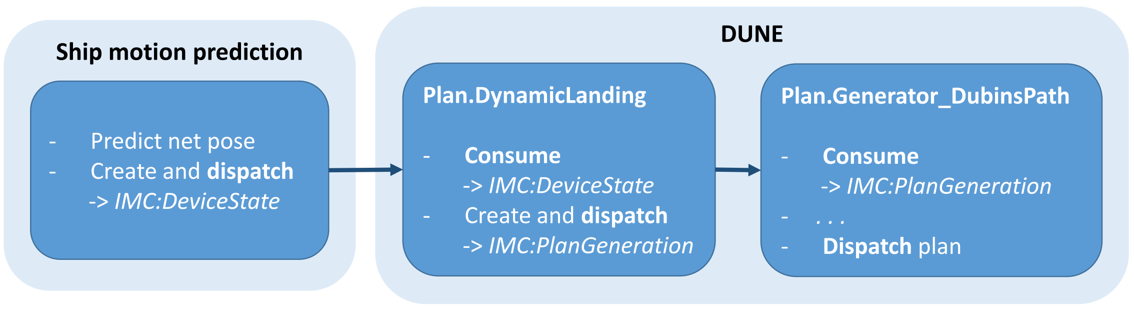

Ship motion prediction
- Predict net pose
- Create and dispatch -> IMC:DeviceState
DUNE
Plan.DynamicLanding
- Consume -> IMC:DeviceState
- Create and dispatch -> IMC:PlanGeneration
Plan.Generator_DubinsPath
- Consume -> IMC:PlanGeneration
- . . .
- Dispatch plan

Figure 4.25: Overview of the dynamic landing path generation process

4.6 Accurate Navigation

When landing in a net there is a relatively small position error tolerance. Regular navigation using Global Navigation Satellite System (GNSS) have to expect several meters of position error. Far more accurate navigation with the use of low-cost RTK-GPS systems is an ongoing research area, where students at the NTNU UAV-Lab are researching both standalone systems (Piksi [76]) and calculations done in DUNE (RTKLIB [77]), with and without IMU integration. The theory behind RTK-GPS and DUNE integration is omitted in this thesis, but interested readers are directed to a 2015 NTNU MSc thesis by Spockeli [75].

When the Pixhawk is used as autopilot system, it has a regular GPS receiver connected to it for position estimation. To fully take advantage of the increased RTK-GPS position accuracy that can be calculated in DUNE, a soft- and hardware setup was developed as part of this thesis to directly connect the calculations to the Pixhawk. By connecting a cable from the embedded computer running DUNE to the GPS input connector on the Pixhawk, it can process the incoming signal as it would with any other GPS receiver. This means no alterations of the Pixhawk software is needed, and it can easily be replaced with other autopilot software and/or hardware systems.

Figure 4.27 presents a picture of the Universal Asynchronous Receiver/Transmitter (UART) cable that was made to connected to the Pixhawk. In the other end it is connected to a computer running a DUNE task created to consume RTK-GPS messages (or other navigation messages), convert it to a National Marine Electronics Association (NMEA) format and finally send the command over UART to the Pixhawk. The NMEA format used is GGA. This is the most common GNSS sentence that includes 3D location and accuracy data. For more information about NMEA and the GGA sentence the interested reader is directed to the NMEA web page [60].

The new task that was created, called *Sensors.RtkGps2UartGps*, takes advantage of a DUNE NMEA parser when setting all the parameters defining the GPS message. After the correct sentence is created, a serial port DUNE class is utilized to construct and send the GPS sentence over UART to the Pixhawk.

The described setup has been tested with a HIL simulation, which strongly resembles the SIL simulation described in Section 4.5. The difference is that the autopilot APM:Plane now runs on the physical Pixhawk, similar to during real flights, instead of on a desktop computer. Figure 4.26 presents a simplified flow chart of the simulation setup. The processing done in JSBSim, Neptus and the Pixhawk is unchanged compared to what was described in Section 4.5, while the DUNE task and physical connection between the task and the Pixhawk is what is new in this test.

A UAV was simulated on a desktop computer using JSBSim. Through a USB cable it communicates with the autopilot software (APM:Plane) on the Pixhawk. The simulated position of the UAV was consumed as a regular GPS message (*IMC::GpsFix*)

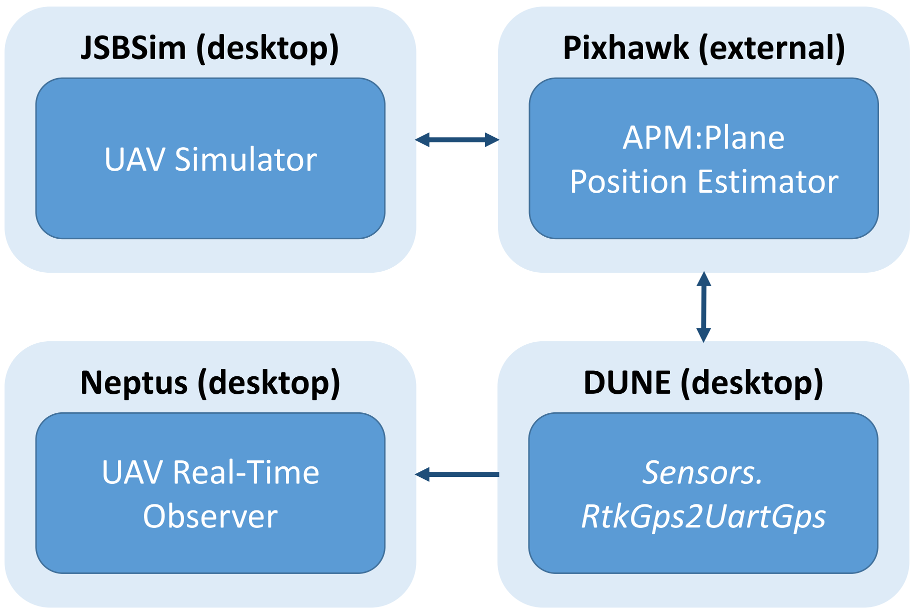


Figure 4.26: Flowchart of RTK-GPS-Pixhawk HIL simulation. Software from the desktop computer (desktop) is connected to the external Pixhawk autopilot (external)

by the newly made DUNE task *Sensors.RtkGps2UartGps*, converted to the correct NMEA-GGP format and sent from another USB port on the desktop computer to the Pixhawk using the custom made UART cable. A RTK-GPS IMC message is under development at the NTNU UAV-Lab, and will eventually replace the GPS message currently being consumed. As JSBSim moved the UAV, the correct position was successfully received by the Pixhawk from DUNE, and forwarded back through DUNE to Neptus that displayed the movements in real time.

When the *Sensors.RtkGps2UartGps* task was modified to always send a constant GPS position to the Pixhawk, ignoring the simulator that continued to dispatch messages of UAV motion, it was verified using Neptus that the Pixhawk thought the UAV had stopped at that position. This ensures that there were no connection errors in the HIL simulation, as the Pixhawk successfully received the static UAV position message through the UART cable, independent of the position message dispatched by JSBSim.

The conclusion is that the presented method provides a solution for how accurate navigation calculated in DUNE (like with RTKLIB) can be sent to the Pixhawk autopilot system without any need for alterations in the latter's software. The only thing needed is to run the DUNE task *Sensors.RtkGps2UartGps* and connect a UART cable from the embedded computer to the GPS input connector on the Pixhawk.

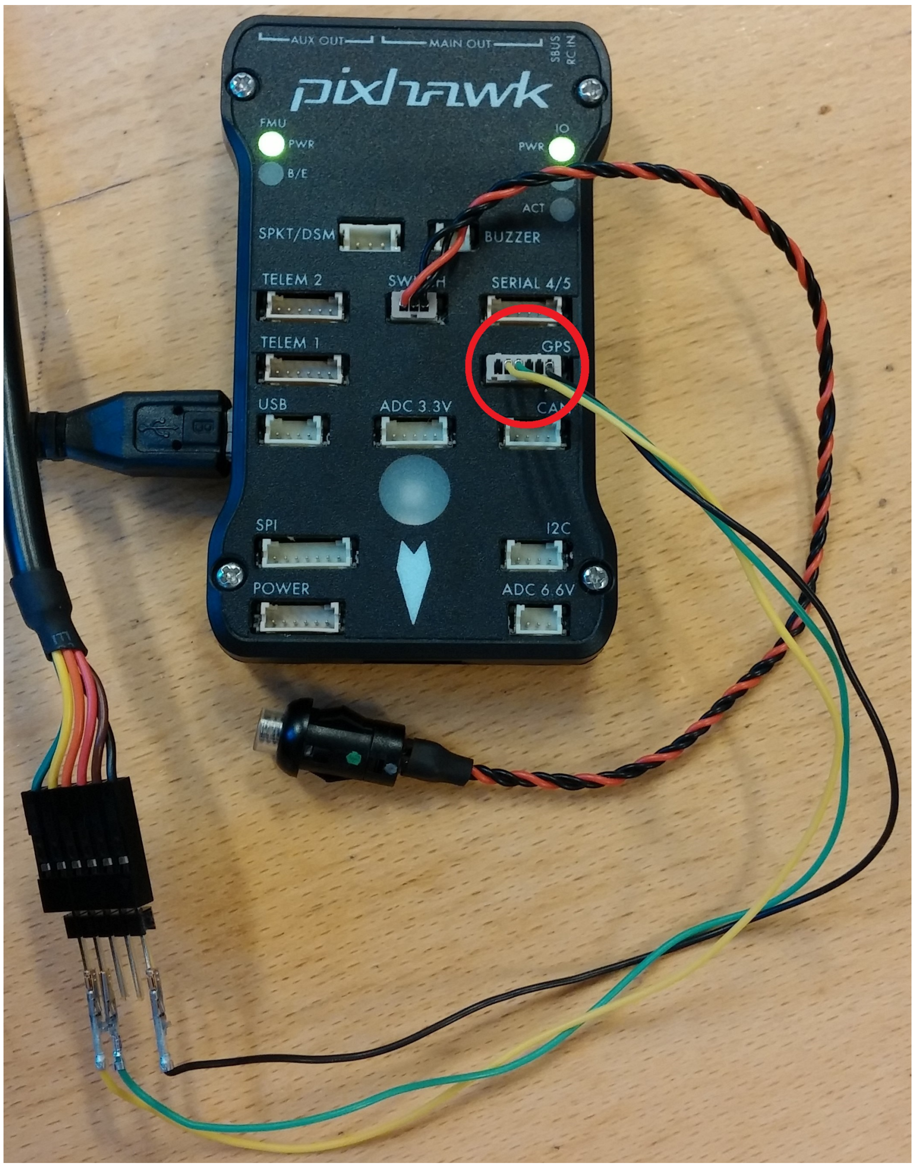


Figure 4.27: RTK-GPS-Pixhawk integration

Chapter 5

Experimental Testing

This chapter presents the experiments performed and results obtained through various simulations for time series forecasting in Section 5.1 and UAV landing in Section 5.2.

5.1 Time Series Forecasting

This section presents tests and discusses results from experiments of the time series forecasting methods presented in Section 4.2. This includes both the general case of ANN forecasting in the presence of oscillations around a varying mean, and ANN ship motion prediction of yaw and heave based on simulated data.

For all computations, Matlab version 8.3.0.532 (R2014a) computer package with the neural network toolbox version 8.2 was used. To incorporate the time series forecasting with the rest of the UAV landing system, ANN can be implemented in DUNE from scratch or using a C/C++ library.

5.1.1 Forecasting Criteria

When forecasting the yaw and heave motion of the ship for UAV landing, a prediction step of up to 60 seconds is advantageous, with 30 seconds as a recommended minimum. As a measurement to evaluate this ability, the following performance criteria are used:

Performance criteria

- Training time is situation dependent (on-line/off-line training)
- 90% of heave displacement predictions within 0.25 m
- 90% of heave displacement peak predictions within 0.25 m
- 90% of heading predictions within 5°
- 90% of heading peak predictions within 5°
- Mean absolute heave displacement prediction error below 0.15 m
- Mean absolute heading prediction error below 3°

When performing off-line training, the training time is to a large degree irrelevant. On the other hand, when training is decided to be done on-line, the time spent on a single training run is important in two ways. If training takes too long, the fully trained network will at worst already be outdated due to severe changes in the weather conditions. In addition, since forecasting methods like ANN does not have an intuitive optimal architecture for all situations, a short training time may give a forecasting system the chance to test multiple architectures on-line and retrain.

The performance criteria for heave displacement is based on the fact that a too high allowed error may lead the UAV to miss the net. At the same time there are already many sources of error making a too low heave prediction error criterion unnecessary, like navigation error, guidance error and error in the estimated time of arrival to the landing net. To compensate for this the landing net already has a size that gives the UAV a safety zone in both horizontal and vertical direction.

The heading plays an important role in the desired path for the UAV landing into the net. Influenced by waves, wind and current the heading of the ship may change while the UAV is descending towards the net. If the heading of the ship changes without updating the landing path, the UAV may miss the ship or hit parts of the ship standing in the way of the new acute angled path towards the rotated net.

In addition, if the net orientation changes, the target area of the net becomes smaller. The horizontal width of the projected net area can be calculated as a function of the horizontal angle of approach according to Equation 5.1:

$$W_p = W_{net} \times \cos(\alpha) \tag{5.1}$$

where W_p is the projected width, W_{net} is the actual net width and α is the angle between the normal of the net and the approach angle.

The heading of a ship with limited actuations does generally not move that many degrees in a minute. Even though it may oscillate due to waves, these oscillations usually have a low amplitude. The performance criteria set for heading prediction is relatively low, although a UAV landing system could handle a larger heading error.

5.1.2 Oscillations Around a Varying Mean

To demonstrate clearly the problems discussed in Subsection 4.2.3.1, a test is performed on the yaw motion example depicted in Figure 5.1. It is a sine wave with constant amplitude of 5°, oscillating first around a mean of −5°, before rising to a mean of 125° following a sigmoid curve. The exact equation and implementation can be seen in Listing B.1 in Appendix B.1.

Including the standard no-processing method used as reference, the following novel solutions presented in Subsection 4.2.3.2 will be tested:

- *Unchanged*
- *Mean*
- *Median*
- *Current*

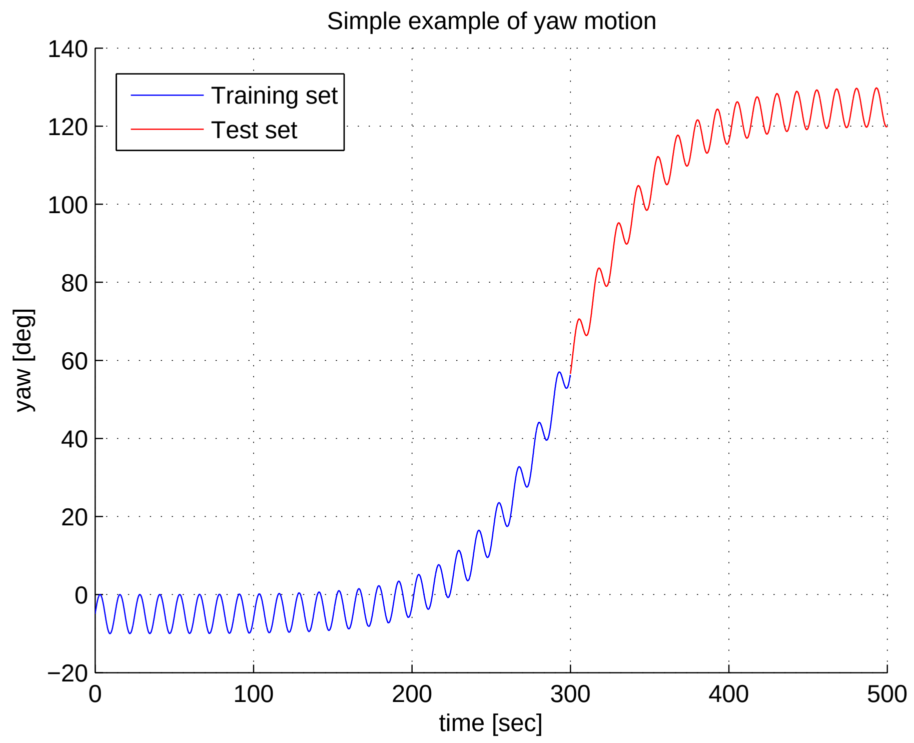


Figure 5.1: Simple yaw motion, split into training set (300 seconds) and test set (200 seconds)

- *Normalization*

All the five ANNs constructed for the different methods have the same settings, only varying the pre- and post-processing. The prediction step is 10 seconds ahead, and 4 samples from the previous 8 seconds are used as inputs to the network. In the only hidden layer there are 5 hidden nodes. All nodes have the linear activation function, including the single output neuron. The training method used is Levenberg-Marquardt back-propagation, which is an offspring of the BP algorithm described in Subsection 2.3.1.4. For more information about the Matlab implementation of the training algorithm, the reader is directed to its web page [53].

Figure 5.2 illustrates a combined plot of the outputs from the five ANNs, together with the target function. It is evident from this figure that the *unchanged* method, meaning the one with no pre- and post-processing, does not manage to sufficiently approximate the underlying function and perform time series forecasting. On the other hand, all the other methods perform well. Detailed information about the performance can be found in Table 5.1. Several alternative network configurations have also been tested, like changing the number of hidden nodes and activation functions in the hidden layer, but the results are always similar.

Focusing on the performance of the four most effective methods, *median* and *current* gives the smallest error, *mean* follows just behind and *normalization* has a slightly

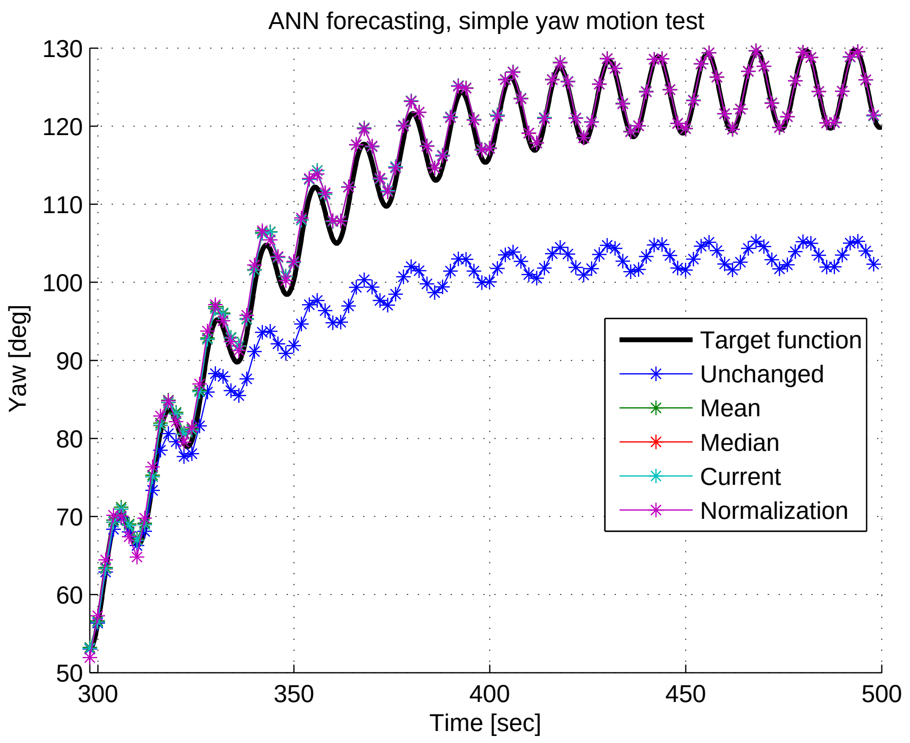


Figure 5.2: ANN test on a simple yaw motion, demonstrating performance of different methods. *Mean*, *median*, *current* and *normalization* are almost completely overlapping on top of the target function

larger error, albeit still functioning well. This coincides with the theory presented in Subsection 4.2.3.2. The first three methods only shifts the values in the input vector, while the latter also scales the values. The additional scaling makes it harder for the ANN to distinguish the differences in the input vectors, hence reducing the forecasting performance.

The training time of the *unchanged* method is more than 2.5 times more than that of the second longest training time. This confirms the theory presented about how data spread around zero and more evenly scaled data speeds up the training process. *Normalization* has the lowest training time, as all values in the input vectors gets normalized to lie between -1 and 1. *Mean* and *median* are almost having the same effect of spreading the values in the input vector around zero. *Current* is not as effective in this as the previous three methods, hence the training time is slightly longer.

Table 5.1: Results from simple yaw motion test

Methods	Training time [s]	Errors within 5° [%]	Errors at peaks within 5° [%]	MAE [°]	MSE [°]
Unchanged	0.390	16.8	16.1	14.89	278.129
Mean	0.125	100.0	100.0	0.998	1.600
Median	0.125	100.0	100.0	0.957	1.517
Current	0.148	100.0	100.0	0.959	1.523
Normalization	0.117	100.0	100.0	1.022	1.722

5.1.3 Discussion - Oscillations Around a Varying Mean

Because the setup with no preprocessing does not compensate for the varying mean, the neural network does not manage to cope with oscillations around a mean it has not been trained for. This is the problem of extrapolation. For all other processing methods, the values in the input vectors are altered to be less dependent on the original mean.

The conclusion of this test is that the extrapolation problem of unknown means is significant, but it can be reduced with some easy pre- and post-processing of the training and test data. All processing methods presented as possible solutions worked well for this constructed exampled, with *median* and *current* giving the smallest error.

Training time was also greatly affected by the different methods, with the unprocessed data slowing down the training time notably. *Normalization* gave the lowest training time, followed by *mean* and *median.* This supports the earlier presented theory.

5.1.4 Ship Simulation

To ensure results that are applicable in the real world, efforts have been made to get good data for the forecasting tests. The choice fell on the *Marine Systems Simulator (MSS)* toolbox for Matlab/Simulink® developed by Fossen and Perez [25]. It is used to simulate a six degrees of freedom (6DOF) ship influenced by waves and current.

The waves generated are based on an International Towing Tank Conference (ITTC) [32] spectrum type. Figure 5.3 illustrates a depiction of the sea state realization, and demonstrates the complexity of the waves. Significant wave height is 3 meters. Other settings used for the wave simulation can be found in Appendix B.2.

The ship models used are based on parameters from either a 175.0 m long container ship called *ShipX: S175* or a 82.8 m long supply vessel called *ShipX: Supply Vessel*

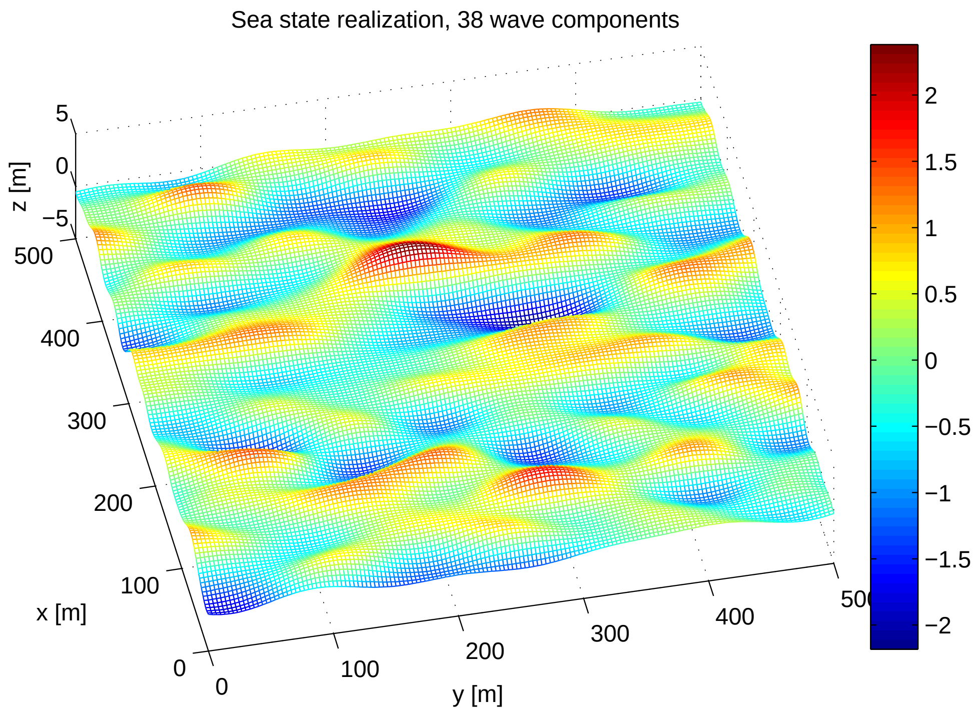


Figure 5.3: Simulated sea state realization, generated from MSS [25]

[23]. Further details of the ships can be found in Appendix B.2. For an introduction to the MSS toolbox the reader is directed to a paper by Perez et al. [63].

From the simulated ship models all of the 6DOF variables and their derivatives are observable, often referred to as the vectors η and ν [24]. The variables used in the following forecasting experiments are heave, roll, pitch and yaw. Figure B.2 in the appendix illustrates the power spectrum density (PSD) for the heave motion of ShipX: S175, describing how the power of the signal is distributed over the different frequencies. It has a wide peak centered at 0.1 Hz (0.63 rad/s), meaning that the most common period of the heave motion is about 10 seconds, albeit it is not constant.

In Section 4.2 a batch on-line training method was proposed. The training window was suggested to be 5 minutes. To replicate this forecasting setup during experiments, only the five first minutes (300 seconds) of the simulated data set is used for training. The separate test set used for performance measurements is the next 1000 seconds, following from where the training window ended. Using a test set that big ensures a good basis for performance measurements.

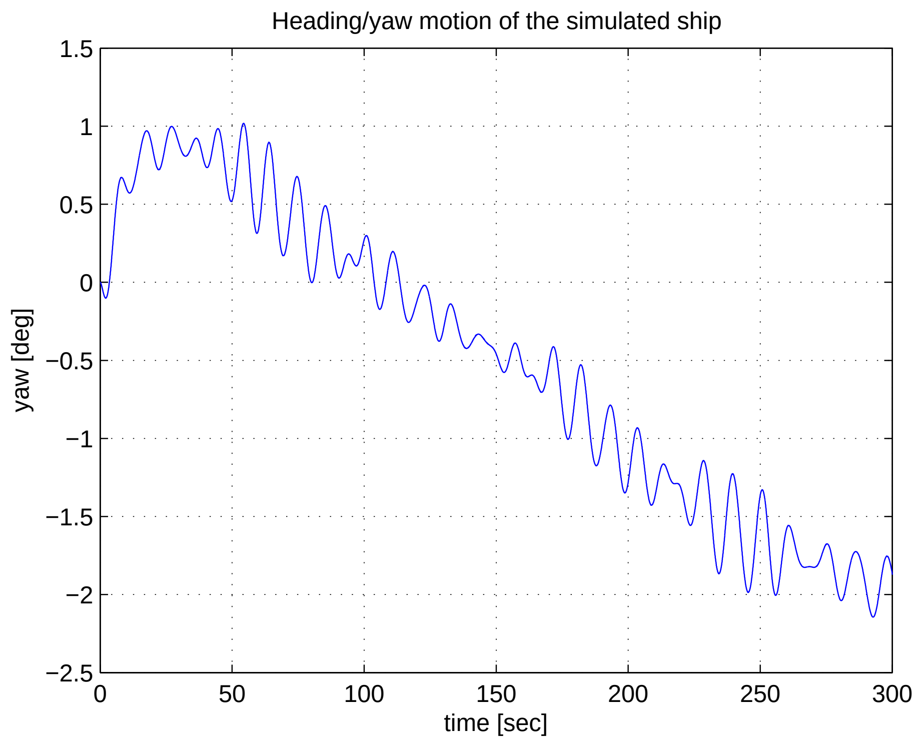


Figure 5.4: Yaw motion of the simulated ship S175

5.1.5 Ship Heading Prediction

This test tries to predict the future heading of ShipX: S175 using ANN. A section of the heading or yaw motion is illustrated in Figure 5.4. To give the UAV time to react properly and converge safely onto the newly recalculated path with an updated angle of approach, the prediction step should be far ahead. A prediction step of 60 seconds is tested. If this test is successful, the capability of shorter prediction steps follows as a consequence.

