POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering



Master's Degree Thesis

Behavioural Modelling and Simulation of Multidomain Platforms with Coordinated Communication Strategies

Academic Supervisor

Prof. Alessandro Rizzo

Company Supervisor

Eng. Silvio Massimino

Candidate

Federica D'Ugo

October 2026

Abstract

Multidomain systems can improve the effectiveness of autonomous operations by enabling cooperation between platforms operating in different physical environments. In maritime applications, aerial vehicles can extend the observation range of surface platforms and provide information that supports the adaptation of a predefined route.

This study presents the behavioural modelling and simulation of a cooperative maritime–aerial scenario involving a Mariner class cargo ship and a multi-UAV fixed-wing system. The ship follows a predefined trajectory defined by a sequence of waypoints