Subsection 5.1.2 above demonstrated with a simple example how no pre- and post-processing of the training and testing data may lead to failure in ANN forecasting. This is now tested on the more realistic data set. All the same processing methods are used.

The ANN architecture has many parameters to vary:

- Processing method
- Sample frequency
- Number of inputs
- Number of hidden layers
- Number of hidden neurons

- Activation functions

Because time series forecasting is a form of function approximation, linear activation functions in both the hidden and output layer is recommended. Other activation functions, like the hyperbolic tangent, could also have been used for the hidden layer. From a test performed on the simulated data, with results illustrated in Appendix B.3.1, it can be concluded that the linear activation function yields better performance for all the processing methods.

The number of inputs used as basis for further tests is chosen to be 10, with a sample frequency of 0.5 Hz. This gives the ANN samples from the previous 20 seconds.

Following Cybenko's theorem, only one hidden layer is used. Thorough testing shows that each method reacts differently to the number of hidden nodes. Figure 5.5 depicts a section of the results from a test where the number of hidden nodes was varied from 1 to 6. For the figure including *unchanged* and *normalization*, see Figure B.4. Each ANN variation (5×6) was ran 20 times, with the median value of those runs saved for performance comparison. The reason for doing multiple trials and taking the median is that ANN training is prone to local minima and every training ends up with slightly different network weights. To ensure a result that best captures the general performance, without being influenced by extremal values, median is a good method.

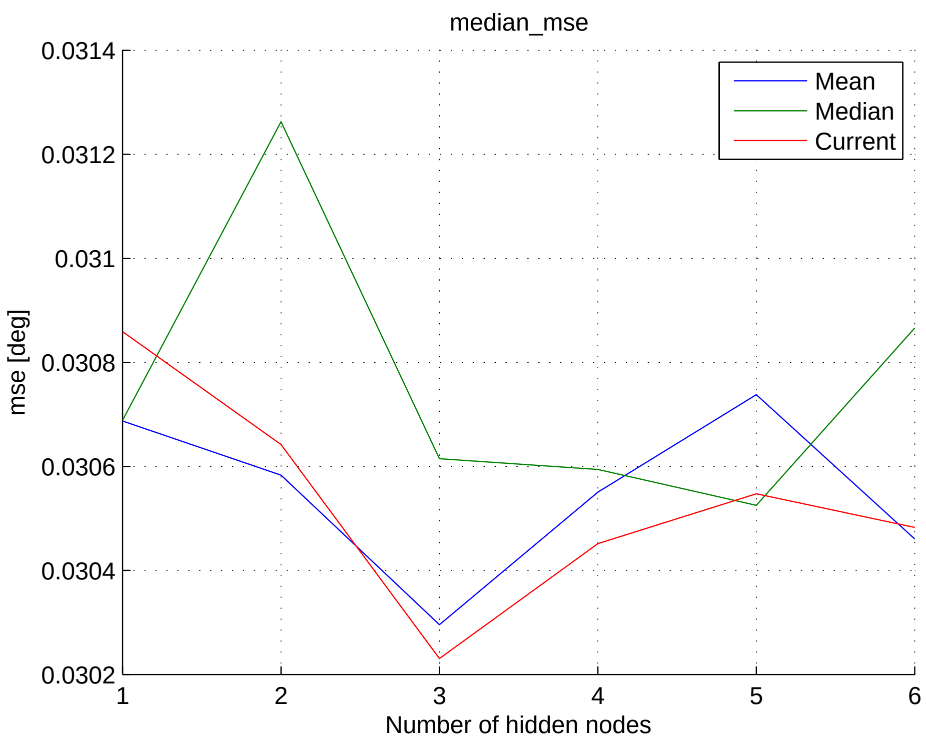


Figure 5.5: Ship heading prediction with different methods and numbers of hidden nodes. Only results for *mean*, *median* and *current* are presented

Looking at the computed training time, the time it takes to train the ANN on the training set, it is evident that the number of neurons also has a large impact for this. Figure 5.6 illustrates how the training time more than triples for all but one method when going from 2 to 3 hidden neurons.

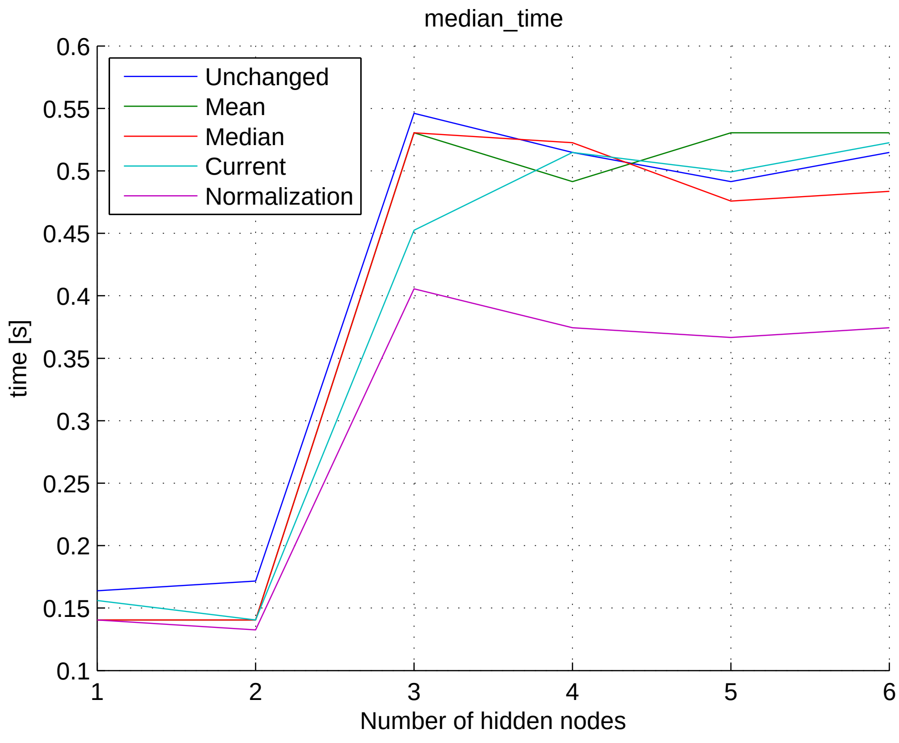


Figure 5.6: Training time for different methods and numbers of hidden nodes

The result of the tests shows that the optimal numbers of hidden neurons with regards to MSE are: 3 for *unchanged*, *mean* and *current*, 5 for *median* and 6 for *normalization*. If training time is highly important in the scale of less then half a second, which is not the case for the UAV landing system, one should limit the number of hidden neurons to a maximum of 2. For more detailed results see Appendix B.3.

Using the optimal settings found from the above tests, comparison of the forecasting performance for the different processing methods is performed. In Table 5.2 both the MAE and training time is presented for the 5 methods. All methods manages to predict the ship's heading 60 seconds ahead of time with an average error less than 0.3 degrees. A reason for the low errors is the slow movements of the ship's heading. It is still evident that *mean*, *median* and *current* outperforms *unchanged* with a factor of about $\frac{1}{2}$, in addition to also having a lower training time.

An example of the actual forecasts made by the ANN using the *median* method is illustrated in Figure 5.7 together with the corresponding target value.

Table 5.2: Ship heading prediction for 60 seconds

Method	Training time [s]	MAE [°]
Unchanged	0.546	0.288
Mean	0.530	0.143
Median	0.476	0.143
Current	0.452	0.142
Normalization	0.374	0.185

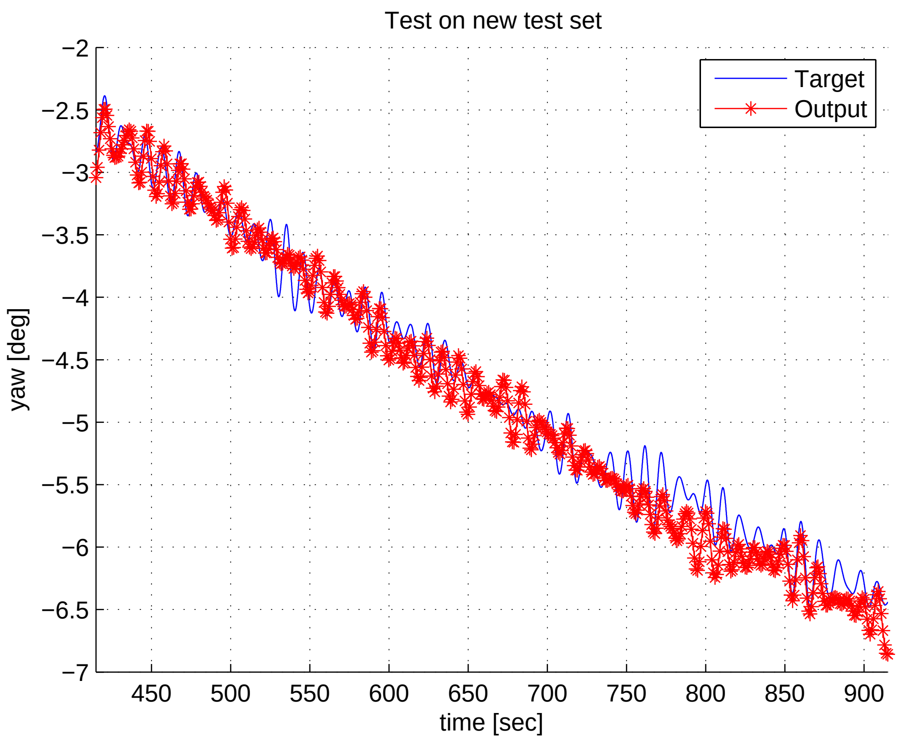


Figure 5.7: Ship heading prediction test using processing method *median*

5.1.6 Ship Heading Prediction - Supply Vessel

Due to the large size of the ship in the previous test, the heading changed very slowly. This test will use the smallest ship available for the MSS ship simulation, 82.8 m long ShipX: Supply Vessel. This is both to see if the performance criteria can be met with faster changes in the heading, and to test the capabilities of the presented ANN processing methods as forecasting tools for various ships. Figure 5.8 illustrates the simulated yaw motion.

Based on knowledge gained from the previous tests, the ANN settings are as follows:

- Prediction step: 60 s
- Processing methods: All (5)

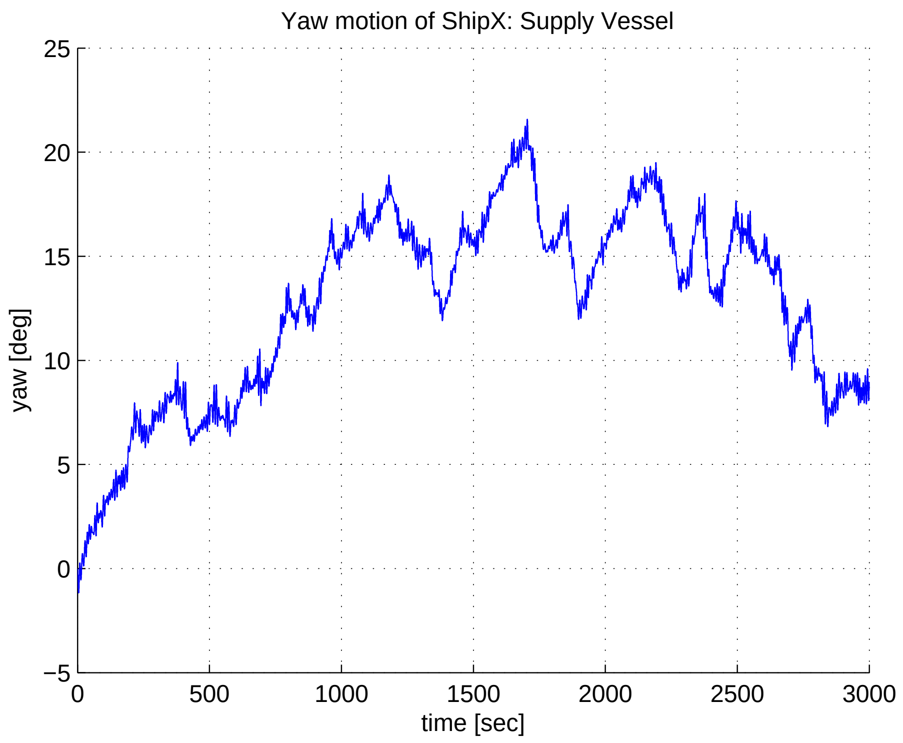


Figure 5.8: Yaw motion of the simulated supply vessel

- Sample frequency: 0.5 Hz
- Time range used for inputs: 60 s
- Number of hidden layers: 1
- Number of hidden neurons: 5
- Activation function: Linear
- Training time range: 300 s
- Test time range: 1000 s

After 20 trials for every processing method, the median of the MAE and training time were calculated and is presented in Table 5.3. Figure 5.9 illustrates an example of forecasting when using processing method *current*.

Even though the mean absolute error is about 1.4° for most of the processing methods and 100 % of the predicted values are within 5° of their target, the performance in Figure 5.9 looks somewhat inaccurate. It is important to keep in mind the added random nature of this simulation, i.e. the parts which no forecasting technique can predict.

Table 5.3: Ship heading prediction for 60 seconds, supply vessel

Method	Training time [s]	MAE [°]
Unchanged	0.437	1.635
Mean	0.398	1.304
Median	0.382	1.305
Current	0.390	1.214
Normalization	0.390	1.407

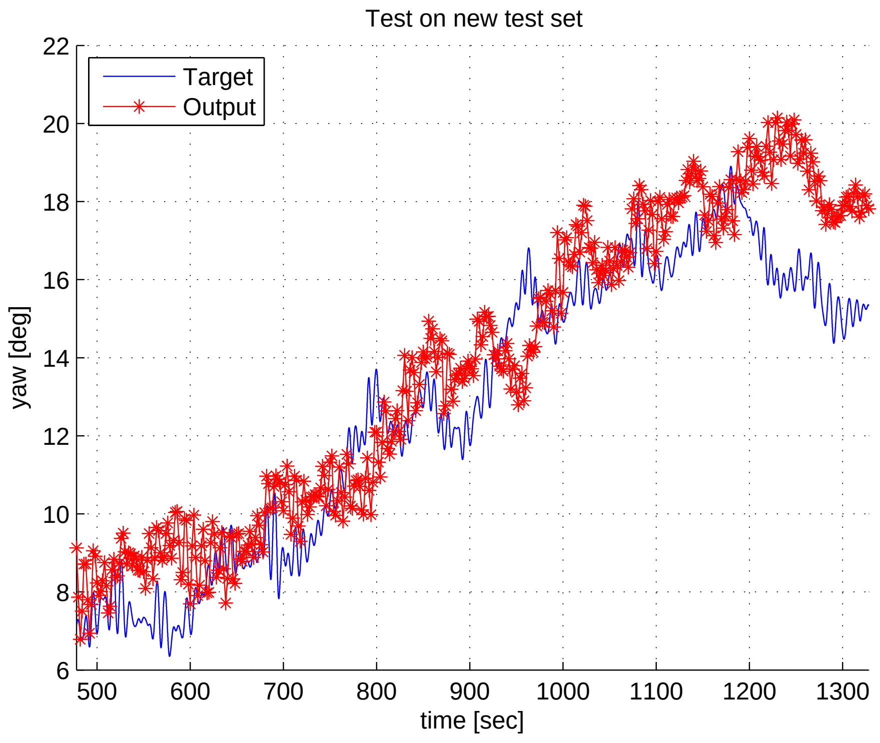


Figure 5.9: Ship heading prediction test for supply vessel using processing method *current*

5.1.7 Discussion - Ship Heading Prediction

The proposed prediction system managed to predict the simulated ships' heading with a low prediction error well within the criteria. This was for a prediction step of 60 seconds, and the only information required by the system was ship motion data gathered from the last five minutes. This was used to train the ANN.

The proposed pre- and post-processing methods all led to lower errors and shorter training times than the *unchanged* method with no individual processing of the data vectors. This finding is similar to the one from Subsection 5.1.3. Of all the methods tested, *current* stood out as the best alternative for ship heading prediction.

5.1.8 Heave Displacement Prediction

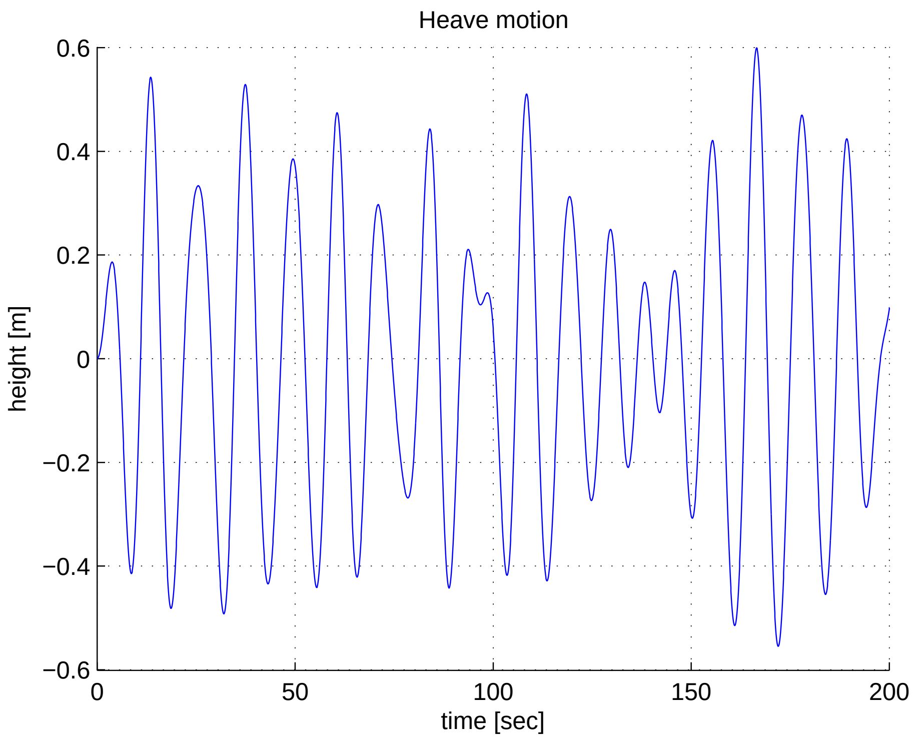


Figure 5.10: Heave motion of the simulated ship S175

When landing on a ship it is important to take into account the vertical movement of the landing target, called heave displacement. Figure 5.10 illustrates a section of the heave motion of the simulated ship used for the coming experiments, ShipX: S175. Depending on the size of the waves, the amplitude of the heave displacement may be big enough to make the UAV miss the landing net, i.e. fly over or crash into either the boat beneath the net or the water. To minimize this risk, a system for prediction of the landing target's heave displacement will be tested based on ANN.

Prediction is done in multiple steps. It is reasonable to assume that longer prediction steps will have greater errors. Weather conditions may change, measurement noise propagates with time and the estimated time of arrival (ETA) may be wrong. As the UAV approaches the net, the predicted heave displacement on impact will be updated. The prediction steps used are 5, 10, 20, 30, 40, 50 and 60 seconds. Because the heave displacement varies more and is more critical than the yaw motion, more prediction steps are tested. This is also to inspect the different ANN architectures needed for different prediction steps.

Because the heave motion of a ship already oscillates around zero, there is not the same need to preprocess the data going into the ANN. The methods used in the previous experiments are therefore omitted in the following tests, only the unchanged data is used. As with ship heading prediction, the linear activation function was found to give best results. Figure 5.11 illustrates how the linear function (*purelin*) outperforms the activation functions hyperbolic tangent (*tansig*), log-sigmoid (*logsig*) and radial basis (*radbas*).

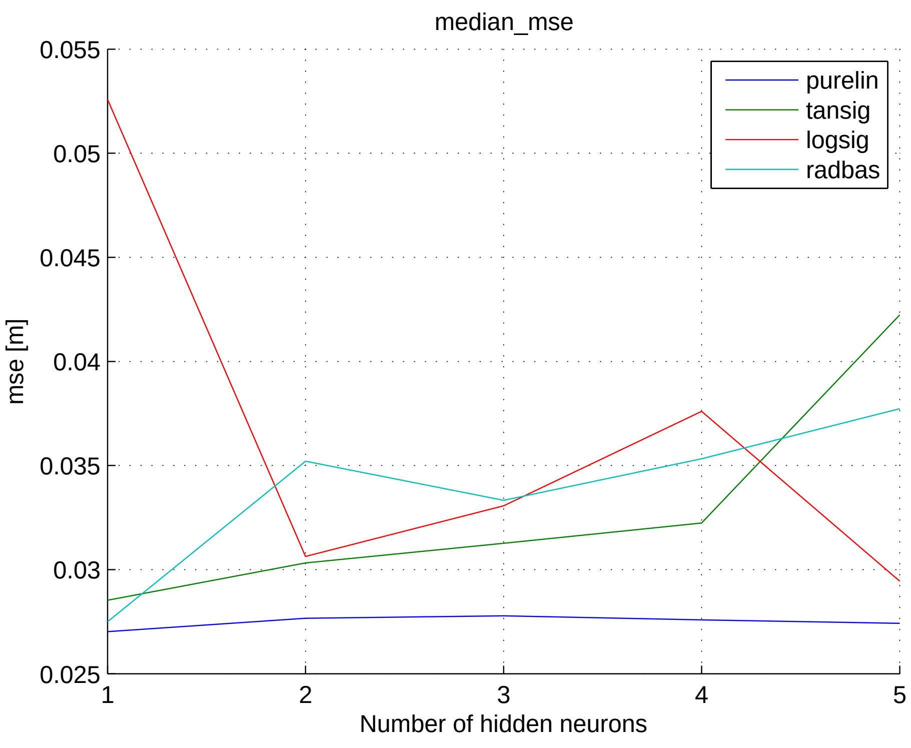


Figure 5.11: Heave displacement prediction MSE for different numbers of hidden neurons and different activation functions

To further investigate the optimal number of hidden nodes, a test with the prediction steps 10-60 s in 10 s intervals were tested with the number of hidden neurons varying from 1 to 10. It is clear from this test, as seen in Figure B.5 in the appendix, that an increase in the number of hidden neurons does not decrease the MSE noticeably in any of these cases. Hence, only one is used for the rest of this experiment.

When deciding the number of inputs it is chosen indirectly by adjusting two other parameters; the sampling frequency and the time range back in time that's sampled from. Each prediction step may have its own optimal combination, meaning for every step a test is preformed varying both parameters. A thorough test with 10 runs for each combination was conducted where the number of training vectors was 300 and the number of test vectors was 500. The prediction steps 5 second ahead and 10-60 s ahead in 10 s intervals were all examined. The different time ranges sampled from was the previous 6 s and previous 10-120 s in 10 s intervals, and the sampling frequencies tested were 0.5, 1.0, 1.5, 2.0, 4.0, 6.0 and 8.0 Hz. This means the number of inputs to the network ranged from 3 (6 s $\times$ 0.5 Hz) to 960 (120 s $\times$ 8.0 Hz) samples.

The performance measure used for comparison is the percentage of predictions within 0.25 meters of the target value. Table 5.4 presents the minimum input parameter combinations necessary to obtain 90 %, 95 % and 100 % of errors within 0.25 meters for each prediction step. Figure B.8 in the appendix presents the plot used to find the optimal combinations for the 60 seconds prediction step. A conclusion from looking at all the plots from these tests is that, in general, the accuracy of the prediction increases both with an increasing sampling frequency and an increasing range backwards in time that is sampled from.

Table 5.4: Prediction steps with their necessary combinations of input time range/sampling frequency

Prediction step [s]	Min. 90 % within 0.25 m [s/Hz]	Min. 95 % within 0.25 m [s/Hz]	100 % within 0.25 m [s/Hz]
5	6/0.5	6/0.5	6/0.5
10	6/0.5	10/0.5	20/0.5
20	20/0.5	40/1.0	80/2.0
30	50/1.0	60/2.0	50/8.0
40	70/1.0	120/1.0	40/8.0
50	70/0.5	80/2.0	80/8.0
60	120/1.0	120/1.0	120/4.0

Using the knowledge gained from all the preliminary tests above, evaluation of heave displacement prediction is performed. Performance measurements from tests on various prediction steps are presented in Table 5.5. The ANN architecture parameters used in these tests are for all prediction steps linear activation functions and 1 hidden neuron. The two input parameters used are based on the column in Table 5.4 containing the parameters that gave minimum 95 % of the predicted values within 0.25 meters of the target value.

To demonstrate the performance of the ANN forecasting system, Figure 5.12 and 5.13 illustrates two sections of predicted values 10 and 60 seconds ahead, respectively. It is clear that for the data set used, it is possible to perform satisfactory heave motion prediction 60 seconds ahead with a mean absolute error less than 10 cm.

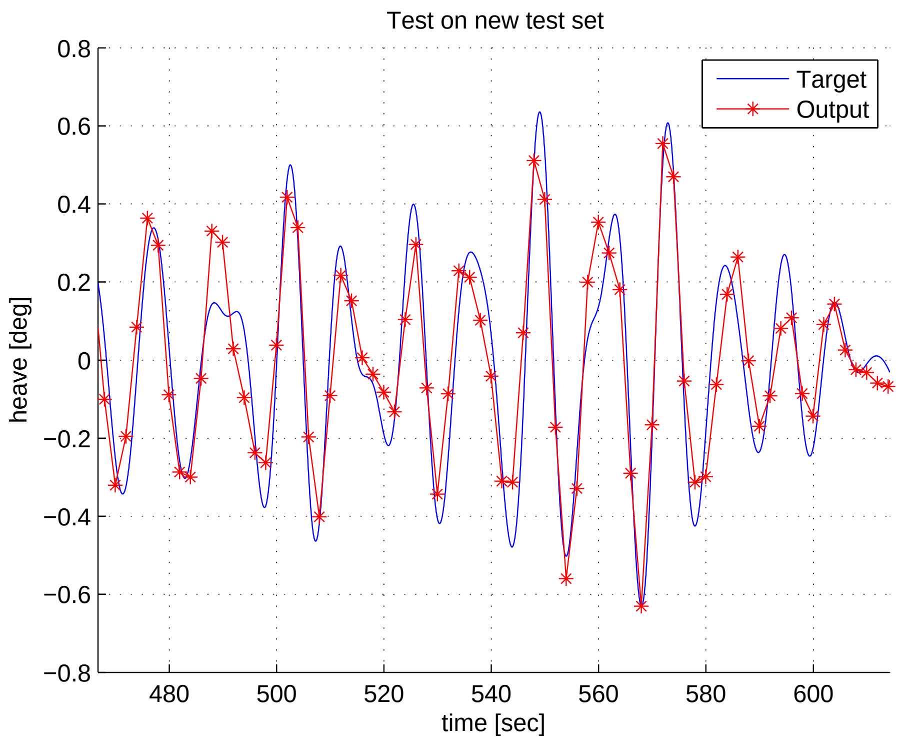


Figure 5.12: Heave displacement prediction 10 seconds ahead

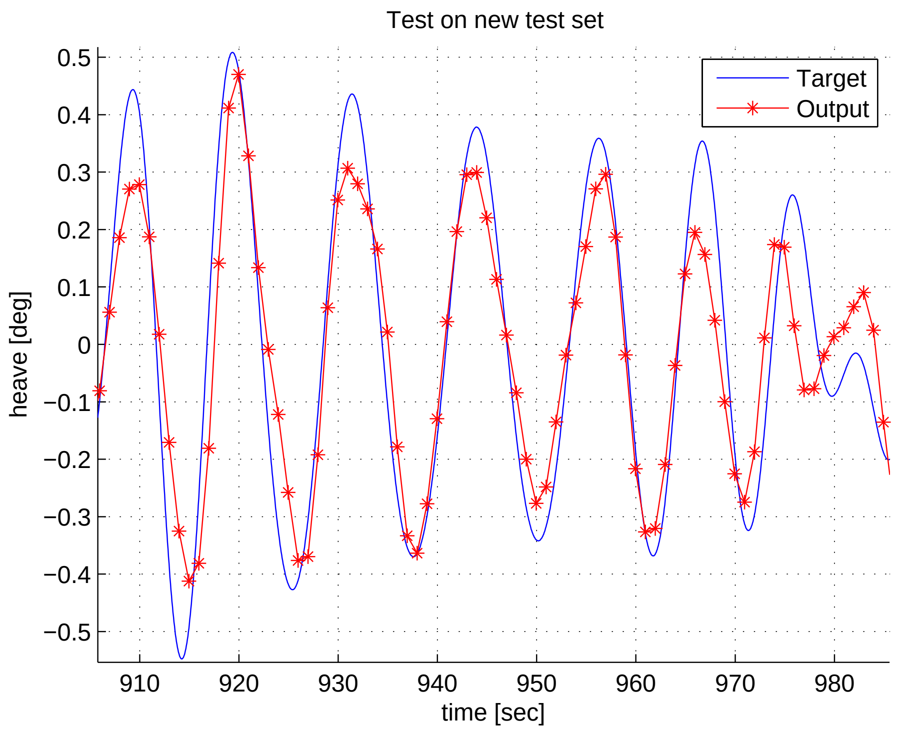


Figure 5.13: Heave displacement prediction 60 seconds ahead

Table 5.5: Final test with different steps for heave displacement prediction

Prediction step [s]	Training time [s]	Within 0.25 m [%]	Within 0.25 m at peaks [%]	MAE [m]	MSE [m]
5	0.281	100.0	100.0	0.045	0.003
10	0.257	98.6	98.4	0.089	0.012
20	1.170	96.7	93.1	0.098	0.015
30	1.388	93.3	87.7	0.114	0.020
40	1.318	93.3	90.5	0.110	0.019
50	1.357	93.0	88.2	0.117	0.021
60	1.373	96.6	93.7	0.097	0.014

5.1.9 Heave Displacement Prediction - Data Fusion

To test if a better forecasting result is possible, data fusion is tested. In addition to the described heave and yaw motion of the ship, the waves will also affect the roll and pitch movements. Because the yaw motion to a large degree can be influenced by control of the ship, it is not used for fusion. Similar to heave motion, the roll and pitch motions are affected almost exclusively by the waves. Hence, they are suitable for data fusion with heave.

As input to the ANN there will be an equal amount of samples from heave, roll and pitch, while the output target is still only the heave displacement. Like the previous test, 300 input vectors in the training set and 500 in the test set were used. Only linear activation functions are used and there is only one hidden neuron. Table 5.6 displays the minimum input parameter combinations necessary to obtain 90 %, 95 % and 100 % of errors within 0.25 meters for each prediction step. Figure B.9 in the appendix presents the plot used to find the optimal combinations for the 60 seconds prediction step in the data fusion case.

Table 5.6: Prediction steps, data fusion, with their necessary combinations of input time range/sampling frequency

Prediction step [s]	Min. 90 % within 0.25 m [s/Hz]	Min. 95 % within 0.25 m [s/Hz]	100 % within 0.25 m [s/Hz]
5	6/0.5	6/0.5	6/0.5
10	6/0.5	10/0.5	10/1.0
20	20/0.5	30/0.5	30/1.0
30	20/0.5	30/0.5	30/1.5
40	20/1.0	40/1.5	–
50	20/1.5	30/4.0	–
60	30/1.5	30/4.0	–

The results from tests with the input parameter combination that gave minimum 95 % of the errors within 0.25 meters are presented in Table 5.7.

Table 5.7: Final test, data fusion, with different steps for heave displacement prediction

Prediction step [s]	Training time [s]	Within 0.25 m [%]	Within 0.25 m at peaks [%]	MAE [m]	MSE [m]
5	0.140	100.0	100.0	0.043	0.003
10	0.172	99.0	98.4	0.083	0.010
20	0.671	96.4	96.8	0.102	0.016
30	0.858	96.0	95.7	0.097	0.015
40	1.092	97.4	98.5	0.106	0.017
50	8.159	93.6	96.0	0.118	0.019
60	4.602	94.2	100.0	0.106	0.018

When comparing the performance achieved from this data fusion test (Table 5.6) with the standard test using only the heave time series (Table 5.4), it is evident that input samples from a smaller time range is needed when using data fusion. To get a minimum of 95 % of the errors within 0.25 meters, samples from no more than 40 seconds backwards in time is needed, compared to up to 120 seconds required for the standard test. A major advantage of looking at a smaller time span is that, because real life weather is highly dynamic and always changing, possibly obsolete old data is excluded and only the most recent data is used for forecasting. A disadvantage with data fusion is that for this data set it is hard to achieve 100 % of the errors within 0.25 m for long prediction steps.

Looking at Table 5.5 and Table 5.7, a comparison of the training times for the standard test and data fusion test, respectively, can be made. Data fusion requires samples from three sensors, but from a much smaller time range. The tests show that the training time for a prediction step of 5 to 40 seconds is less for data fusion than for the pure heave time series. On the contrary, for a prediction step of greater than 50 seconds, data fusion leads to a longer training time.

5.1.10 Heave Displacement Prediction - Added Noise

In the above tests it is assumed that all the values are measured perfectly. When using the system with data collected in real time from sensors on board a ship, there will always be some degree of measurement noise. To see how well the ANN predictors constructed above will cope with this, noise is added to the data used. Only the target values of the test data are retained original, as the predicted value should be compared with the true value, independent of measurement noise.

The noise added to the data is random values from a normal distribution with zero mean and a standard deviation depending on the standard deviation of the different data sets. The formula for the normal distribution is:

$$\overline{x}_i = \frac{\sum_{n=1}^{N} x_i(n)}{N}, \quad \sigma = \sqrt{\frac{\sum_{n=1}^{N}(x_i(n) - \overline{x}_i)^2}{N}} \tag{5.2}$$

$$y = f(x|s\sigma, \mu = 0) = \frac{1}{s\sigma\sqrt{2\pi}} e^{\frac{-x^2}{2(s\sigma)^2}} \tag{5.3}$$

where x_i is the original data, N is the number of samples, s is the scaling factor for the standard deviation σ and y is the normal probability density function, i.e. the function describing the relative likelihood for a random variable x to take a specific value.

The scale factor used for the normal distributed noise is $s = 0.1$, meaning 10 % of the data's standard deviation is added as noise. This results in $s\sigma_{heave} = 0.026$ m, $s\sigma_{roll} = 0.028°$ and $s\sigma_{pitch} = 0.056°$. An example of how this noise influences the heave data can be seen in Figure 5.14.

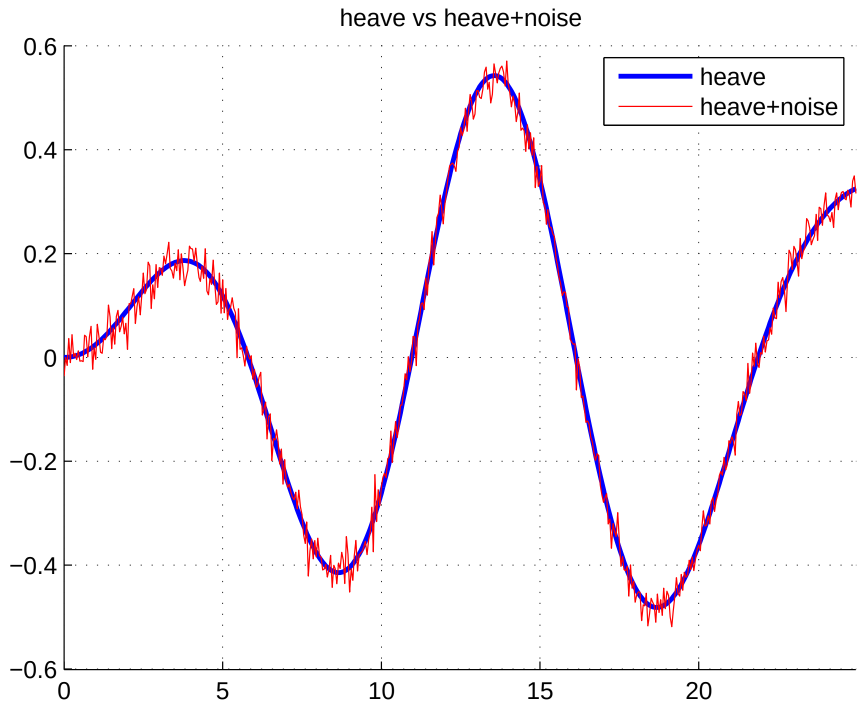


Figure 5.14: Heave data plus added noise

Heave displacement prediction for a prediction step of 30 seconds is tested for both the standard and the data fusion approach. The ANN architecture discussed in their respective subsections that gave 95% of the errors within 0.25 meters is used. Table 5.8 presents the median result after 20 runs.

Table 5.8: Heave displacement prediction, 30 seconds prediction step. With and without added noise

Prediction approach	Training time [s]	Within 0.25 m [%]	Within 0.25 m at peaks [%]	MAE [m]	MSE [m]
Without noise:					
Standard	1.927	95.2	91.8	0.111	0.018
Data fusion	0.827	95.2	94.1	0.104	0.018
With added noise:					
Standard	0.655	88.0	73.5	0.130	0.025
Data fusion	0.499	91.4	87.2	0.121	0.023

When noise is added to the training set and the test input vectors, data fusion gives the best results for all the performance measurements. Data fusion also experiences the least performance degradation with noise added. This demonstrates that the data fusion approach to ANN heave displacement prediction is more robust than the standard approach.

One reason may be that for all time steps used as inputs, data fusion takes three samples (heave, roll and pitch) with independent measurement noise. Because the noise from the three samples are uncorrelated, they will generally push the ANN prediction in different directions (e.g. higher or lower heave displacement). Combined, the three samples will to some degree cancel each others effect out, hence making more accurate predictions than with only one noisy sample.

It is interesting that the training time is reduced drastically in both cases even though the only change is that noise is added. In the general case, noisy data may help the ANN to both generalize better (avoid over-fitting) and easier escape from local minima during training. This is because noisy data leads to a noisy estimate of the true gradient in the training algorithm, helping the network to jump out of the local minimum and move into a better minimum. This helps reassure that the training time will not be a problem for a real life on-line system.

5.1.11 Discussion - Heave Displacement Prediction

ANN has proven a successful forecasting tool for heave displacement of the simulated ship. Prediction steps of up to 60 seconds have been tested and found feasible within the given performance criteria. As with ship heading prediction, no other prior information is required than the automatically logged ship motion from the last five minutes.

Different prediction steps require different ANN architectures for optimal performance. In the general case of heave displacement prediction, only one hidden layer and one hidden neuron is necessary, only linear activation functions should be used and more samples from a larger time range generally leads to better performance.

Two reasons for not to always use the maximum sampling frequency and largest time range available are training time and over-fitting. The execution time for a single prediction once the ANN is fully trained will also be negatively affected by the increasing number of inputs, as more inputs leads to more computations being necessary when making a new prediction. Despite this, when considering the relatively low number of inputs discussed in these tests and the time requirements for the UAV landing system, the execution time for a prediction will always be small enough to be neglected (in the order of 10 milliseconds).

When comparing ANN settings that gave approximately the same prediction errors, the data fusion approach was found to generally need less training time, require a shorter time range of previous samples and be more robust to measurement noise compared to the standard heave time series approach. The only exception was at the longest prediction steps (50 and 60 seconds) where the same maximum performance obtained with standard prediction was not achieved using data fusion.

5.2 UAV Landing - Software-In-the-Loop (SIL)

To test the capability of the landing system's guidance laws and controllers, SIL simulations are performed according to what was described in Section 4.5. During all simulations the initial landing path is similar. In Appendix C.1 follows both a table presenting all the values used to define the waypoints and a table presenting their calculated latitudes, longitudes and heights. The necessary landing net information gets sent from the developed Neptus automatic landing plug-in, and in DUNE the complete landing path gets created according to Subsection 4.5.3.

Before landing is initiated, the UAV is manually set to take off and loiter (flying in a circle motion). To ensure easily comparable simulations where the UAV initiates landing at approximately the same place every time, the same loiter path settings are always used, summarized in Table C.3 in the appendix. Figure 5.15 illustrates a simplified version of the static landing path with the loiter circle drawn with dotted line.

Review of the performance is done using Neptus MRA and Matlab. MRA allows data collected during execution of the landing to be exported to Matlab, where code has been developed to calculate and plot information like comparison of desired and estimated altitude, cross-track error and the trajectory projected onto the East-North plane.

Below follows tests of UAV landing with both a static and dynamic target/net. Simulated weather conditions are also varied to examine the robustness of the guidance laws and controllers, and the ability to perform an evasive maneuver is evaluated. All tests are performed 5 times each to give enough samples to say something about the general performance.

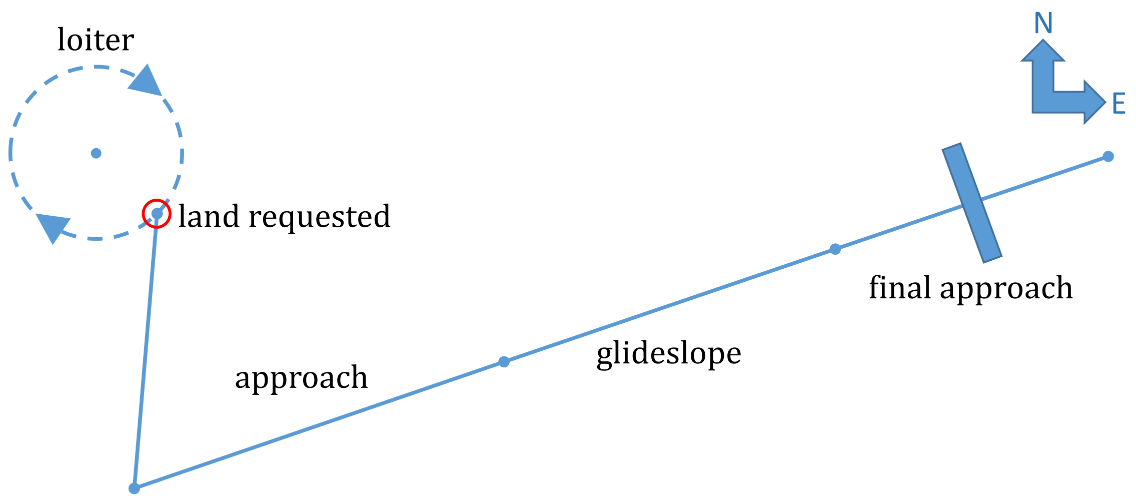


Figure 5.15: Example of lateral landing path with the added loiter circle

5.2.1 Landing Criteria

To evaluate if a simulated net landing is successful, a set of landing criteria is used. The performance is measured at the moment of impact with the position of the simulated net.

Criteria for successful net landing

- Cross-track error less than ± 1 m
- Altitude error less than ± 1 m
- Course over ground vs. heading less than $\pm 45°$

All criteria and supporting explanations can be found in Subsection 4.3.4 "Evasive Maneuver".

5.2.2 SIL Simulation - Static Path, No Wind

To tune and investigate the feasibility of the guidance systems and controllers, the initial test is with a static net and no wind disturbance.

Plots of the desired path and flown trajectory from a single flight can be seen in Figure 5.16 and 5.17 in longitudinal and lateral direction respectively. When landing is initiated, the UAV is already in the air. The approach phase contains some vertical oscillations and divergence from the desired path because of insufficient tuning. This thesis mostly concerns performance when descending in the FA phase and not during horizontal flight.

Figure 5.18 and 5.20 illustrate the altitude- and cross-track error, respectively, of the 10 last seconds for 5 different flights. During the entire 10 last seconds before net impact, the altitude error is within ± 1 m and the cross-track error is within ± 0.5 m for all five flights. Error plots for the entire landing sequence can be found in Figure

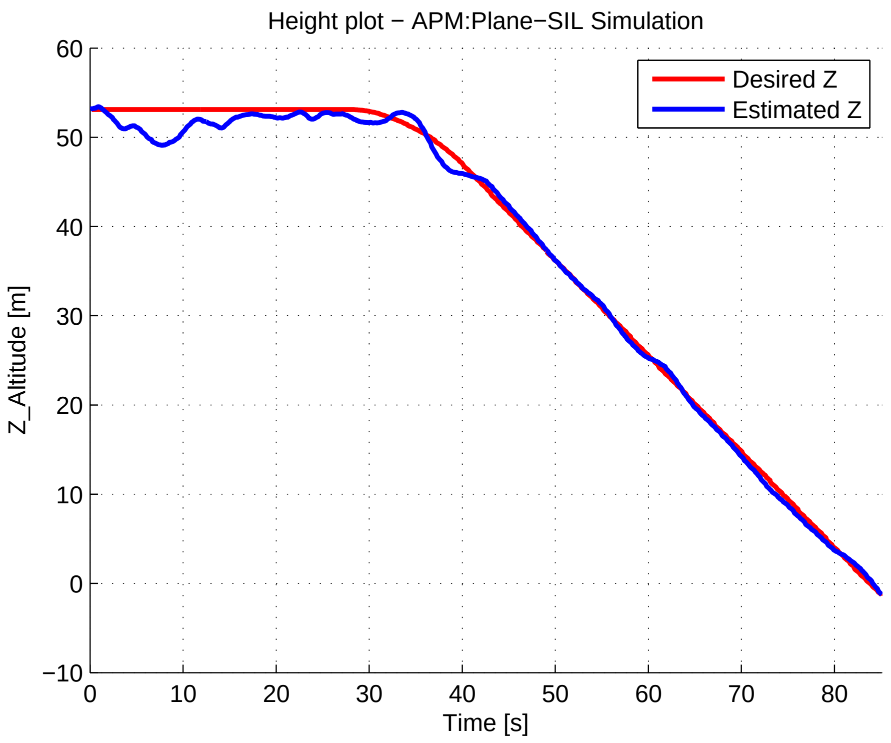


Figure 5.16: Altitude plot of one flight, static path - no wind

5.20 and Appendix C.3. Due to no wind, the COG and heading are approximately equal. This satisfies all the successful net landing criteria.

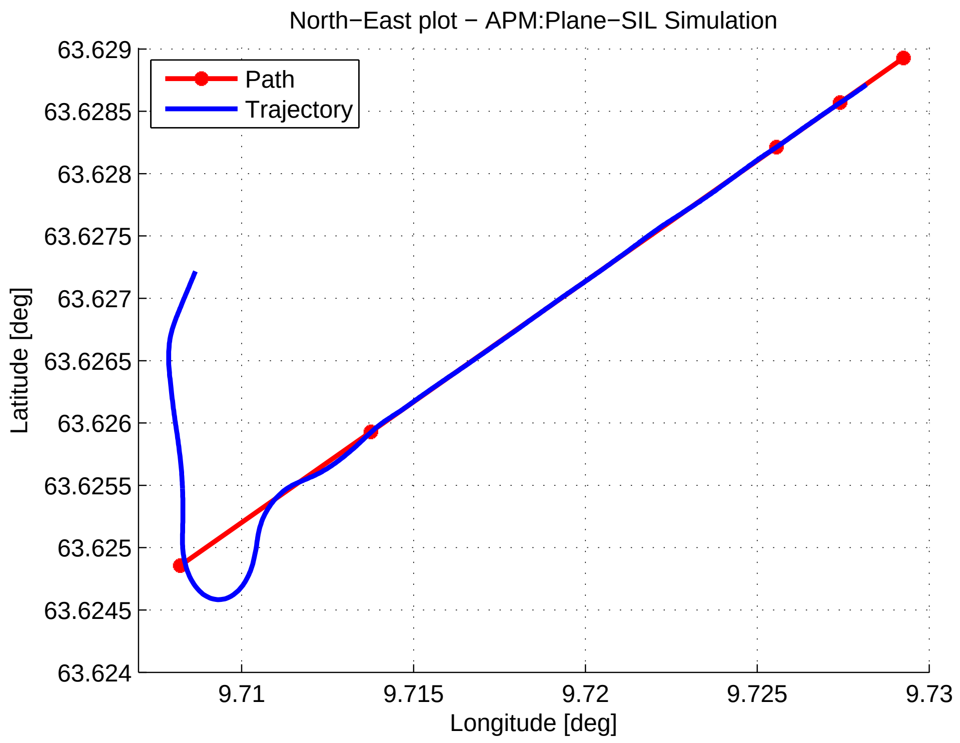


Figure 5.17: Lateral trajectory plot of one flight, static path - no wind

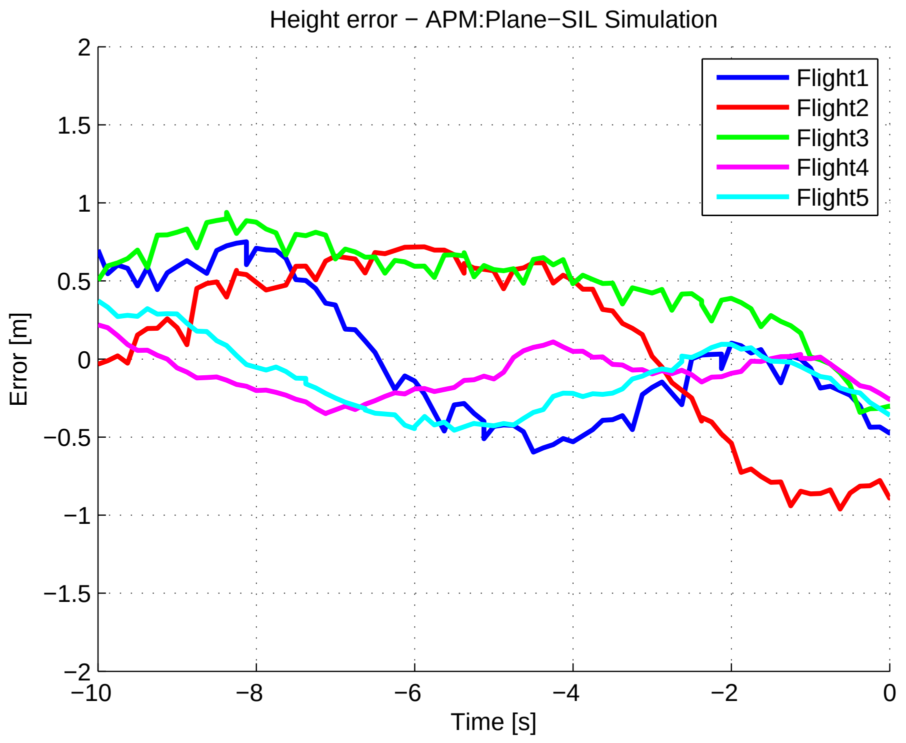


Figure 5.18: Altitude error of 5 flights, 10 last seconds, static path - no wind

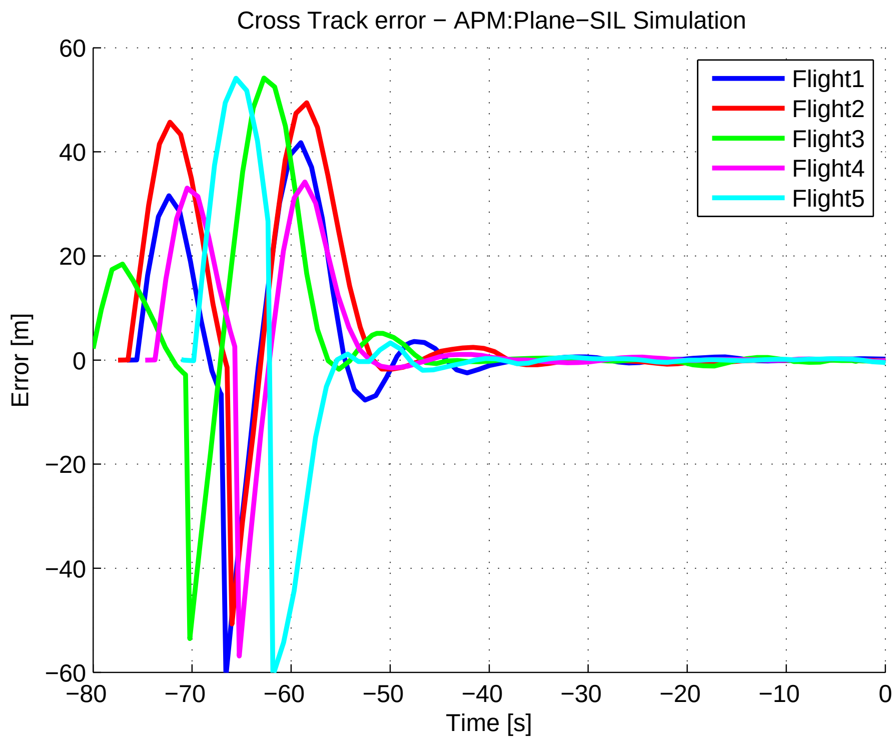


Figure 5.19: Cross-track error of 5 flights, static path - no wind

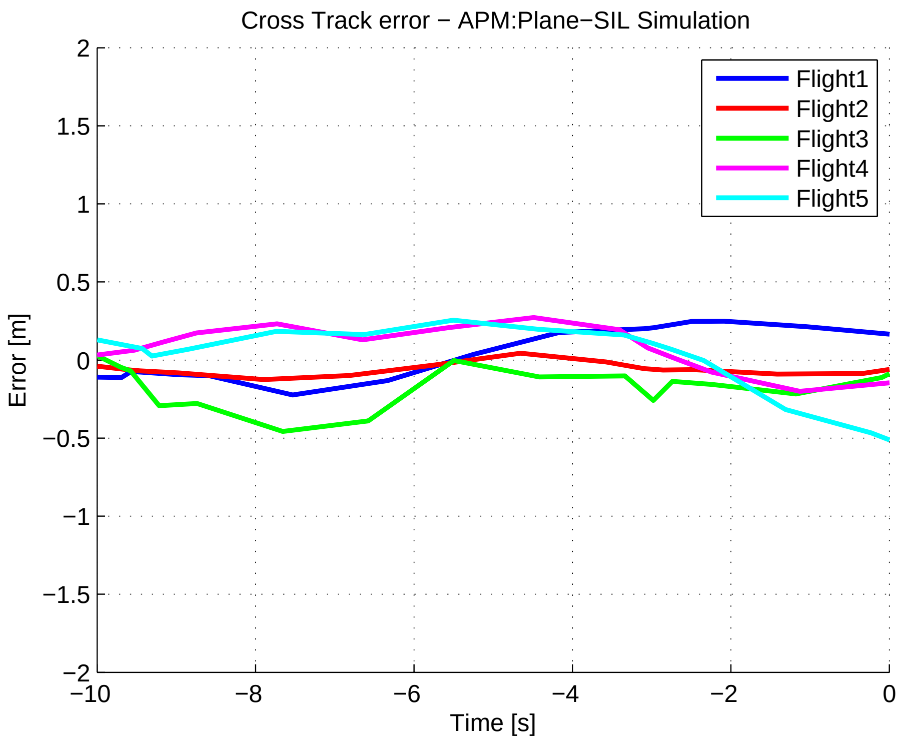


Figure 5.20: Cross-track error of 5 flights, 10 last seconds, static path - no wind

5.2.3 SIL Simulation - Static Path, Wind and Turbulence

For a more realistic test, wind and turbulence was added to the simulation. Wind speed was set to 6 knots (3 m/s) going north with a turbulence of 0.5 knots. This gives a tail and crosswind during landing that is reasonably strong considering the light weight of the X8 UAV. Appendix C.4 presents plots of the resulting varying wind speed and wind direction.

Plots of the height error and cross-track error from the landing of five flights can be seen in Appendix C.4, while Figure 5.21 and Figure 5.22 below only illustrates the 10 last seconds before net impact. Figure 5.23 illustrates the difference between COG and heading for the UAV during the entire landing sequence.

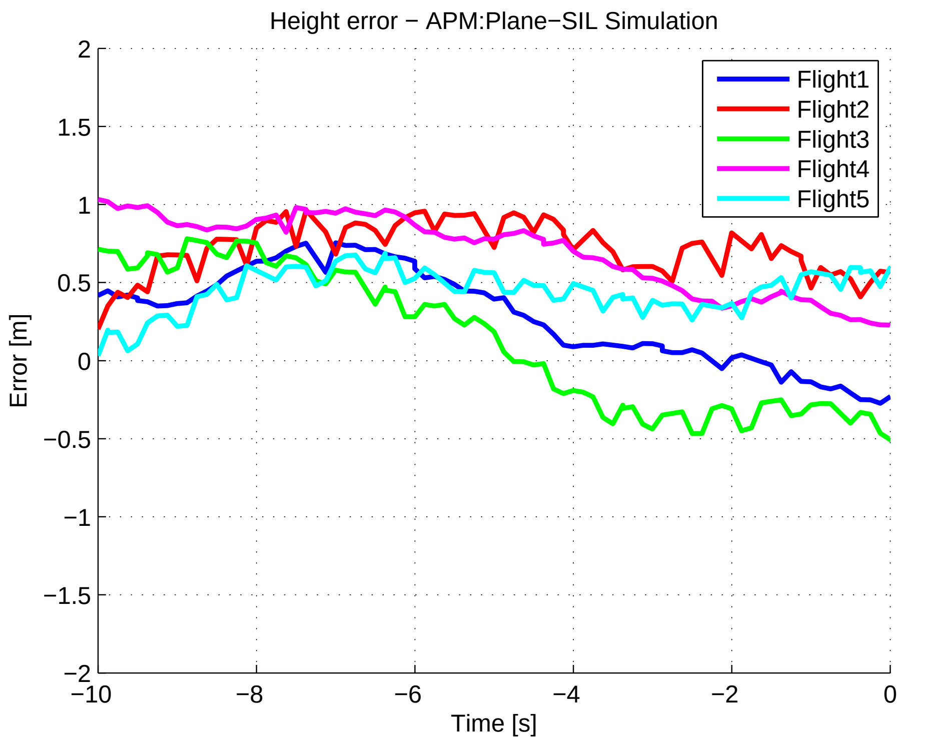


Figure 5.21: Altitude error of 5 flights, 10 last seconds, static path - wind and turbulence

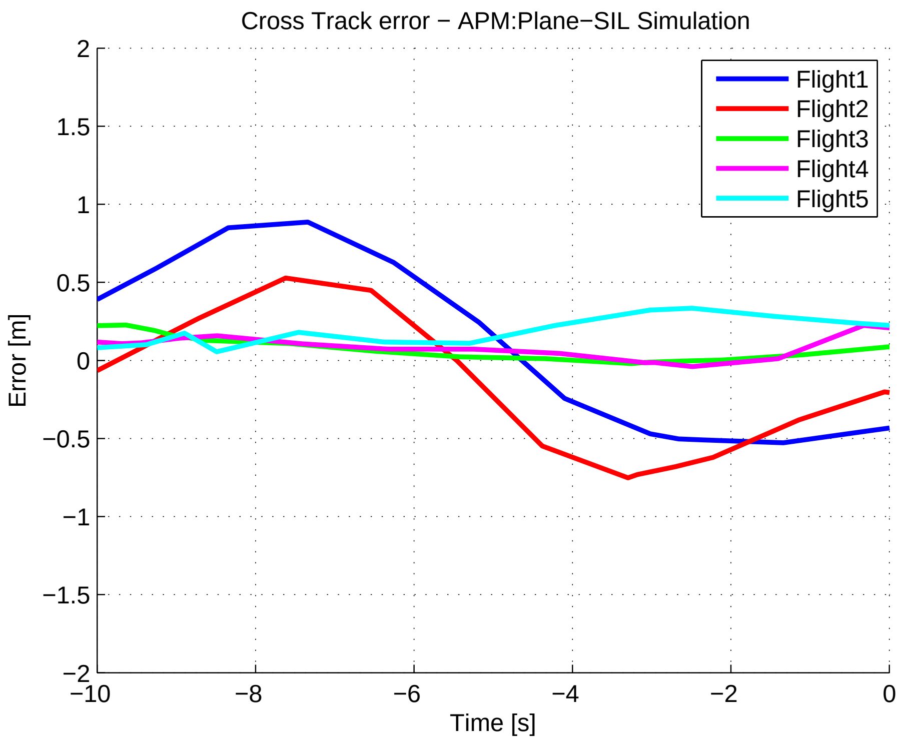


Figure 5.22: Cross-track error of 5 flights, 10 last seconds, static path - wind and turbulence

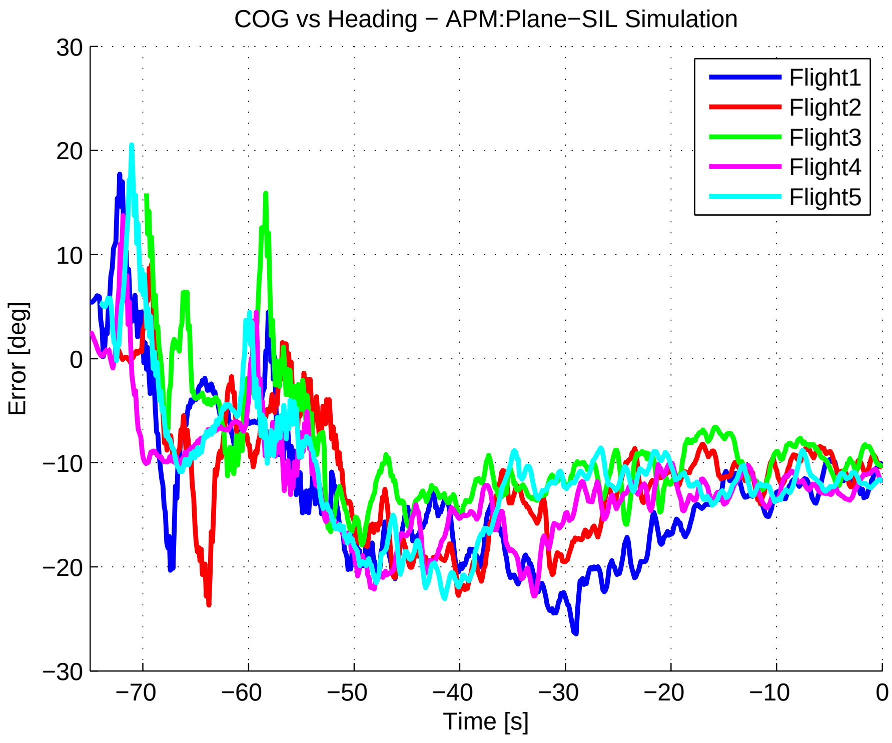


Figure 5.23: Difference between COG and heading, static path - wind and turbulence

During the entire 10 last seconds before net impact, both the altitude error and cross-track error are within ±1 m for all five flights. As wind is introduced in this test, there is a difference between COG and heading. It is always less than ±25° and around −10° on impact. All of the successful net landing criteria are satisfied.

5.2.4 Discussion - Static Path

Looking at the tests performed in Subsection 5.2.2 and Subsection 5.2.3, the control system manages to withstand wind disturbance and hit the target within the landing criteria. The lateral controller shows great performance, while the height controller is somewhat suboptimal.

The lateral controller manages to converge the cross-track error towards zero both with and without wind. It has some oscillations to begin with, but they are quickly removed or reduced to an acceptable level. Oscillations are a trade-off between fast response to large cross-track errors and a smooth trajectory. Because the guidance system is based on *integral* LOS (ILOS), it is able to compensate for the crosswind without introducing an offset. The cascade coupling with the output of the ILOS connected to the input of the BTT-PID provides a robust and stable response.

The longitudinal controller manages to follow the vertical Dubins path. Even in the presence of tail- and cross-wind, the error is kept low. During the approach phase where the desired altitude is constant and the desired climb rate is zero, there were some oscillations and deviations. A reason for this is that most effort has been put into tuning the controller for the descend and FA phase. During descent the proactive PID controller works better, albeit it could be fine-tuned to further reduce the oscillations in FA. In the absence of a mathematical model of the system, limiting the choices of guidance systems, the proactive PID controller proves to be a good choice.

5.2.5 SIL Simulation - Evasive Maneuver

If an evasive maneuver is needed during FA, it is important to know that the ship in fact will be avoided. The criterion for this is that the UAV does not pass the simulated target by less then 2.5 m vertically upwards and 3.5 m horizontally to each side. This ensures a safety zone of 1.0 m around the net.

To test the evasive capability, an evasive maneuver is automatically triggered regardless of errors when the UAV is 4 seconds away from hitting the net, as is defined in Subsection 4.3.4 to be the minimum ETA before an evasive request is ignored. Figure 5.24 illustrates the lateral trajectory taken by the UAV.

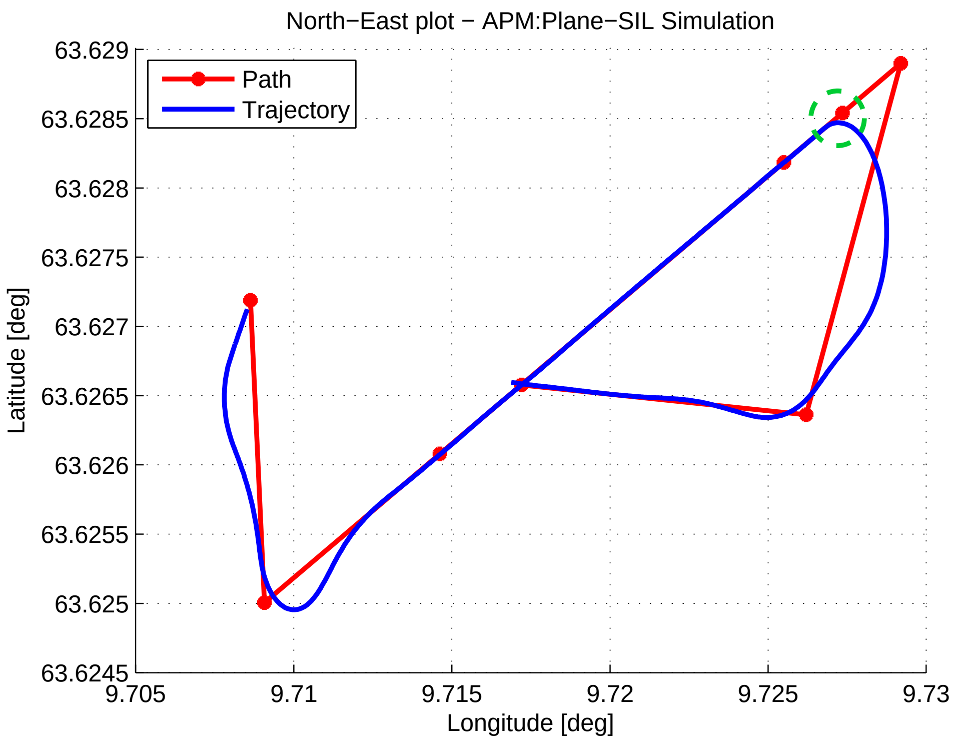


Figure 5.24: Lateral trajectory plot of one flight with net marked as green circle, evasive maneuver

Figure 5.25 below illustrates a plot of the lateral trajectory from five flights when an evasive maneuver is executed. It shows the relative position in meters, where the net is positioned at $(0, 0)$. A circle around the net with a radius of 3.5 m clarifies that all flights avoids getting too close to the net.

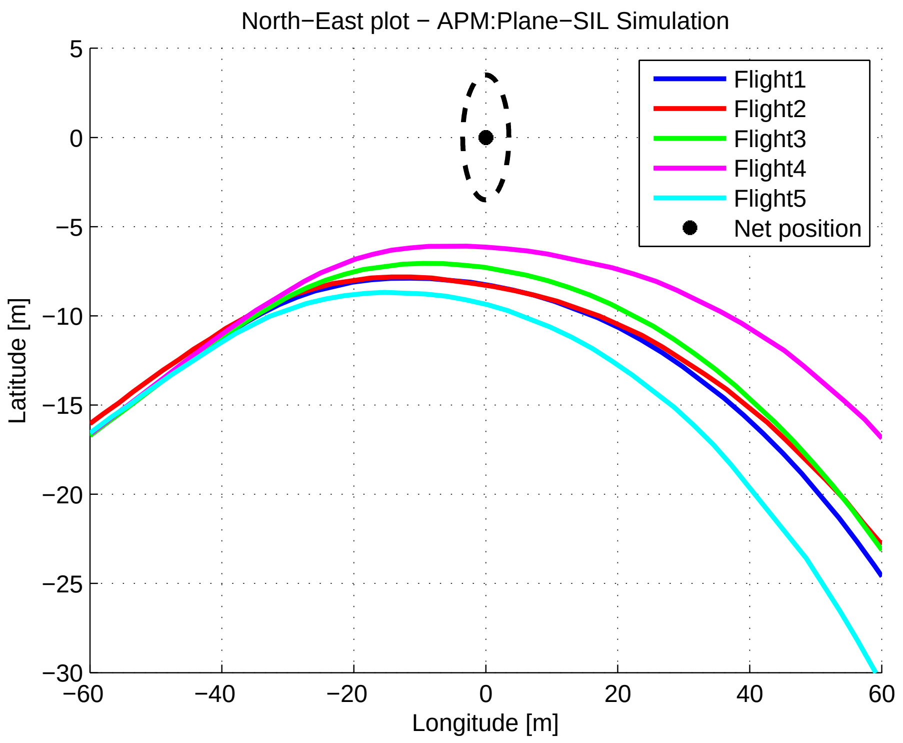


Figure 5.25: Relative lateral trajectory of 5 flights right after an evasive maneuver is requested. 3.5 m safety radius around the net is marked as a black circle

Figure 5.26 and Figure 5.27 illustrate the vertical trajectory of the UAV when the evasive maneuver is requested at time 0 s and the altitude is shifted to be 0 m at this point. They illustrate the trajectories for the whole simulation and a zoomed in view respectively.

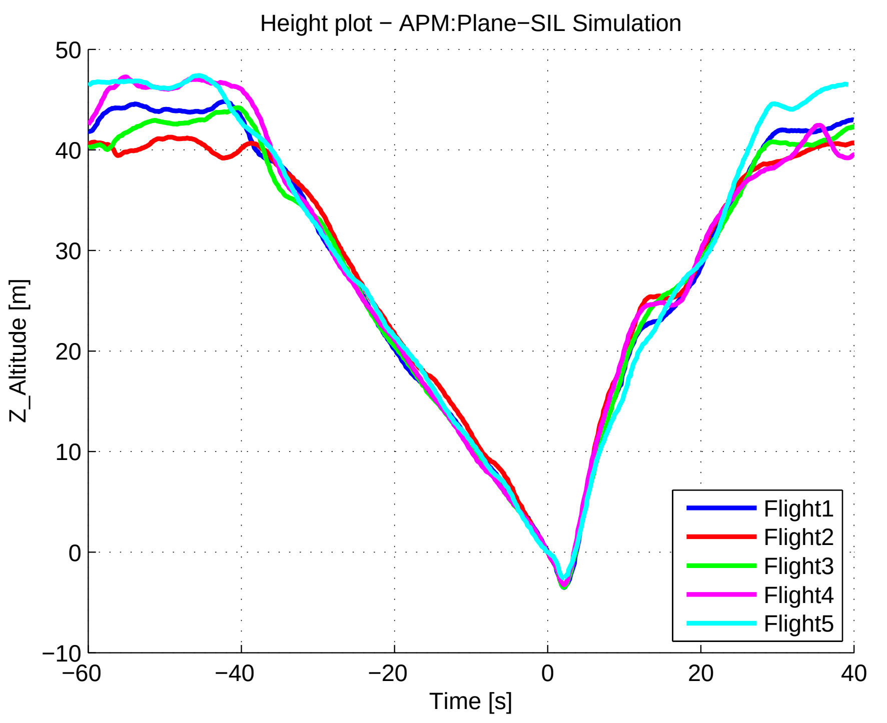


Figure 5.26: Relative altitude of 5 flights, where request for evasive maneuver is at time 0 s and altitude 0 m

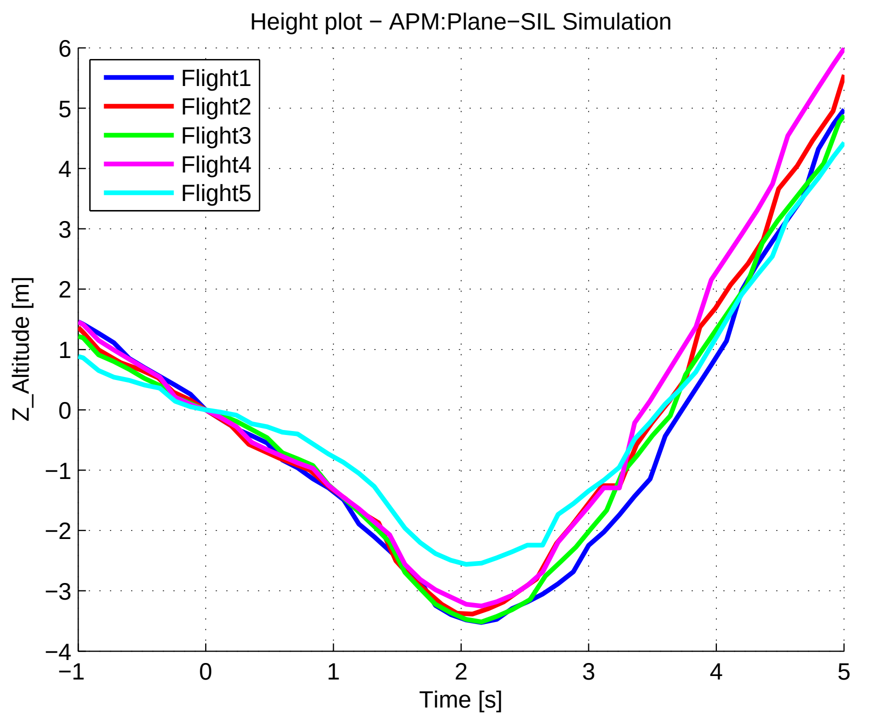


Figure 5.27: Relative altitude of 5 flights, where request for evasive maneuver is at time 0 s and altitude 0 m. Zoomed in view

There is for all flights a consistent delay of about 2 seconds before the UAV changes climb rate. A reason for this is that the height controller designed is high-level, so the change in climb rate has to propagate through the low-level controllers in APM:Plane instead of the aileron deflection being controlled directly. The DUNE environment also adds some computational delay. Despite this, when the UAV starts to change climb rate, it reacts quickly. After about 3.5 s it is at the altitude it was when the evasive maneuver was requested.

From the tests it is evident that when an evasive maneuver is requested 4 seconds before net impact, the UAV has time to avoid the net horizontally with about twice the required 3.5 m and vertically the descent is turned into ascent after 2 seconds. This gives redundancy, meaning that if the height controller fails to react the UAV will still avert a crash laterally, and if the lateral controller is too slow the UAV will still fly about 5 meters above the net.

As the evasive maneuver in this test demonstrates a relatively fast response, the minimum ETA required for an evasive maneuver could be lowered from 4 to 3 seconds without getting too close to the net. However, further tests should be carried out with different weather conditions before this change in required minimum ETA can be fully justified.

5.2.6 SIL Simulation - Dynamic Path, No Wind

If the ship carrying the net changes pose after landing is already initiated, the landing path gets updated without needing to reset the entire landing sequence. By utilizing ship motion prediction, the path can be updated far ahead in advance. For the following two tests, the pose of the net is changed approximately 30 seconds away from the net, in accordance to the recommended minimum prediction step for accurate ship motion prediction presented in "Forecasting Criteria" in Subsection 5.1.1.

The change in net pose was manually sent from *Neptus IMC Message Sender*, and the values is listed in Table 5.9 below. It involves a change in horizontal position, heave displacement and heading/yaw angle. Only the two latter is forecasted in the ship motion predictions presented earlier in this thesis, but the change in horizontal position is added as it may be applicable later. By updating all the four pose parameters of the landing net at the same time, the guidance and control system's response to a dynamic path is thoroughly tested.

Table 5.9: Net pose change, sent as *IMC::DeviceState*

Variable	Value
x (East)	5.0 m
y (North)	−5.0 m
z (Up)	2.5 m
phi (roll)	0.0°
theta (pitch)	0.0°
psi (yaw)	5.0°

Figure 5.28 and Figure 5.29 illustrate the height plot and lateral trajectory of a single flight, respectively. The path update is seen as an abrupt change in the red line representing the desired path.

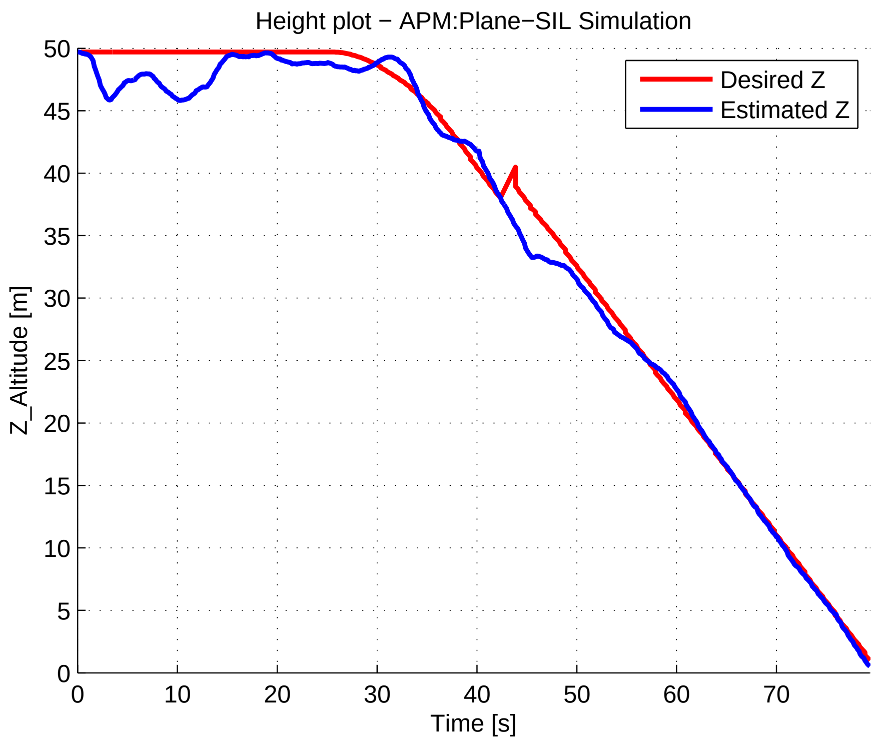


Figure 5.28: Altitude plot of one flight, dynamic path - no wind

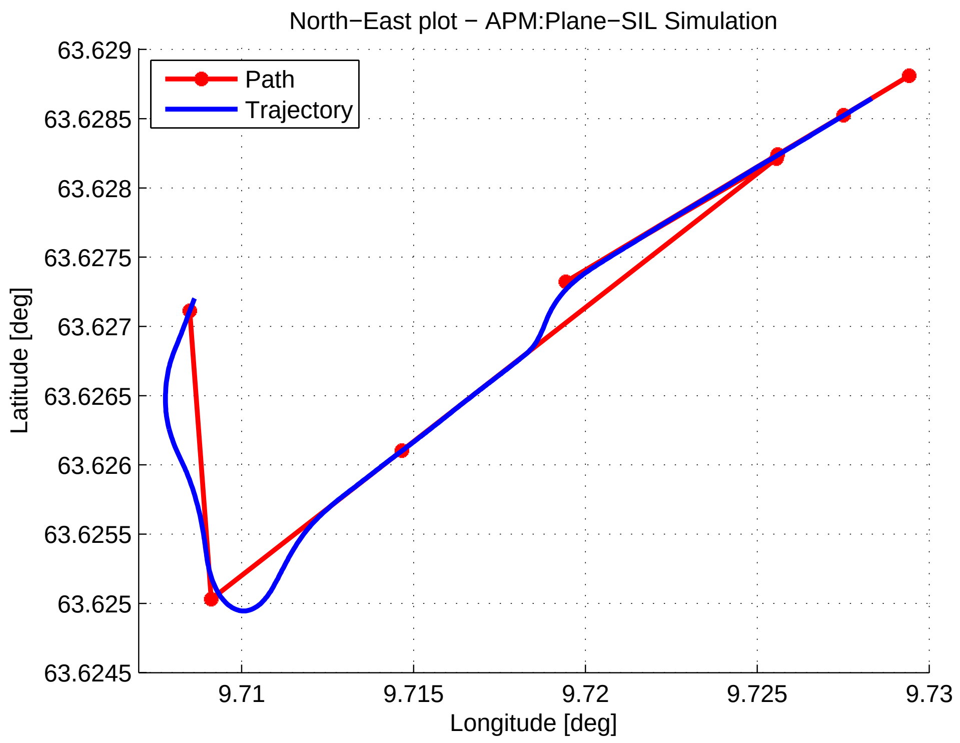


Figure 5.29: Lateral trajectory plot of one flight, dynamic path - no wind

Plots of the height error and cross-track error from the moment of the path update until simulated net impact can be seen in Figure 5.30 and Figure 5.32 respectively. They demonstrate how a large error is introduced at the moment of path update, but relatively quickly reduced. The lateral controller demonstrates better performance than the hight controller, albeit they both perform satisfactory.

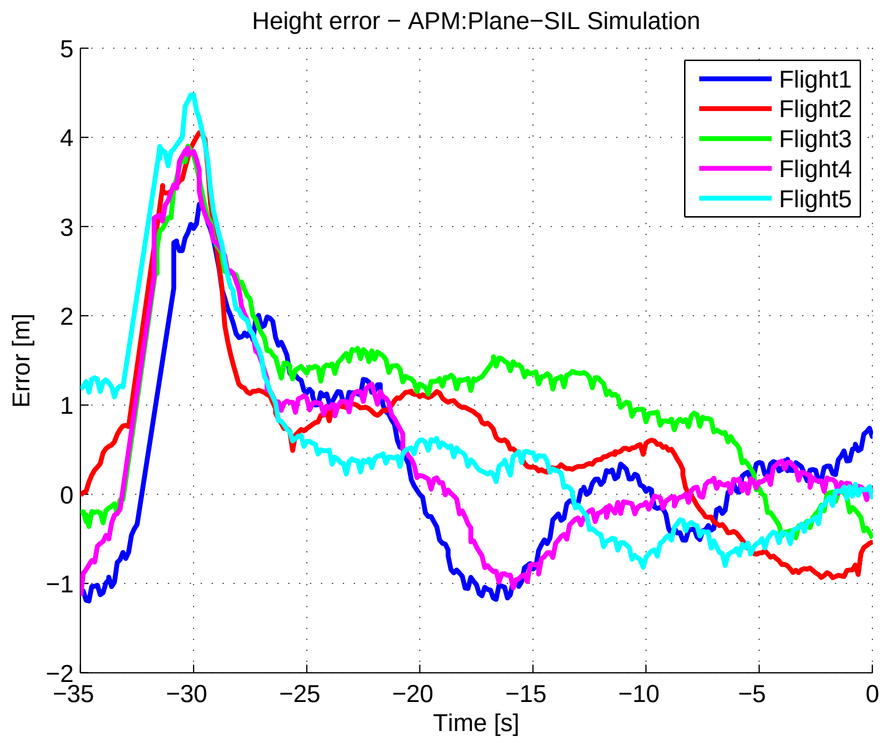


Figure 5.30: Altitude error of 5 flights - starting from path update, dynamic path - no wind

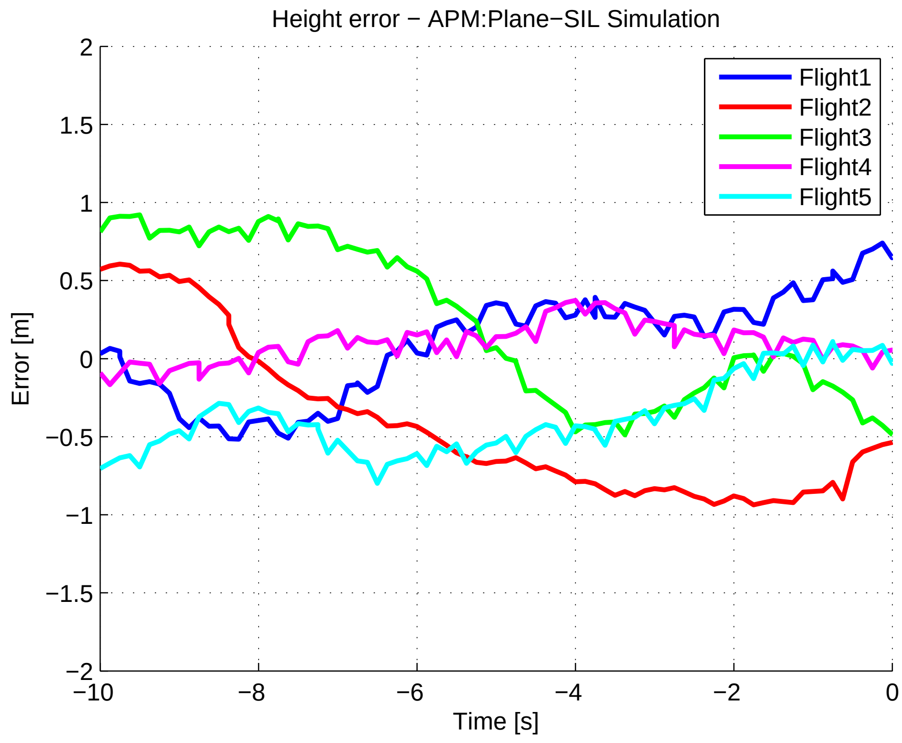


Figure 5.31: Altitude error of 5 flights, 10 last seconds, dynamic path - no wind

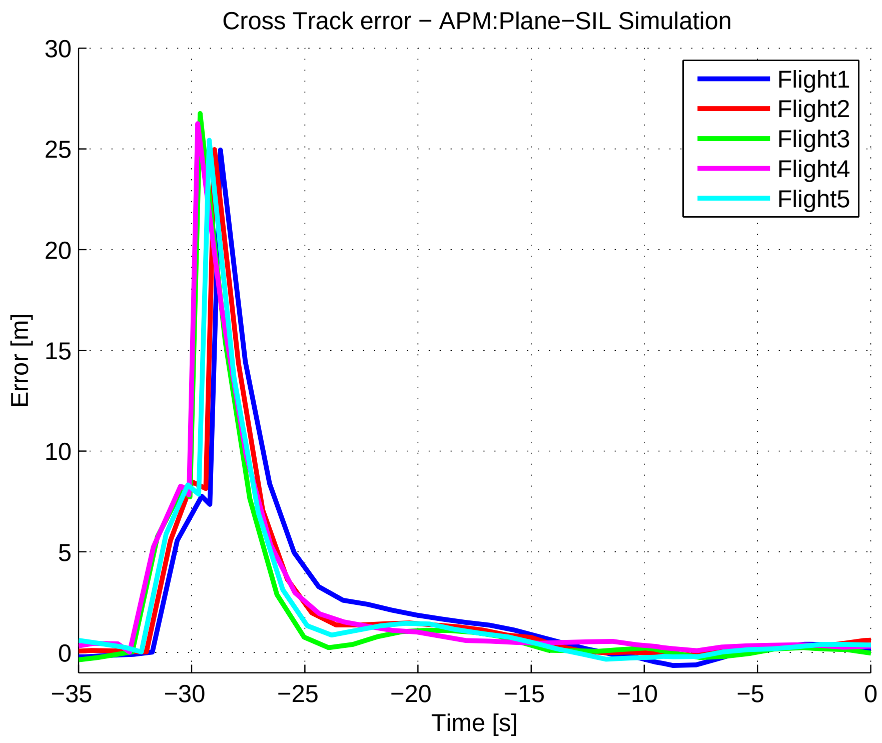


Figure 5.32: Cross-track error of 5 flights, from path update, dynamic path - no wind

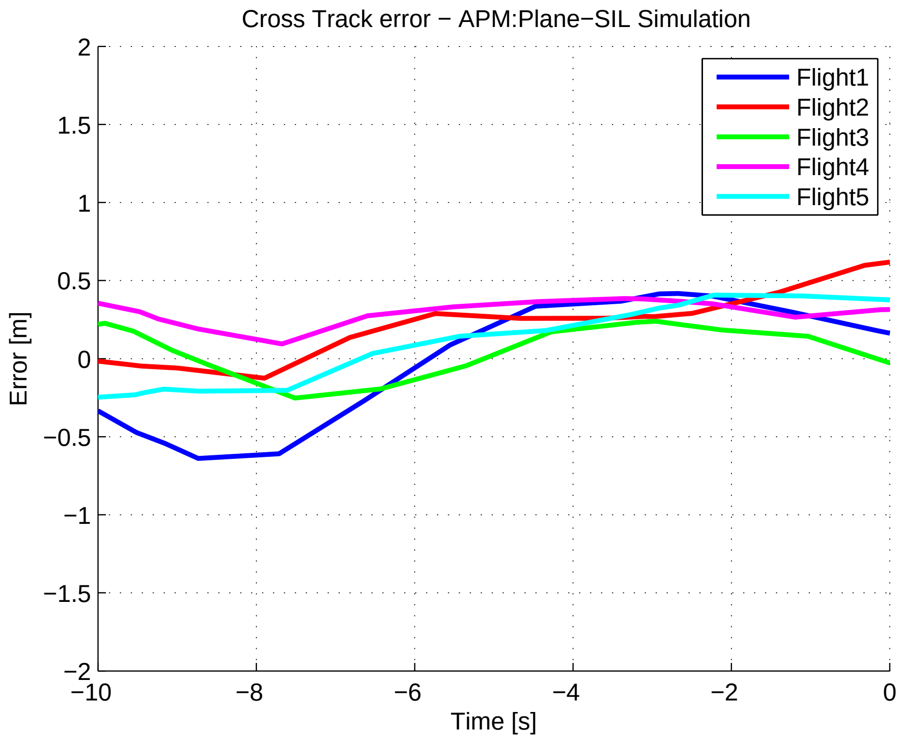


Figure 5.33: Cross-track error of 5 flights, 10 last seconds, dynamic path - no wind

During the entire 10 last seconds before net impact, both the altitude error and cross-track error are within ±1 m for all five flights, as seen in Figure 5.31 and Figure 5.33 respectively. Due to no wind, the COG and heading are approximately equal. This satisfies all the successful net landing criteria.

5.2.7 SIL Simulation - Dynamic Path, Wind and Turbulence

It is more realistic to assume that a dynamic path is needed when there is rough weather. The same wind conditions are used as in the static path test, meaning wind speed of 6 knots going north with a turbulence of 0.5 knots. The change in net pose is the same as earlier (Table 5.9).

Complete plots of the height error and cross-track error from the landing of five flights can be seen in Appendix C.5, while Figure 5.34 and Figure 5.35 below only illustrates the 10 last seconds before net impact. Figure 5.36 illustrates the difference between COG and heading for the UAV during the entire landing sequence.

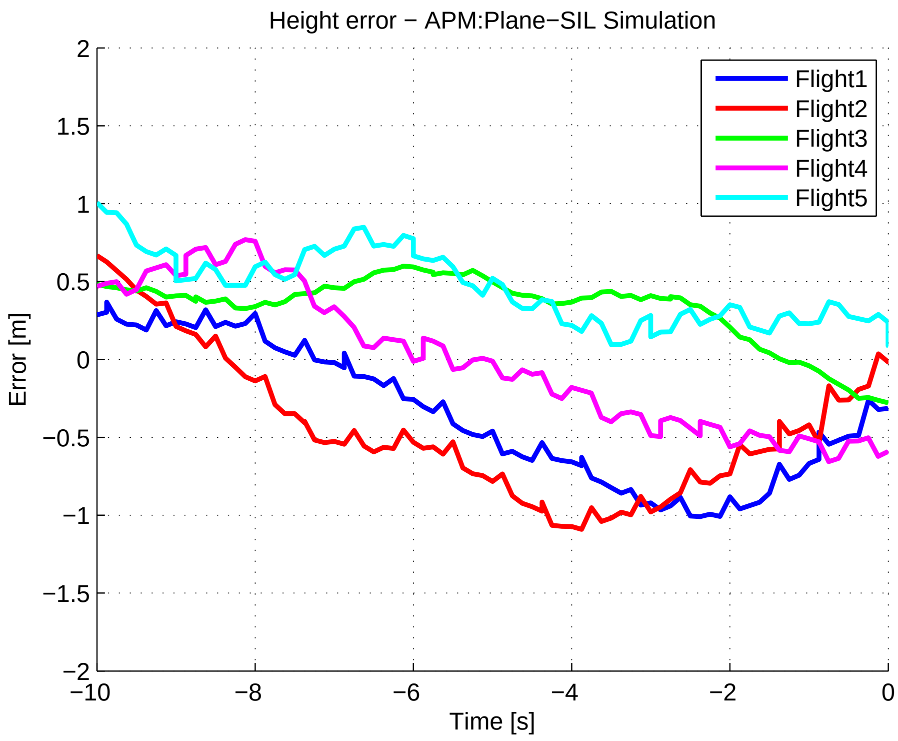


Figure 5.34: Altitude error of 5 flights, 10 last seconds, dynamic path - wind and turbulence

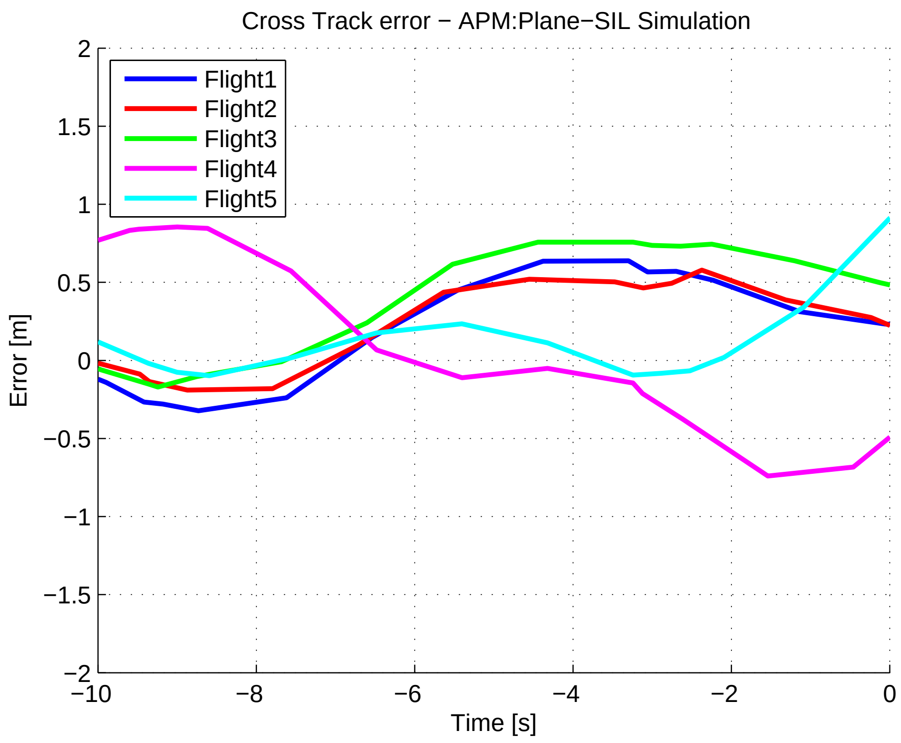


Figure 5.35: Cross-track error of 5 flights, 10 last seconds, dynamic path - wind and turbulence

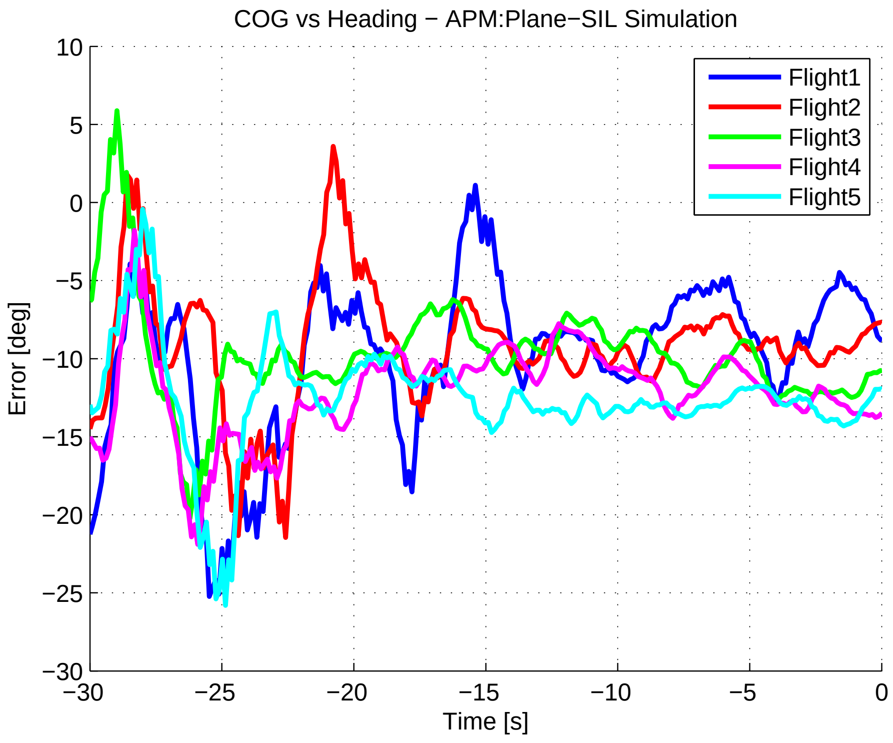


Figure 5.36: Difference between COG and heading, static path - wind and turbulence

The performance is slightly worse than with no wind, but both the altitude error and cross-track error are still within ± 1 m for all five flights, except for a 0.5 s period for Flight 2, 4 s before net impact. As wind is introduces, there is a difference between COG and heading, illustrated in Figure 5.36. It is always less than $\pm 25°$ and around $-10°$ on impact. All of the successful net landing criteria are satisfied.

5.2.8 Discussion - Dynamic Path

Subsection 5.2.6 and Subsection 5.2.7 present results from tests where the net pose is changed and the landing path is updated dynamically during landing. They demonstrate how the control system successfully manages to converge the UAV onto the new path in a smooth manner, without a need to reset the landing process or lead the UAV on a detour.

Especially the lateral controller demonstrates a fast and smooth convergence onto the new path leading into the translated and rotated landing target. The height controller is also fast, albeit a bit less smooth. This suggests that the use of a dynamic path, updated based on ship motion prediction, is feasible. Even severe changes to the landing target as close as 30 seconds before estimated impact can be handled by the landing system.

Chapter 6

Closing Discussion and Conclusion

This chapter concludes the thesis in Section 6.1, and provides recommendations for further work in Section 6.2.

6.1 Closing Discussion and Conclusion

The goal of this project was to design and verify the performance of an automatic landing system for a fixed-wing UAV to be used with a net on board a moving ship. A sufficient system design was proposed that uses RTK-GPS for accurate navigation, ANN for ship motion prediction and that guides the UAV with a combination of climb rate and roll angle commands. It is controlled from a simple Neptus plug-in. Successful tests with simulations were performed that separately landed a UAV into a net and predicted ship motion, all within the performance criteria.

A novel approach to ANN time series forecasting has been presented and tested for data containing oscillations around an unknown/varying mean. Pre- and post-processing of the individual data vectors have proven superior to the regular processing of the data set as a whole. In this thesis it was used for ship heading prediction. The yaw oscillates due to waves, but the average yaw angle may also change as a result of rudder and thrust actuations of the ship. The proposed method *current* gave the best result.

This finding is a further improvement of ANN pre- and post-processing and can be utilized in other situations with a similar problem of oscillations around an unknown or varying mean. For instance, all time series with a higher frequency oscillation on top of a slow varying oscillation may also benefit from this approach.

The ANN ship motion prediction presented in this thesis builds upon multiple previous studies, with the contribution of this thesis being increased prediction steps, performance and robustness coming from improved pre- and post-processing and data fusion. Based on the tests performed on data from simulated ships influenced by both waves and current, ANN has proven to be a good tool for ship motion prediction with no need for knowledge of the specific ship dynamics. Both yaw motion and heave displacement was successfully predicted 60 seconds in advance within reasonable error limits. Using data fusion, where time series from multiple variables are utilized, heave displacement prediction showed an increased performance and robustness to data noise. The nature of ANN makes data fusion an easy task.

The only requirement for the forecasting system is that the ship movements for the last five minutes are being logged as time series. These are first used to train the ANN. Later, when performing the forecasting, a period of up to 60 seconds of the last samples are used as inputs to the network. Because the system is trained online and may automatically be retrained repeatedly during operation, it is robust to changes in the weather and will work on any ship type. This is with no need for an operator to set parameters or perform tuning.

Software-in-the-loop (SIL) simulations with the Pixhawk autopilot software APM:Plane have successfully been carried out using DUNE, Neptus, MAVProxy and JSBSim in order to evaluate the performance of different guidance and control systems. The SIL simulations sending only waypoint commands to APM:Plane proved insufficient in a preliminary test, leading to a decoupled guidance system sending desired climb rate and roll angle instead.

The constructed feasible landing path contains the three phases approach, glideslope and final approach (FA), assuring alignment, descent and aiming, respectively. In the horizontal plane it is a purely straight line path based on five waypoints. From the same waypoints a Dubins path is constructed for the desired UAV height, used by the longitudinal guidance system. The slopes are constrained by the maximum allowed descent rate of the UAV and the desired vertical angle of attack into the net. In order to accommodate the possibility of ship motion during landing, a system for a dynamic landing path was created. The initial position and orientation of the landing target can be updated during flight.

For calculation of the desired roll angle, a lateral guidance system based on successive loop closure was designed. The contribution of this thesis is a bank-to-turn (BTT) controller based on proportional-integral-derivative (PID) design that connects an integral line-of-sight (ILOS) heading controller to APM:Plane. Careful tuning of the ILOS and BTT-PID controller provided a stable lateral guidance system that withstands wind disturbance, has few oscillations and leads the UAV into the net with a small error. The designed BTT controller may also be used to connect other heading controllers then ILOS to APM:Plane.

To calculate the desired climb rate, a novel proactive PID controller was constructed for longitudinal guidance. In addition to having the regular PID characteristics of the UAV height being controlled to equal the height of the desired Dubins path, the well defined path derivative is also used in the controller. Ground speed multiplied with the path derivative gives the desired climb rate. By using the calculated climb rate at a lookahead time, the controller manages to react in time for the actual change in desired climb rate to happen, despite the system lag time.

Although previous studies have successfully landed a UAV in a net, this thesis contributes by having increased modularity in the automatic ship landing system, a dynamic landing path and integration of ship motion prediction.

SIL simulations demonstrated great performance of the automatic landing system, where the UAV successfully hit the target within ± 1 meter in both the horizontal and vertical direction. This included both with and without moderate wind and turbulence, for both a stationary and moving target. The dynamic path was updated about 30 seconds before the estimated time of arrival. This was to simulate a change in the predicted ship pose of 5.0 m east, 5.0 m south, 2.5 m up and a rotation of 5.0° clockwise. Despite all the drastic changes happening at the same time, the UAV hit the target within the landing criteria.

An evasive maneuver was prepared and tested for the cases where the landing system predicts that the UAV will miss the target, by detecting that some boundaries are breached during FA. An evasive maneuver was triggered by the system only four seconds before net impact, demonstrating that the UAV had enough time to avoid the net by a large safety distance in both horizontal and vertical direction.

The sufficient DUNE software and connector hardware was developed to successfully connect RTK-GPS navigation messages calculated in DUNE to the Pixhawk GPS connector. This ensures easy integration of accurate navigation without the need of altering the Pixhawk software, thus preserving the modularity of the automatic ship landing system.

6.2 Future Work

During the main UAV SIL simulations of this thesis, a generic UAV model has been used. A fellow student from the NTNU UAV-Lab is developing a mathematical model of the X8. It has been tested with the automatic landing system, showing promising results. When it is fully developed, more realistic SIL simulations can be performed as preparation for real-life tests of the landing system. In addition, a mathematical model of the UAV will enable the use of more advanced controllers.

Further integration and real-life tests should be performed with the accurate navigation system RTK-GPS. Fellow students from the NTNU UAV-Lab are working on systems showing promising results involving the stand-alone system Piksi and software library RTKLIB.

Similar to the way ANN was used for forecasting of yaw motion and heave displacement, it can also be used to predict roll, pitch and the trajectory of the ship. Predictions of roll and pitch can be used in extreme cases where there is danger for the safety of the landing if the angles exceed a predefined operational limit. A change of the landing target position has already successfully been tested as part of the dynamic path, but forecasting of the ship position is yet to be developed.

Data fusion demonstrated an increased performance for the heave displacement prediction. To further increase robustness, more data can be tested for data fusion. This may for instance include velocity measurements, where the heave velocity can be fused in heave displacement prediction and yaw rate can be fused in heading prediction.

When relying on ship motion prediction for landing on a ship with large and fast movements, the system is heavily dependent on a correct estimated time of arrival (ETA). The current ETA calculation in DUNE assumes a constant speed and a straight line towards the next waypoint. This is poor assumptions for a system following a Dubins path that lowers the ground speed right before net impact.

Appendix A

Guidance and Control System

This appendix presents some preliminary experiments of the guidance and control system in Section A.1 and tuning values for the ILOS controller, bank-to-turn PID controller and PID height controller in Section A.2, A.3 and A.4, respectively.

A.1 Preliminary Experiments

This section contains some preliminary experiments done as preparatory work to find the final guidance and control system for the UAV landing system.

A.1.1 Waypoint Guidance

Figure A.1 and Figure A.2 below demonstrates the performance of the APM:Plane height controller when waypoints are sent directly. The desired height seen in the plots is calculated by APM:Plane based on the five waypoints that was sent.

To ensure APM:Plane uses the defined waypoints directly when controlling the UAV, instead of DUNE calculating and sending desired climb rate and roll angle, the APM's flight mode *GUIDED* is used. It gets activated by setting the DUNE autopilot parameter *Ardupilot Tracker* to true when defining the waypoints making up the landing path. Otherwise the similar procedure is used as in the SIL simulation described in Section 4.5.

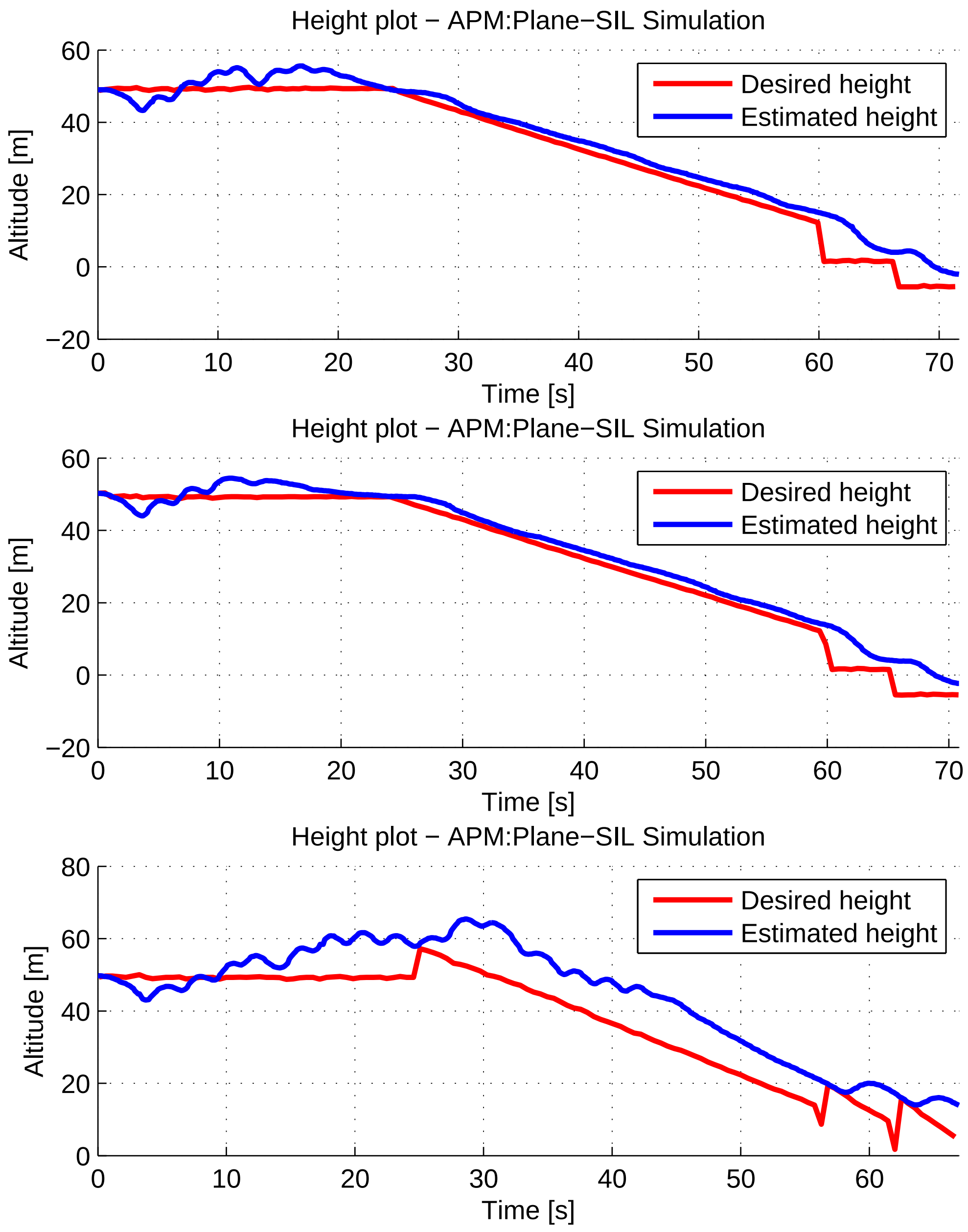


Figure A.1: SIL simulation with waypoints sent directly to APM:Plane

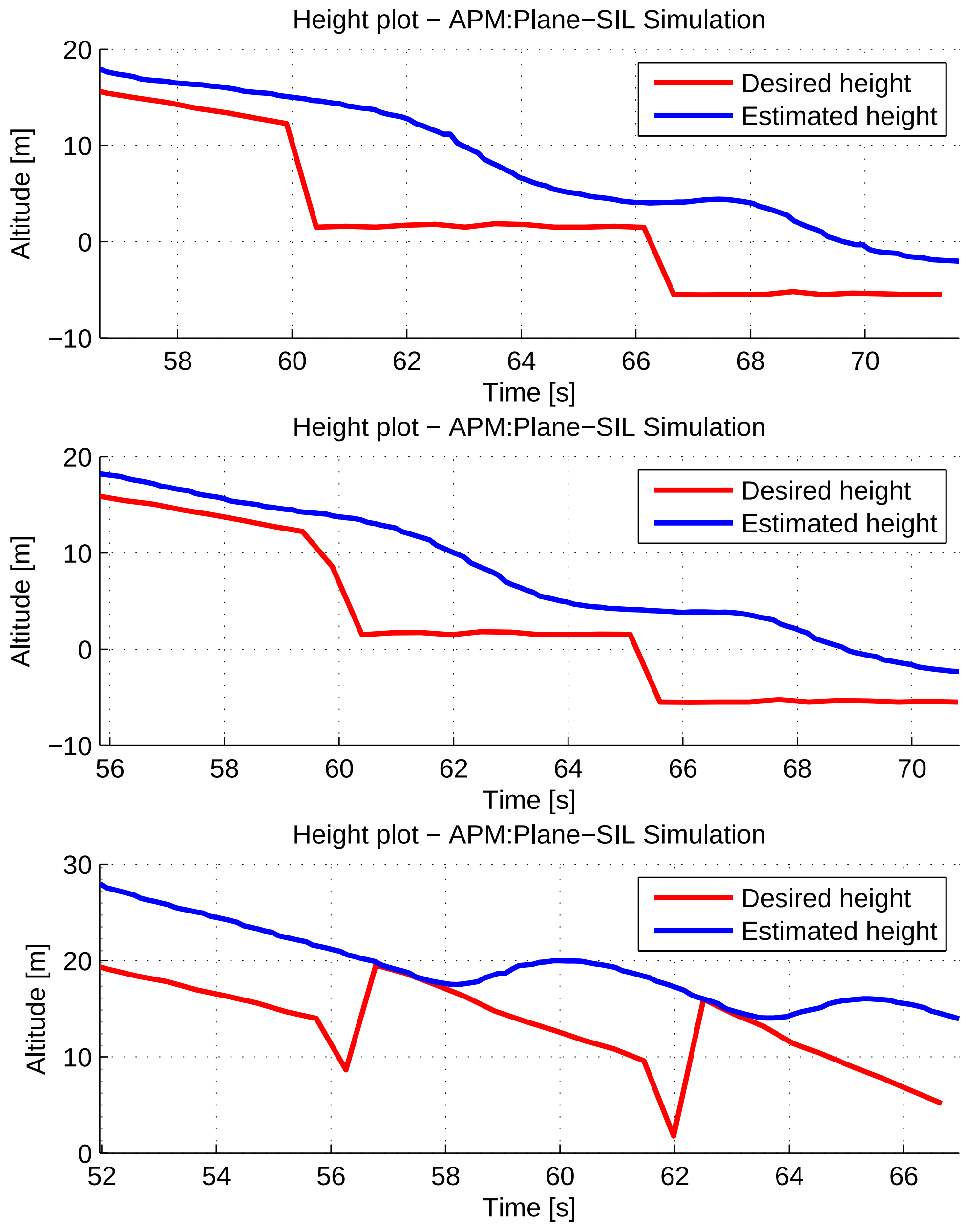


Figure A.2: SIL simulation with waypoints sent directly to APM:Plane, 15 last seconds

A.1.2 Aerosonde Lateral Controller

Aerosonde is a lateral controller already programmed in DUNE. A test using this controller was conducted according to the SIL simulation described in Section 4.5. Figure A.3 and Figure A.4 illustrate with plots of the cross-track errors from two flights how the controller in lateral direction leads to both oscillations and constant deviation from the desired path. This is despite that there was no wind present during the simulations. Tuning has been performed in an effort to minimize the oscillations.

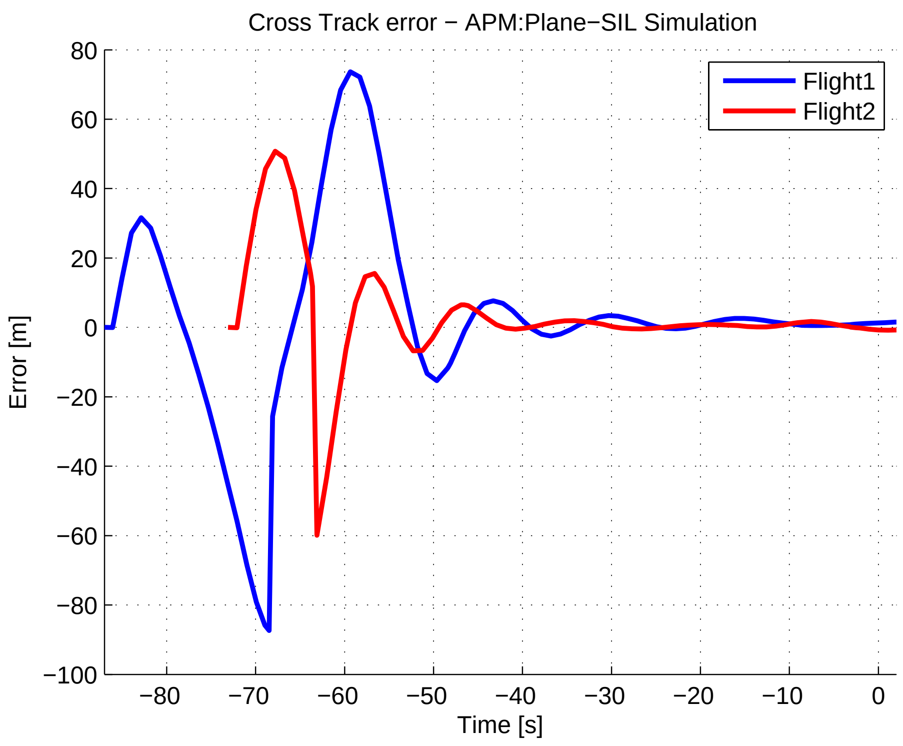


Figure A.3: Cross-track error from SIL simulation with Aerosonde lateral controller

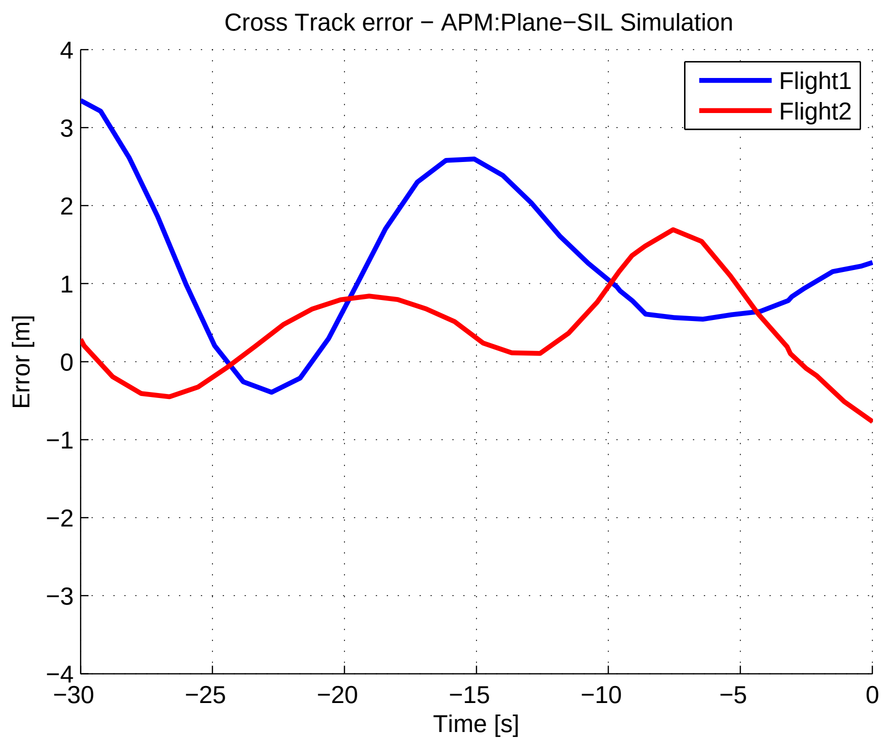


Figure A.4: Cross-track error from SIL simulation with Aerosonde lateral controller, last 30 seconds

A.1.3 Sliding Mode Height Controller

A test was performed with a sliding mode height controller already programmed in DUNE. The test was conducted according to the SIL simulation described in Section 4.5. Figure A.5 demonstrates how the sliding mode controller works as a set-point regulator with a constant set-point between the waypoints.

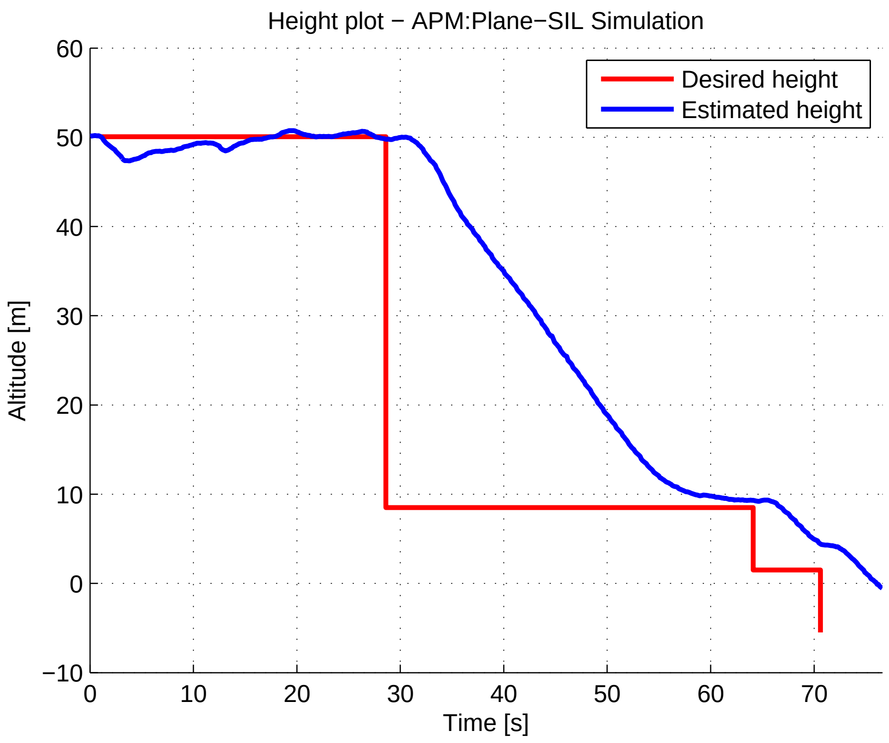


Figure A.5: Height plot from SIL simulation with sliding mode height controller

A.2 ILOS Controller Tuning Values

Table A.1 displays the most relevant integral line-of-sight (ILOS) controller parameters related to the lateral guidance system described in Subsection 4.4.1.

Table A.1: Tuning values for the ILOS controller implemented in DUNE

Tuning parameter	Value
Control Frequency	10
ILOS Lookahead Distance	50.0
ILOS Integrator Gain	2.5

A.3 Bank-To-Turn Controller Tuning Values

Table A.2 displays the most relevant bank-to-turn (BTT) controller parameters related to the lateral guidance system described in Subsection 4.4.1. It were tested on several straight line response tests, giving the controller a desired heading of constant 0°. Figure A.6 and A.7 illustrate the response from two tests where the initial COG was approximately −120° and −50° respectively.

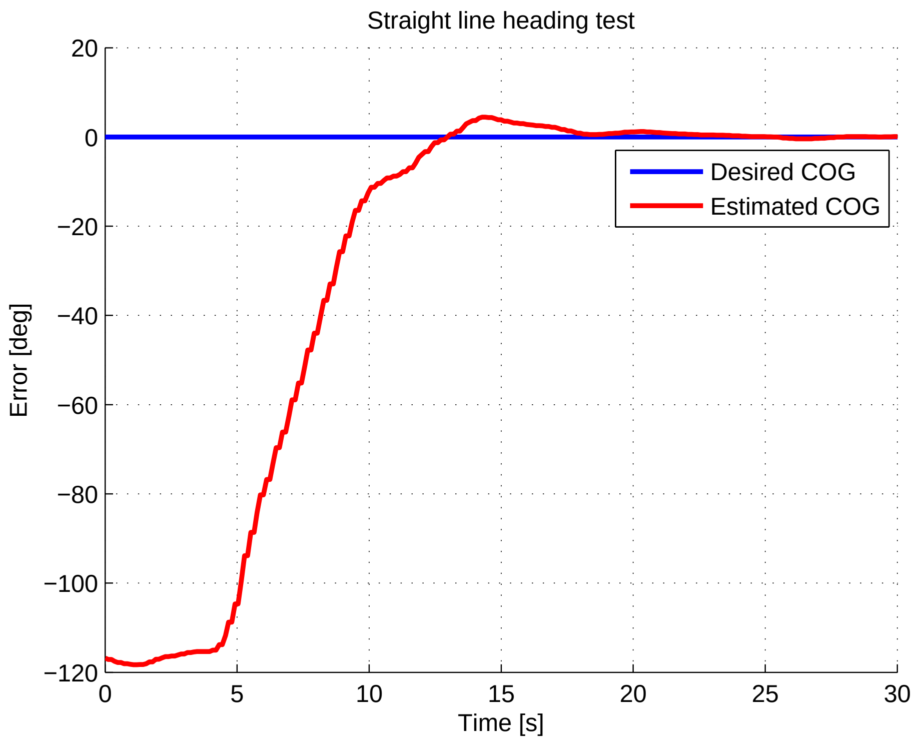


Figure A.6: Straight line test of BTT response, from $-120°$

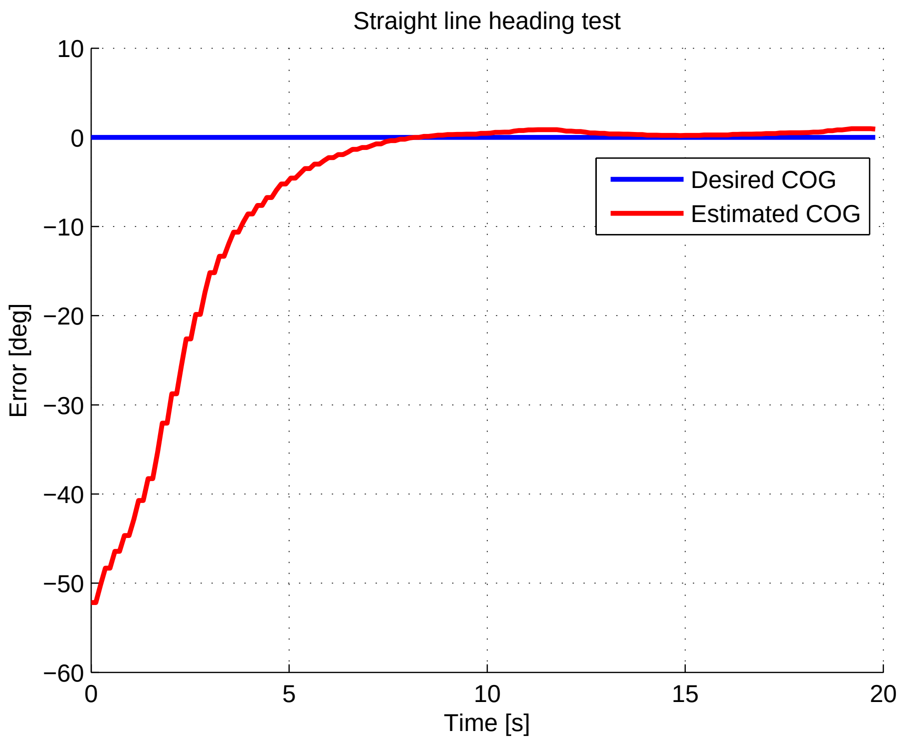


Figure A.7: Straight line test of BTT response, from $-50°$

Table A.2: Tuning values for the BTT controller implemented in DUNE

Tuning parameter	Value
PID Gain Prop (K_p)	1.2
PID Gain Int (K_i)	0.1
PID Gain Der (K_d)	0.6
Maximum Bank	35.0
Anti Windup Maximum Bank	3.0

A.4 PID Height Controller Tuning Values

Table A.3 displays the most relevant proportional-integral-derivative (PID) height controller parameters related to the lateral guidance system described in Subsection 4.4.2.

Table A.3: Tuning values for the PID height controller

Tuning parameter	Value
Approach – PID Gain Prop (K_p)	0.15
Approach – PID Gain Int (K_i)	0.05
Approach – PID Gain Der (K_d)	0.10
Descend – PID Gain Prop (K_p)	0.80
Descend – PID Gain Int (K_i)	0.20
Descend – PID Gain Der (K_d)	1.00
Climb Rate Min	-2.0
Climb Rate Max	1.5
Anti Windup Max Climb Rate	0.60
PID Lookahead Time	0.5

Appendix B

Time Series Forecasting

This appendix contains supplementary information and results for the experiments regarding time series forecasting. Section B.1 contains a Matlab code snippet to produce a simple yaw motion example, Section B.2 presents information for the ship simulation, Section B.3 presents additional results for experiments regarding ship heading prediction, Section B.4 presents additional results for simple heave displacement prediction and Section B.5 presents additional results for heave displacement prediction using data fusion.

B.1 Oscillations Around a Varying Mean

```
a = -5;
b = 125;

shift = 300;
slope = 1/30;

x = 0:0.05:500;
z = a + (b-a)*logsig(slope*(x-shift));
yaw = 5*sin(0.5*x) + z;
```

Listing B.1: Matlab code to produce the simple yaw motion used in Subsection 5.1.2

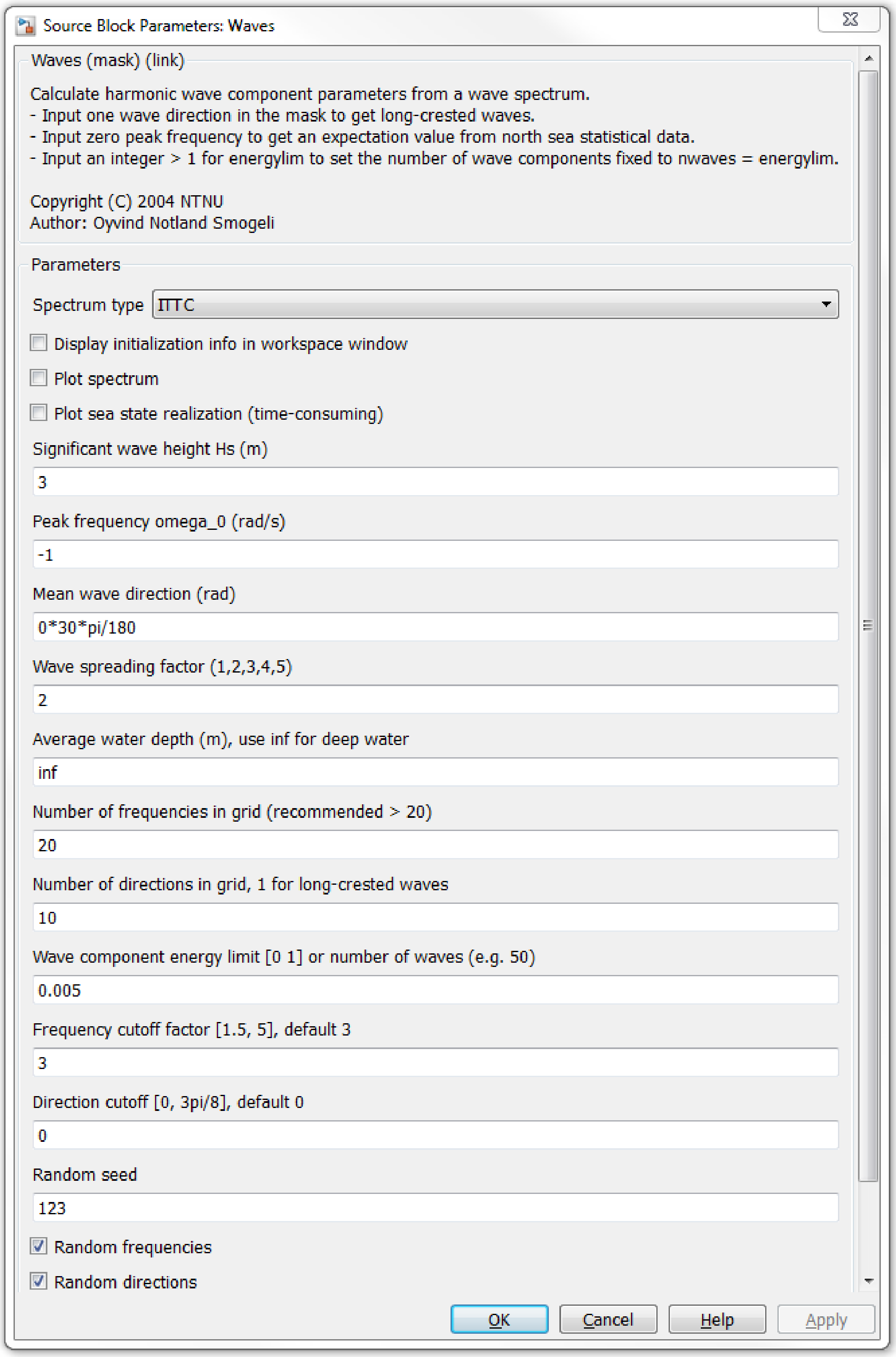
B.2 Ship Simulation

The main dimensions for the simulated ships are found in Table B.1, where m denotes mass, L_{pp} is length between perpendiculars, B is breadth and T is the draught. More information about the ship descriptions can be found in a description paper by Fossen [23].

Table B.1: ShipX main dimensions

Vessel	$mass\ [tonnes]$	$L_{pp}\ [m]$	$B\ [m]$	$T\ [m]$
S175	24.609	175.0	25.4	9.5
Supply Vessel	6.362	82.8	19.2	6.0

Figure B.1 presents the settings used for the wave block in the ship motion simulations.



Source Block Parameters: Waves

Waves (mask) (link)

Calculate harmonic wave component parameters from a wave spectrum.
- Input one wave direction in the mask to get long-crested waves.
- Input zero peak frequency to get an expectation value from north sea statistical data.
- Input an integer > 1 for energylim to set the number of wave components fixed to nwaves = energylim.

Copyright (C) 2004 NTNU
Author: Oyvind Notland Smogeli

Parameters

Spectrum type: ITTC

☐ Display initialization info in workspace window

☐ Plot spectrum

☐ Plot sea state realization (time-consuming)

Significant wave height Hs (m): 3

Peak frequency omega_0 (rad/s): -1

Mean wave direction (rad): 0*30*pi/180

Wave spreading factor (1,2,3,4,5): 2

Average water depth (m), use inf for deep water: inf

Number of frequencies in grid (recommended > 20): 20

Number of directions in grid, 1 for long-crested waves: 10

Wave component energy limit [0 1] or number of waves (e.g. 50): 0.005

Frequency cutoff factor [1.5, 5], default 3: 3

Direction cutoff [0, 3pi/8], default 0: 0

Random seed: 123

☑ Random frequencies

☑ Random directions

Figure B.1: Wave settings for the ship simulation

Figure B.2 illustrates the power spectral density (PSD) plot of the heave motion for ShipX: 175. The PSD is calculated using the discrete Fourier transform from Matlab. It shows a wide peak centered at 0.1 Hz.

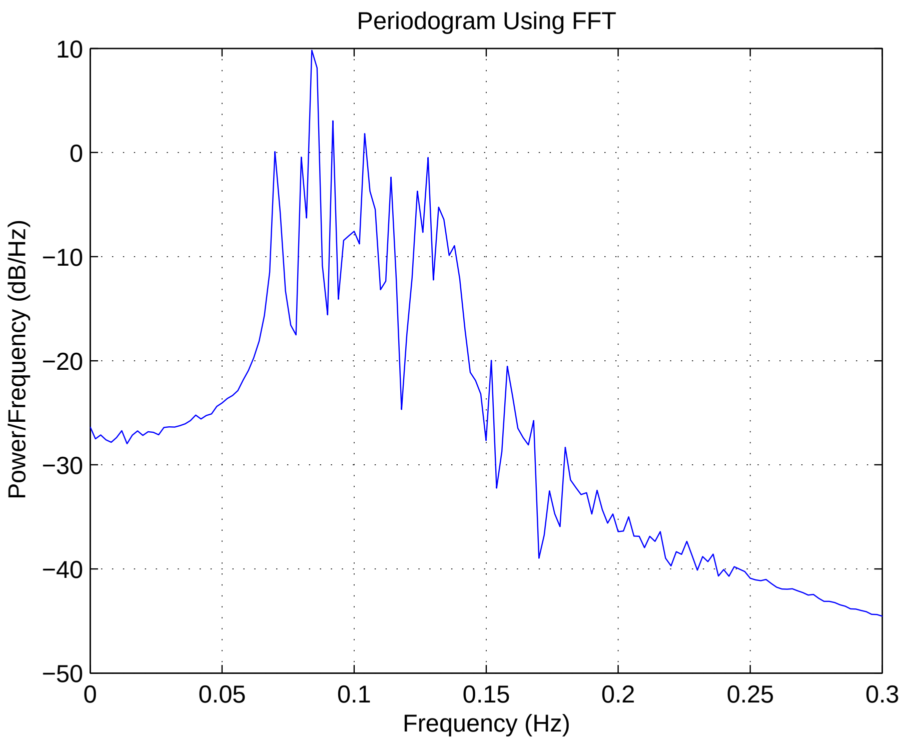


Figure B.2: PSD plot of the heave motion with ShipX: S175

Figure B.3 illustrates a plot highlighting all the calculated peaks in a heave displacement plot. These peaks are where the "error at peaks" are measured.

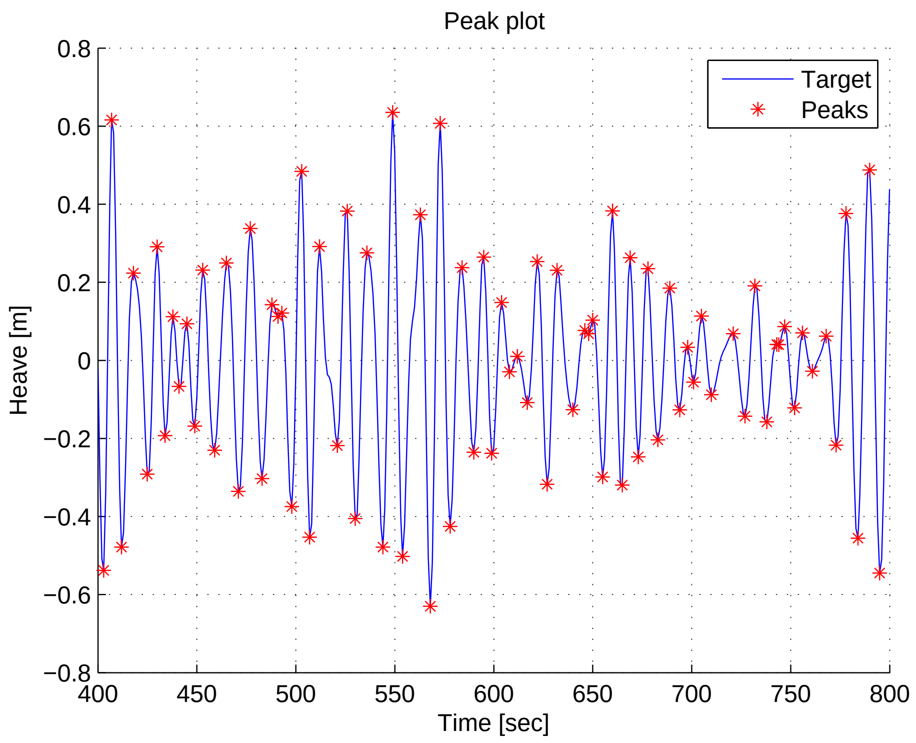


Figure B.3: Peaks in a heave displacement plot

B.3 Ship Heading Prediction

Below follows results from the tests described in Subsection 5.1.5.

B.3.1 Methods and Activation Functions

Tests have been performed to find the best activation function for the hidden neurons. As can be read from Table B.2 and Table B.3, the linear activation function, called *purelin* in Matlab, is the best alternative for all methods. This is both considering the lowest MSE and the shortest training time. The other activation functions tested were hyperbolic tangent (*tansig*), log-sigmoid (*logsig*) and radial basis (*radbas*).

Table B.2: MSE [°] for all methods and activation functions

Method	purelin	tansig	logsig	radbas
Unchanged	0.0928	16.29	19.03	28.03
Mean	0.0301	0.0320	0.0317	0.0342
Median	0.0298	0.0341	0.0344	0.0352
Current	0.0305	0.0326	0.0329	0.0334
Normalization	0.0504	0.0748	0.0769	0.0787

Table B.3: Training time [s] for all methods and activation functions

Method	purelin	tansig	logsig	radbas
Unchanged	0.507	0.671	0.686	0.718
Mean	0.515	0.585	0.554	0.601
Median	0.476	0.530	0.577	0.601
Current	0.515	0.523	0.562	0.569
Normalization	0.312	0.608	0.593	0.577

B.3.2 Methods and Hidden Neurons

Different number of hidden neurons were tested against all the methods. Below follows both tables with individual results and combined plots.

Table B.4: Method - *Unchanged*

Hidden nodes	Training time [s]	MSE [°]	MAE [°]
1	0.164	0.1309	0.3317
2	0.171	0.1246	0.3049
3	0.546	0.1075	0.2876
4	0.515	0.1175	0.3058
5	0.491	0.1141	0.3089
6	0.515	0.1106	0.2923

Table B.5: Method - *Mean*

Hidden nodes	Training time [s]	MSE [°]	MAE [°]
1	0.140	0.0307	0.1433
2	0.140	0.0306	0.1422
3	0.530	0.0303	0.1430
4	0.491	0.0306	0.1428
5	0.530	0.0307	0.1429
6	0.530	0.0305	0.1434

Table B.6: Method - *Median*

Hidden nodes	Training time [s]	MSE [°]	MAE [°]
1	0.140	0.0307	0.1427
2	0.140	0.0313	0.1432
3	0.530	0.0306	0.1432
4	0.523	0.0306	0.1448
5	0.476	0.0305	0.1433
6	0.484	0.0309	0.1424

Table B.7: Method - *Current*

Hidden nodes	Training time [s]	MSE [°]	MAE [°]
1	0.156	0.0309	0.1437
2	0.140	0.0306	0.1441
3	0.452	0.0302	0.1419
4	0.515	0.0305	0.1435
5	0.499	0.0305	0.1434
6	0.523	0.0305	0.1420

Table B.8: Method - *Normalization*

Hidden nodes	Training time [s]	MSE [°]	MAE [°]
1	0.140	0.0527	0.1893
2	0.133	0.0546	0.1931
3	0.406	0.0538	0.1912
4	0.374	0.0521	0.1867
5	0.367	0.0522	0.1845
6	0.374	0.0515	0.1851

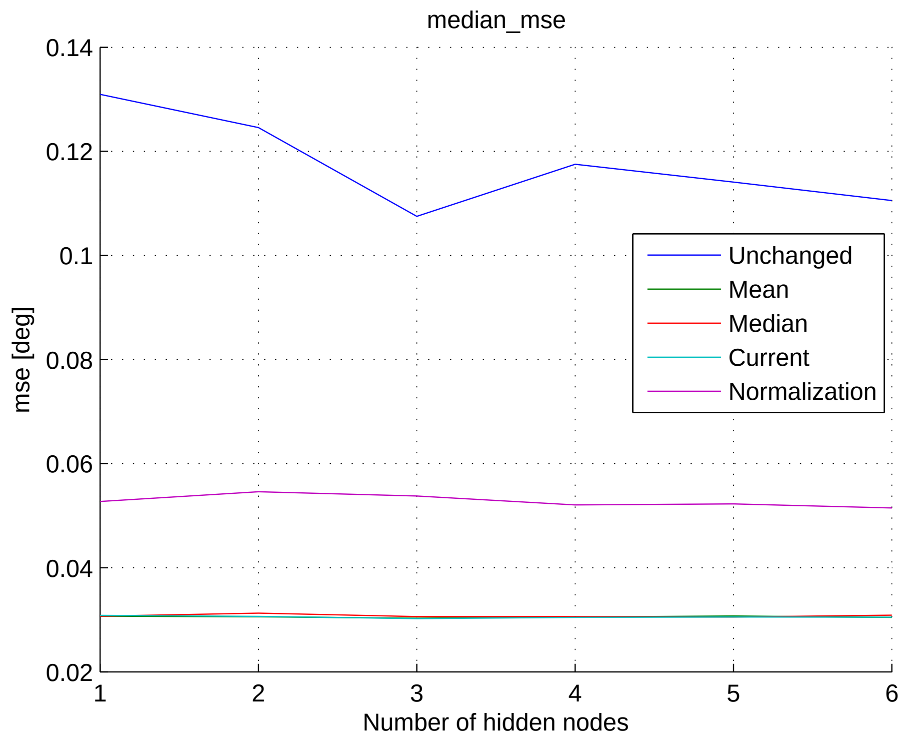


Figure B.4: MSE for different methods and numbers of hidden nodes

B.4 Heave Displacement Prediction

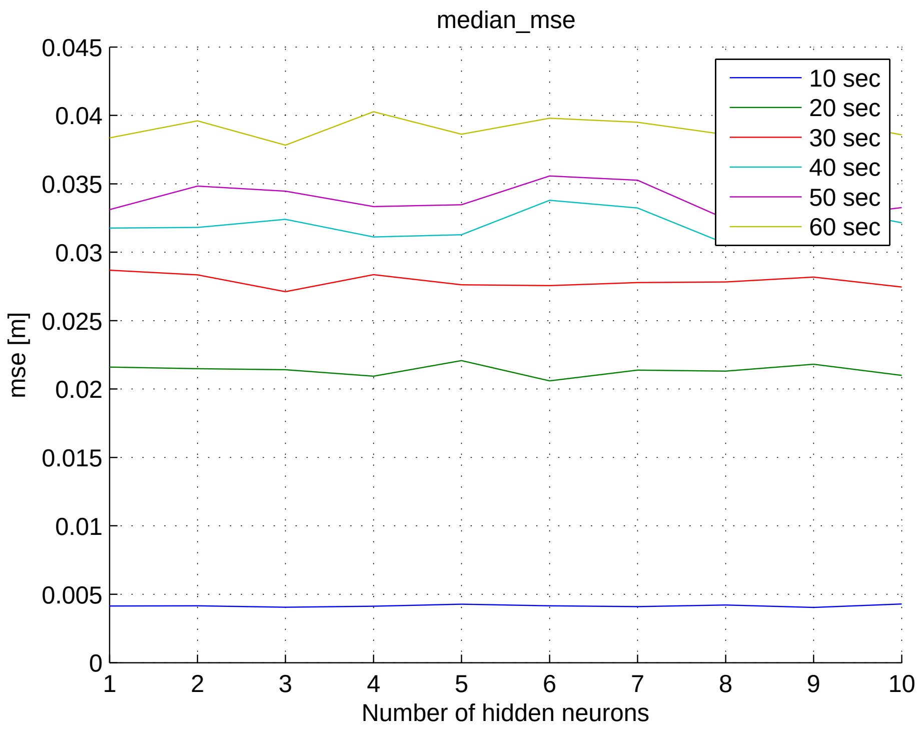


Figure B.5: MSE for different prediction intervals and number of hidden neurons

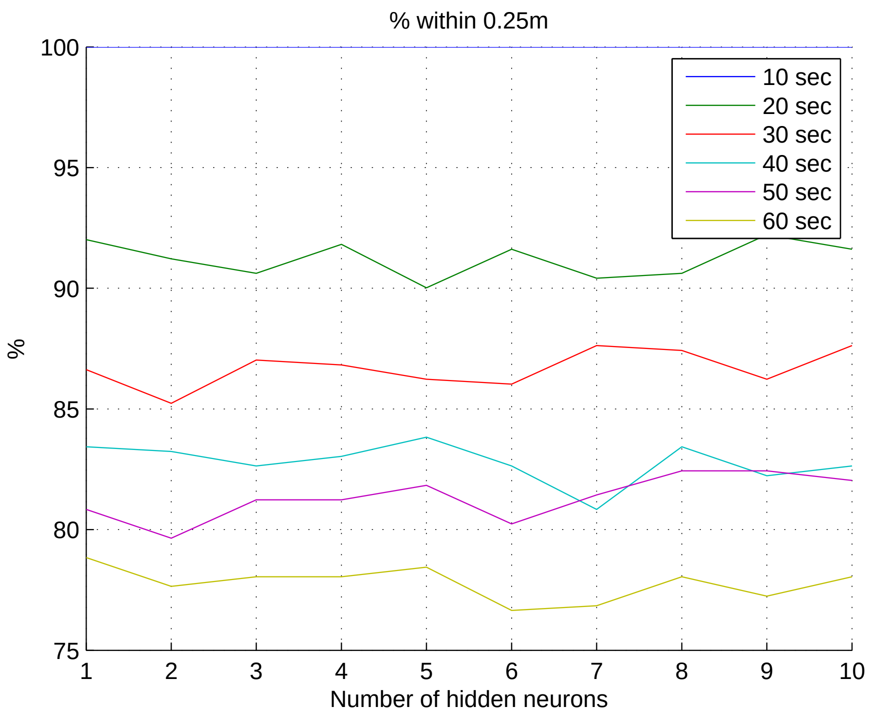


Figure B.6: Percentage of errors withing 0.25 m at peaks for different prediction intervals and number of hidden neurons

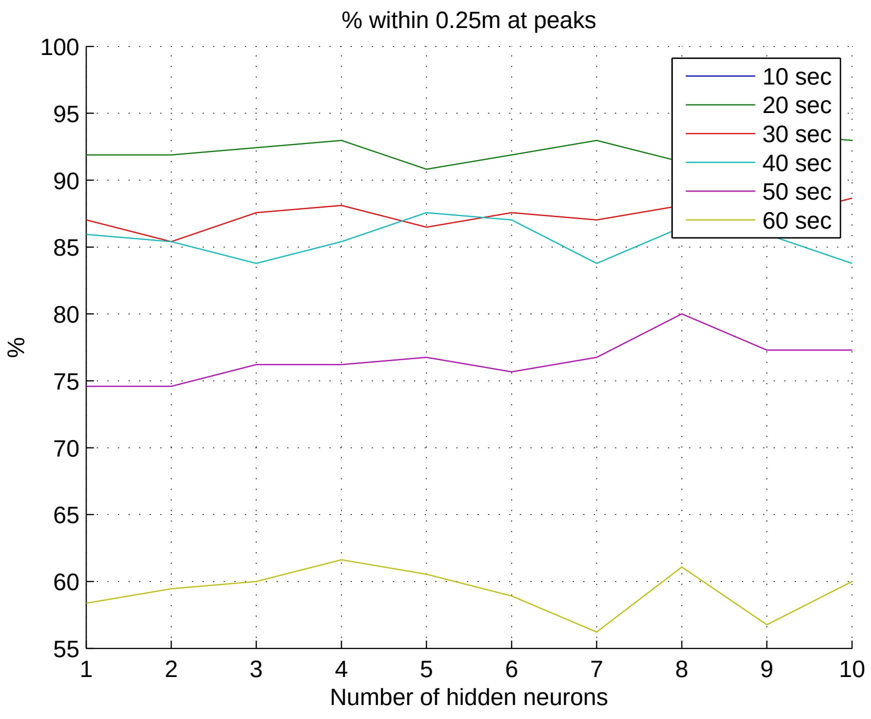


Figure B.7: Percentage of errors withing 0.25 m for different prediction intervals and number of hidden neurons

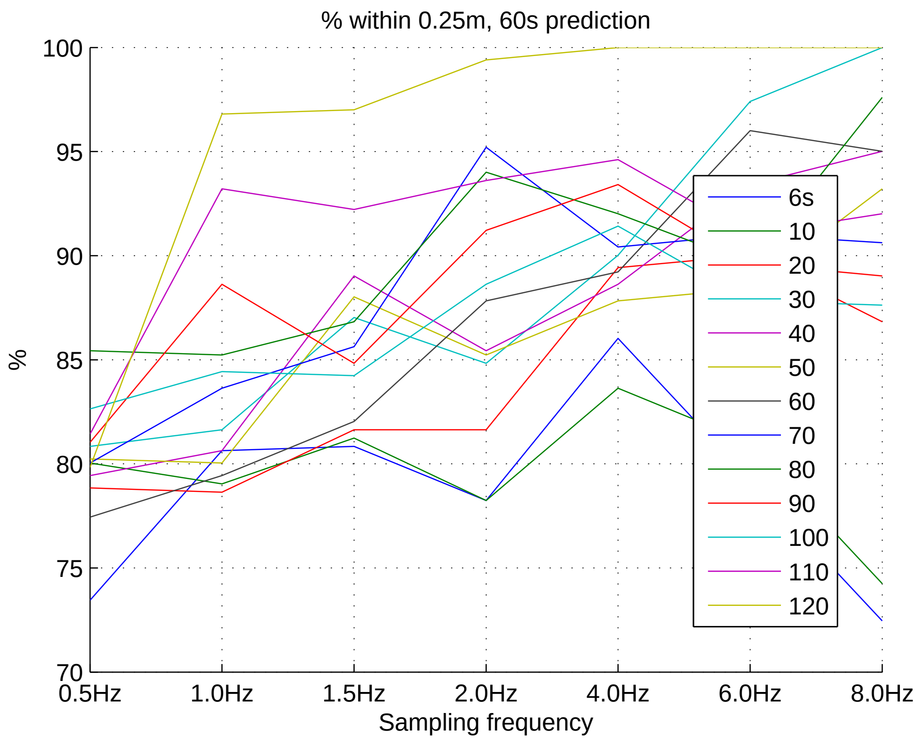


Figure B.8: 60 seconds prediction step performance with combinations of input time range/sampling frequency

B.5 Heave Displacement Prediction - Data Fusion

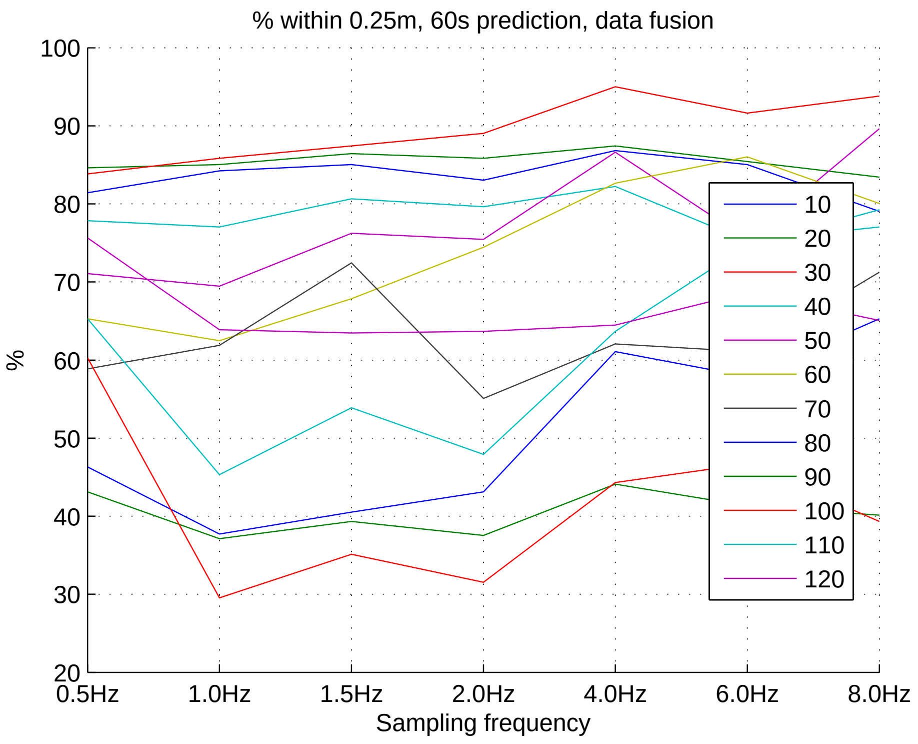


Figure B.9: 60 seconds prediction step performance, data fusion, with combinations of input time range/sampling frequency

Appendix C

Landing Simulation, Software-In-the-Loop

This appendix contains supplementary information and results for the software-in-the-loop landing simulations. Section C.1 presents information about the landing path, Section C.2 presents information about the loiter alignment path, Section C.3 presents additional results for the static path test without wind, Section C.4 presents additional results for the static path test with wind and Section C.5 presents additional results for the dynamic path test with wind.

C.1 Waypoint Path Calculation

During simulation the waypoints building up the basis of the initial landing paths were calculated from the values given in Table C.1, which resulted in the 5 waypoints given in Table C.2.

Table C.1: Values defining the landing path from Figure 4.10. The UAV height is assumed 50.0 m (h2), but varies with every individual flight

Variable	Value
land_lat	63.628600°
land_lon	9.727570°
land_head	66.5°
net_hight	3.0 m
min_turn_radius	150.0 m
h0	$d0 * \tan\alpha = 7.0$ m
h1	$d1 * \tan\alpha = 7.0$ m
h2	50.0 m
d0	100.0 m
d1	100.0 m
d2	$(h2 - h1)/\tan\beta = 615.0$ m
d3	$2 * min\_turn\_radius = 300.0$ m
α	4°
β	4°

Table C.2: Calculated waypoints for the landing path described in Table C.1

Waypoint	Latitude [°]	Longitude [°]	Z_Altitude [m]	Speed [m/s]
Wp1	63.623346	9.700949	50.0	18
Wp2	63.624438	9.706479	50.0	18
Wp3	63.628236	9.725726	8.5	16
Wp4	63.628600	9.727570	1.5	16
Wp5	63.628958	9.729420	-5.5	16

C.2 Loiter Alignment Path

Table C.3 below presents the parameters defining the loiter path followed by the simulated UAV before the landing sequence is initiated. It helps the UAV align in the same way before every landing, making comparisons easier.

Table C.3: Values defining the loiter path used before initiating landing

Variable	Value
Latitude	63.627633°
Longitude	9.705761°
Z	50.0 m
Z-Units	HEIGHT
Direction	Clockwise
Speed	18.0 m/s
Radius	150.0 m

C.3 Static Path, No Wind

Figure C.1 and Figure 5.19 below present plots for five flights of altitude error and cross-track error respectively. The entire landing sequence is included in the plots.

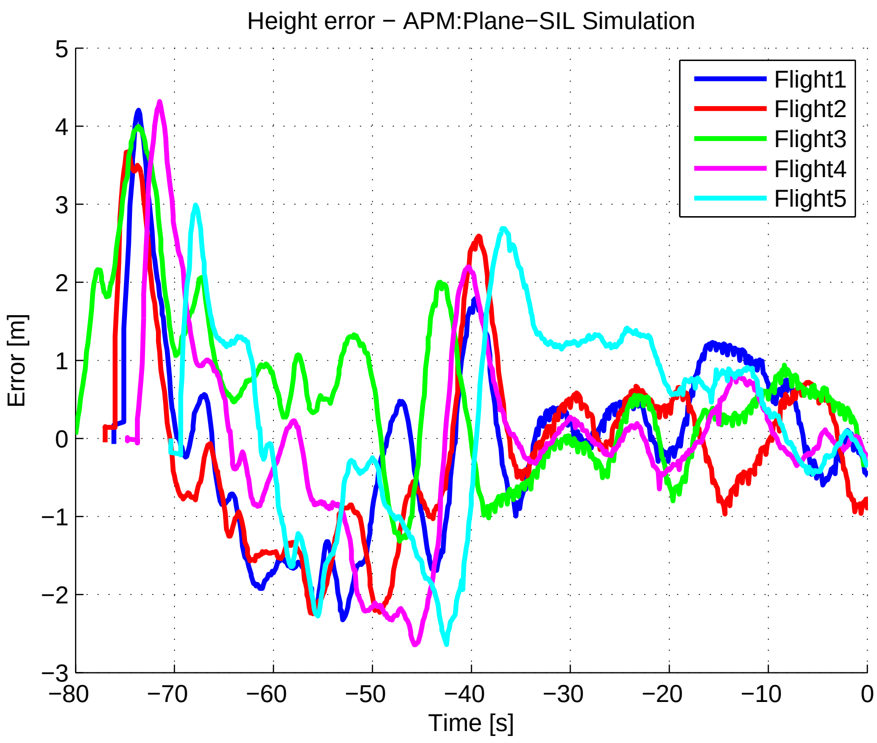


Figure C.1: Altitude error of 5 flights, static path - no wind

C.4 Static Path, Wind and Turbulence

Figure C.2 and Figure C.3 below present plots of an estimation of the simulated wind direction and wind speed respectively. Only the wind from a single simulation is presented. Note that what is displayed is not the direct wind output from the simulator JSBSim, but an estimation done in DUNE based on how the UAV is affected during flight.

Figure C.4 and Figure C.5 below present plots for five flights of altitude error and cross-track error respectively. The entire landing sequence is included in the plots.

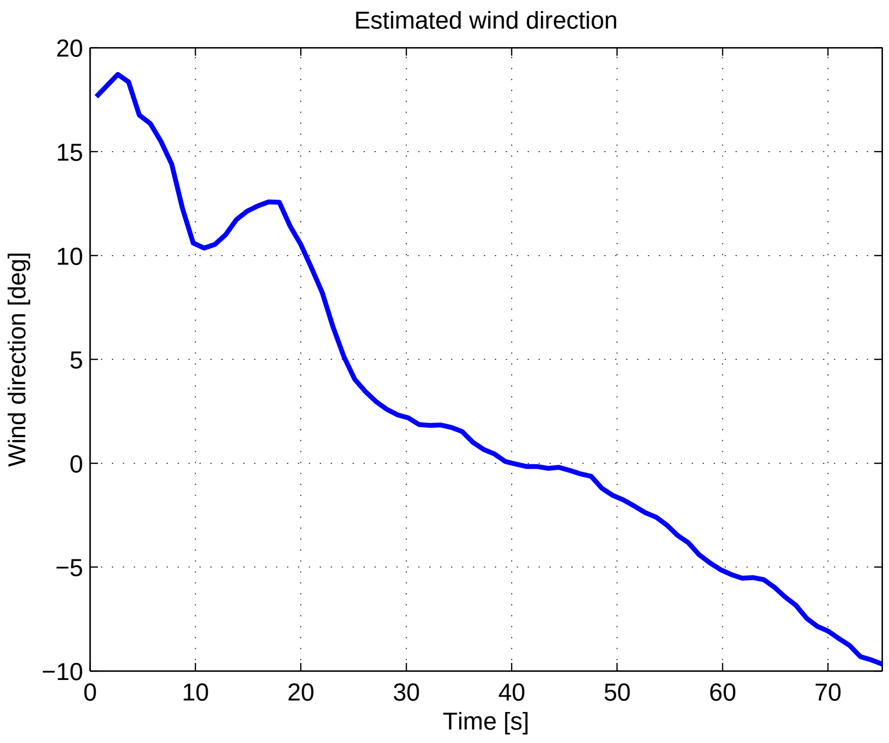


Figure C.2: Wind direction during static path landing - wind and turbulence

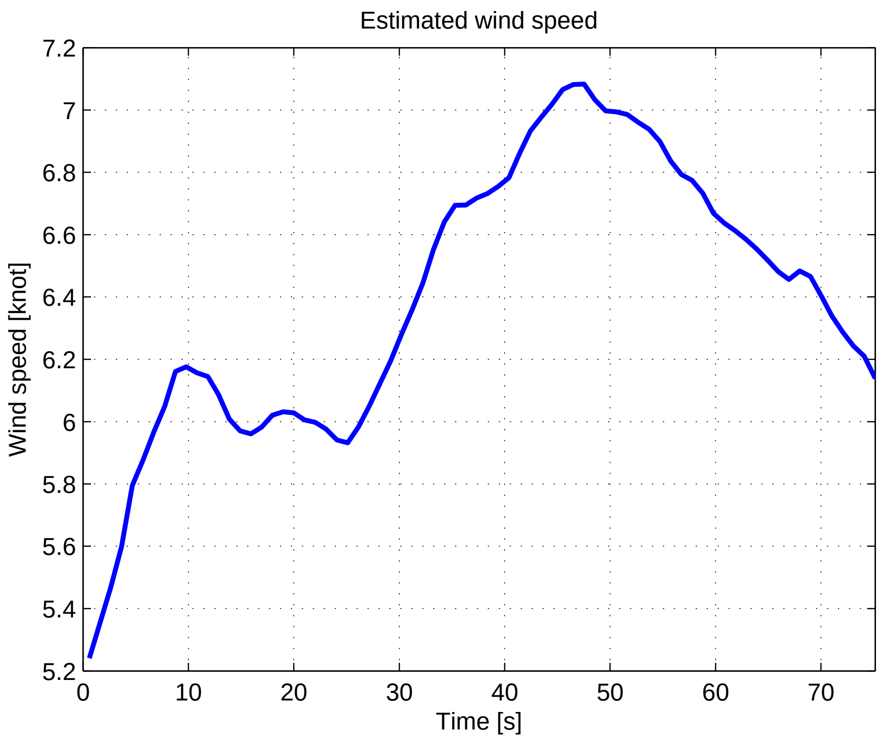


Figure C.3: Wind speed during static path landing - wind and turbulence

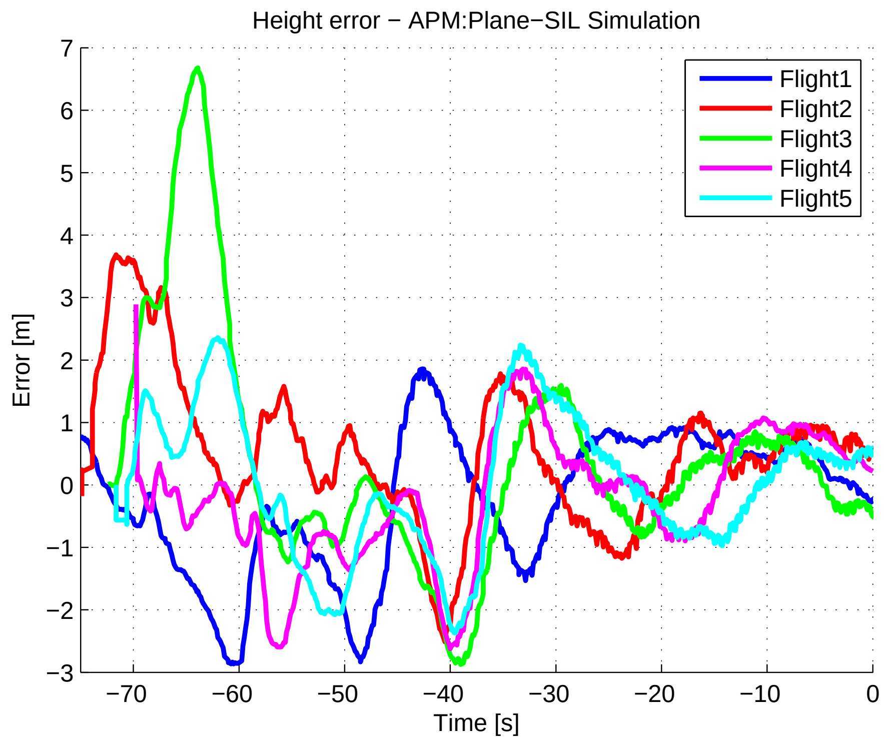


Figure C.4: Altitude error of 5 flights, static path - wind and turbulence

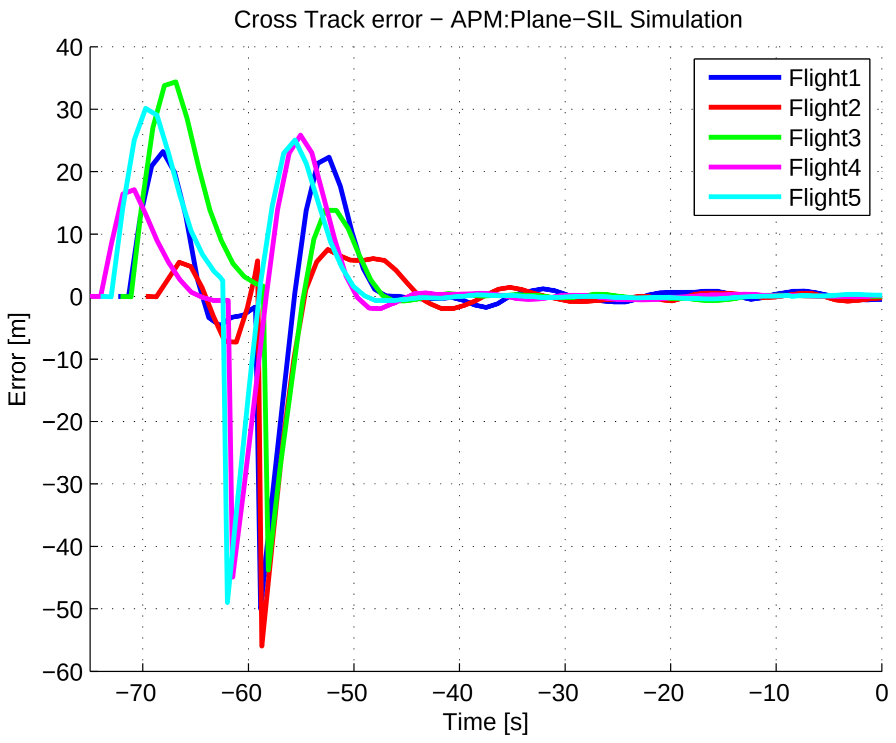


Figure C.5: Cross-track error of 5 flights, static path - wind and turbulence

C.5 Dynamic Path, Wind and Turbulence

Figure C.6 and Figure C.7 below present plots for five flights of altitude error and cross-track error respectively. The period from path update to net impact is included in the plots.

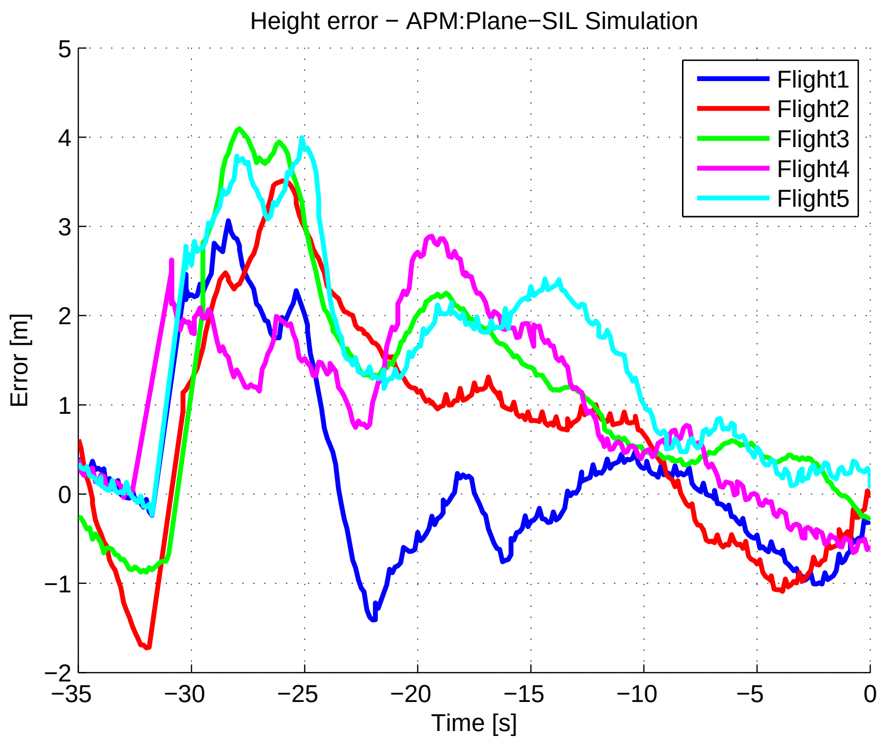


Figure C.6: Altitude error of 5 flights, from path update, dynamic path - wind and turbulence

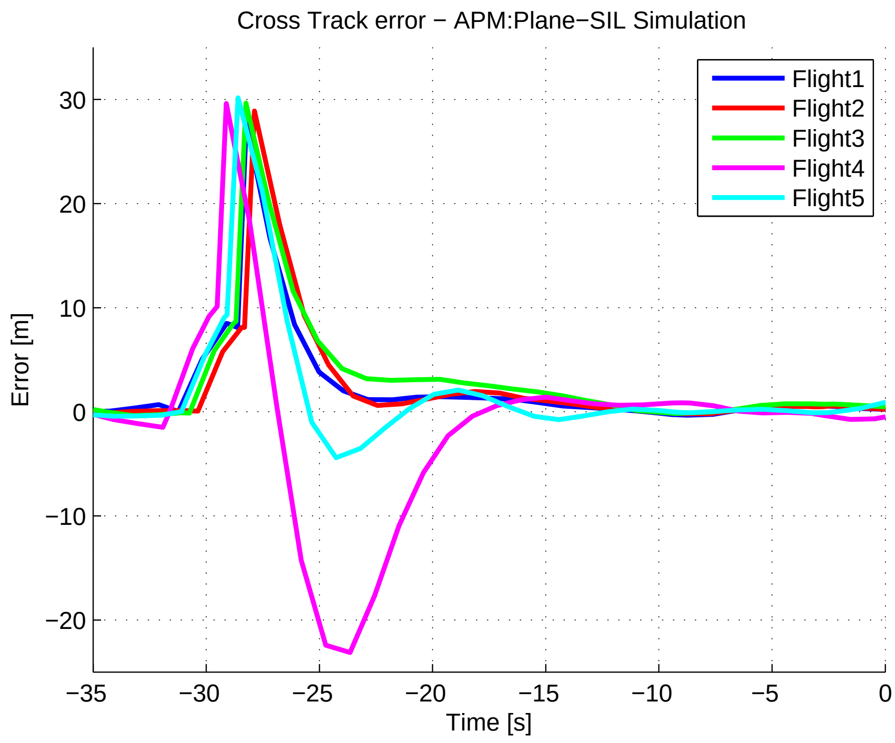


Figure C.7: Cross-track error of 5 flights, from path update, dynamic path - wind and turbulence

Bibliography

[1] 3DRobotics (2014a). ArduPlane | Fixed-wing aircraft UAV. http://plane.ardupilot.com. Accessed: 14.11.2014.

[2] 3DRobotics (2014b). Flight modes | ArduPlane. http://plane.ardupilot.com/wiki/flying/flight-modes. Accessed: 16.12.2014.

[3] 3DRobotics (2014c). Mission Planner | Ground Station. http://planner.ardupilot.com. Accessed: 25.11.2014.

[4] 3DRobotics (2014d). Pixhawk Autopilot Specifications. https://store.3drobotics.com/products/3dr-pixhawk. Accessed: 11.12.2014.

[5] 3DRobotics (2014e). Tecs (total energy control system) for speed and height tuning guide. http://plane.ardupilot.com/wiki/tecs-total-energy-control-system-for-speed-height-tuning-guide. Accessed: 18.12.2014.

[6] BeagleBoard.org (2014). Beaglebone Black Specifications. http://www.elinux.org/Beagleboard:BeagleBoneBlack. Accessed: 11.12.2014.

[7] Beard, R. W. and McLain, T. W. (2012). *Small Unmanned Aircraft: Theory and Practice*. Princeton University Press.

[8] Berndt, J. S. (2011). JSBSim: An Open Source Flight Dynamics Model in C++. *AIAA Modeling and Simulation Technologies Conference and Exhibit (2004)*.

[9] Box, G. E. P. and Jenkins, G. M. (1970). *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA.

[10] Box, G. E. P. and Jenkins, G. M. (1976). *Time Series Analysis: Forecasting and Control*. Holden-Day, San Francisco, CA.

[11] Breivik, M. and Fossen, T. I. (2009). Guidance laws for autonomous underwater vehicles. In Inzartsev, A. V., editor, *Underwater Vehicles*, number December, chapter 4, pages 51–76. InTech.

[12] Broomhead, D. S. and Lowe, D. (1988). Multivariable Functional Interpolation and Adaptive Networks. *Complex systems*, 2:321–355.

[13] Caharija, W., Candeloro, M., Pettersen, K. Y., and Sørensen, A. J. (2012). Relative Velocity Control and Integral LOS for Path Following of Underactuated Surface Vessels. *IFAC Proceedings Volumes (IFAC-PapersOnline)*, 9(PART 1):380–385.

[14] Canny, J. and Reif, J. (1987). New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 49–60, Los Angeles, CA, USA. IEEE.

[15] Charalambous, C. (1992). Conjugate gradient algorithm for efficient training of artificial neural networks. *IEE Proceedings G Circuits, Devices and Systems*,

139(3):301–310.

[16] Cortes, N. B. (1999). Predicting ahead on ship motions using Kalman filter implementation. *Department of Aerospace Engineering*, page 68.

[17] Crump, M. R. (2002). The Dynamics and Control of Catapult Launching Unmanned Air Vehicles from Moving Platforms. *Department of Aerospace Engineering*, page 264.

[18] Cybenko, G. (1989). Approximations by Superpositions of Sigmoidal Functions. *Mathematics of Control, Signals, and Systems*, 2(4):303–314.

[19] Davis, I. L. (1995). Sensor Fusion for Autonomous Outdoor Navigation Using Neural Networks. *Proceedings 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots*, 3.

[20] Deo, M. C. (2010). Artificial neural networks in coastal and ocean engineering. *Indian Journal of Geo-Marine Science*, 39(December):589–596.

[21] Dubins, L. E. (1957). On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *American Journal of Mathematics (1957)*, 79(3):497–516.

[22] Elkaim, G. H., Lie, F. A. P., and Gebre-Egziabher, D. (2012). Principles of Guidance, Navigation and Control of UAVs. In Valavanis, K. P. and Vachtsevanos, G. J., editors, *Handbook of Unmanned Aerial Vehicles*, chapter 18, pages 347–380. Springer.

[23] Fossen, T. I. (2008). Description of MSS Vessel Models: Configuration Guidelines for Hydrodynamic Codes.

[24] Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons, Ltd, Chichester, UK.

[25] Fossen, T. I. and Perez, T. (2010). MSS. Marine Systems Simulator. http://www.marinecontrol.org. Accessed: 07.04.2015.

[26] Fritsch, F. N. and Carlson, R. E. (1980). Monotone Piecewise Cubic Interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246.

[27] Funahashi, K.-I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural Networks*, 2(3):183–192.

[28] Geva, A. B. (1998). ScaleNet - Multiscale neural-network architecture for time series prediction. *IEEE Transactions on Neural Networks*, 9(6):1471–1482.

[29] Haykin, S. O. (2008). *Neural Networks and Learning Machines: International Version*. Pearson, 3rd edition.

[30] Hestenes, M. R. and Stiefel, E. (1952). Methods of Conjugate Gradients for Solving Linear Systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436.

[31] Insitu Inc (2013). ScanEagle System. http://www.insitu.com/systems/scaneagle. Accessed: 16.12.2014.

[32] ITTC (2015). International Towing Tank Conference. http://ittc.info/. Accessed: 09.04.2015.

[33] Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45.

[34] Khan, A., Bil, C., Marion, K., and Crozier, M. (2004). Real Time Prediction of Ship Motion and Attitudes Using Advanced Prediction Techniques. pages 1–10.

[35] Khan, A., Bil, C., and Marion, K. E. (2005a). Ship Motion Prediction for Launch and Recovery of Air Vehicles. In *Proceedings of MTS/IEEE OCEANS, 2005*, volume 2005, pages 1–10.

[36] Khan, A., Bil, C., and Marion, K. E. (2005b). Theory and Application of Artificial Neural Networks for the Real Time Prediction of Ship Motion. In *Knowledge-Based Intelligent Information and Engineering Systems*, volume 1, pages 1064–1069, Melbourne, Australia. Springer-Verlag.

[37] Khashei, M. and Bijari, M. (2010). An artificial neural network (p, d, q) model for timeseries forecasting. *Expert Systems with Applications*, 37(1):479–489.

[38] Khashei, M. and Bijari, M. (2011). A novel hybridization of artificial neural networks and ARIMA models for time series forecasting. *Applied Soft Computing Journal*, 11(2):2664–2675.

[39] Khashei, M., Bijari, M., and Raissi Ardali, G. A. (2012). Hybridization of autoregressive integrated moving average (ARIMA) with probabilistic neural networks (PNNs). *Computers and Industrial Engineering*, 63(1):37–45.

[40] Kim, H. J., Kim, M., Lim, H., Park, C., Yoon, S., Lee, D., Choi, H., Oh, G., Park, J., and Kim, Y. (2013). Fully Autonomous Vision-Based Net-Recovery Landing System for a Fixed-Wing UAV. *IEEE/ASME Transactions on Mechatronics*, 18(4):1320–1333.

[41] LaValle, S. M. (2006). *Planning Algorithms.* Cambridge University Press, Cambridge.

[42] Lekkas, A. M., Dahl, A. R., Breivik, M., and Fossen, T. I. (2013). Continuous-Curvature Path Generation using Fermat's Spiral. *Modeling, Identification and Control: A Norwegian Research Bulletin*, 34(4):183–198.

[43] Lekkas, A. M. and Fossen, T. I. (2014). Integral LOS Path Following for Curved Paths Based on a Monotone Cubic Hermite Spline Parametrization. pages 1–15.

[44] Leung, H., Lo, T., and Wang, S. (2001). Prediction of noisy chaotic time series using an optimal radial basis function neural network. *IEEE Transactions on Neural Networks*, 12(5):1163–1172.

[45] Li, G., Shi, J., and Zhou, J. (2011). Bayesian adaptive combination of short-term wind speed forecasts from neural network models. *Renewable Energy*,

36(1):352–359.

[46] Lopez, C. (1999). Looking inside the ANN "black box": classifying individual neurons as outlier detectors. *IJCNN'99. International Joint Conference on Neural Networks. Proceedings*, 2:1185 – 1188.

[47] LSTS (2013a). LSTS: About. http://lsts.fe.up.pt/about. Accessed: 14.11.2014.

[48] LSTS (2013b). LSTS: DUNE, DUNE: Unified Navigation Environment. http://lsts.fe.up.pt/software/dune. Accessed: 14.11.2014.

[49] LSTS (2013c). LSTS: IMC, Inter-Module Communication Protocol. http://lsts.fe.up.pt/software/imc. Accessed: 14.11.2014.

[50] LSTS (2013d). LSTS: Neptus, Neptus Command and Control Software. http://lsts.fe.up.pt/software/neptus. Accessed: 14.11.2014.

[51] Marinkovic, Z. and Markovic, V. (2010). Training Data Pre-Processing for Bias-Dependent Neural Models of Microwave Transistor Scattering Parameters. *Scientific Publication of the State University of Novi Pazar*, 2:21–28.

[52] Masi, G. D., Gaggiotti, F., Bruschi, R., and Venturi, M. (2011). Ship motion prediction by radial basis neural networks. *2011 IEEE Workshop On Hybrid Intelligent Models And Applications*, pages 28–32.

[53] MathWorks (2015). trainlm - Levenberg-Marquardt backpropagation. https://se.mathworks.com/help/nnet/ref/trainlm.html. Accessed: 26.04.2015.

[54] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

[55] Minsky, M. L. and Seymour, P. (1969). *Perceptrons; an Introduction to Computational Geometry*. MIT Press.

[56] Møller, M. F. (1990). A scaled conjugate gradient algorithm for fast supervised learning. Technical report, Computer Science Department, University of Aarhus, Aarhus, Denmark.

[57] Nicolau, V., Aiordachioaie, D., and Popa, R. (2004). Neural Network Prediction of the Wave Influence on the Yaw Motion of a Ship. *IEEE International Conference on Neural Networks - Conference Proceedings*, 4:2801–2806.

[58] Nicolau, V., Palade, V., Aiordachioaie, D., and Miholca, C. (2007). Neural Network Prediction of the Roll Motion of a Ship for Intelligent Course Control. In Apolloni, B., editor, *Knowledge-Based Intelligent Information and Engineering Systems*, pages 284–291, Vietri sul Mare, Italy. Springer-Verlag.

[59] Niculescu, M. (2001). Lateral Track Control Law for Aerosonde UAV. In *Proceedings of the 39th AIAA Aerospace Sciences Meeting and Exhibit*, Reston, Virigina. American Institute of Aeronautics and Astronautics.

[60] NMEA (2015). National Marine Electronics Association. http://www.nmea.org/. Accessed: 28.05.2015.

[61] Pandaboard.org (2014). Pandaboard ES Specifications. http://pandaboard.org/node/300/. Accessed: 11.12.2014.

[62] Park, S., Deyst, J., and How, J. P. (2004). A New Nonlinear Guidance Logic for Trajectory Tracking. *Proceedings of the AIAA Guidance Navigation and Control Conference*, (August):1–16.

[63] Perez, T., Smogelif, O. y. N., Fossen, T. I., and Sørensen, A. J. (2006). An Overview of the Marine Systems Simulator (MSS): A Simulink® Toolbox for Marine Control Systems. *Modeling, Identification and Control*, 27(4):259–275.

[64] Pinto, J., Dias, P. S., Martins, R., Fortuna, J. a., Marques, E., and Sousa, J. a. B. (2013). The LSTS Toolchain for Networked Vehicle Systems. In *2013 MTS/IEEE OCEANS - Bergen*, pages 1–9. IEEE.

[65] Rosenblatt, F. (1957). The perceptron: a perceiving and recognizing automaton. Technical report, Cornell Aeronautical Laboratory, Buffalo, NY.

[66] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. In Rumelhart, David E. and McClelland, James L. and PDP Research Group, C., editor, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, USA.

[67] Schouwenaars, T., Feron, E., and How, J. P. (2002). Safe Receding Horizon Path Planning for Autonomous Vehicles $*$. *Proceedings of the 40th Allerton Conference on Communication, Control, and Computation.*

[68] Schwenker, F., Kestler, H. a., and Palm, G. (2001). Three learning phases for radial-basis-function networks. *Neural Networks*, 14(4):439–458.

[69] Serway, R. A. and Jewett, J. W. (2003). *Physics for Scientists and Engineers.* Thomson-Brooks/Cole, 6th edition.

[70] Shin, D. H. and Singh, S. (1990). Path Generation for Robot Vehicles Using Composite Clothoid Segments. Technical report, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

[71] Sidar, M. M. and Doolin, B. F. (1983). On the feasibility of real-time prediction of aircraft carrier motion at sea. *IEEE Transactions on Automatic Control*, 28(3):350–356.

[72] Siouris, G. M. (2004). *Missile Guidance And Control Systems.*

[73] Skulstad, R. and Syversen, C. L. (2014). *Low-Cost Instrumentation System for Recovery of Fixed-Wing UAV in a Net.* Msc thesis, Norwegian University of Science and Technology.

[74] Smit, S. J. A. (2013). *Autonomous Landing of a Fixed-Wing Unmanned Aerial Vehicle sing Differential GPS.* Msc thesis, Stellenbosch University.

[75] Spockeli, B. A. (2015). *Integration of RTK GPS and IMU for Accurate UAV Positioning.* Msc thesis, Norwegian University of Science and Technology.

[76] Swift Navigation Inc. (2013). *Piksi Datasheet (2.3.1).* Product sheet.

[77] Takasu, T. (2013). *RTKLIB ver. 2.4.2 Manual (2.4.2).* Product manual.

[78] Tovée, M. J. (1994). Neuronal processing. How fast is the speed of thought? *Current biology : CB*, 4(12):1125–1127.

[79] Tridgell, A. (2014). MAVProxy. http://tridge.github.io/MAVProxy. Accessed: 25.11.2014.

[80] Tsourdos, A., White, B., and Shanmugavel, M. (2010). *Cooperative Path Planning of Unmanned Aerial Vehicles.* John Wiley & Sons, Ltd, Chichester, UK.

[81] Utkin, V. I. (1977). Variable Structure Systems with Sliding Modes. *IEEE Transactions on Automatic Control*, 22(2):212–222.

[82] Werbos, P. (1974). *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences.* Phd thesis, Harvard University.

[83] Xu, T., Liu, X., and Yang, X. (2011). Ship Trajectory Online Prediction Based on BP Neural Network Algorithm. *2011 International Conference of Information Technology, Computer Engineering and Management Sciences*, 1:103–106.

[84] Yang, X. and Zhao, X. (2005). Modeling of Ship Vertical Motion with self-organizing radial basis aotin dintellent mel Somedscholars hved. pages 1131–1136.

[85] Yanushevsky, R. (2011). *Guidance of Unmanned Aerial Vehicles.* CRC Press.

[86] Yin, J. C., Zou, Z. J., and Xu, F. (2013). On-line prediction of ship roll motion during maneuvering using sequential learning RBF neural networks. *Ocean Engineering*, 61:139–147.

[87] Yolcu, U., Egrioglu, E., and Aladag, C. H. (2013). A new linear & nonlinear artificial neural network model for time series forecasting. *Decision Support Systems*, 54(3):1340–1347.

[88] Yoon, S., Kim, H. J., and Kim, Y. (2010). Spiral Landing Guidance Law Design for Unmanned Aerial Vehicle Net-Recovery. *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, 224(10):1081–1096.

[89] Zapranis, A. and Livanis, E. (2005). Prediction Intervals for Neural Network Models. In *Proceedings of the 9th WSEAS International Conference on Computers*, pages 1–7. World Scientific and Engineering Academy and Society (WSEAS).

[90] Zhang, G. P. and Qi, M. (2005). Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514.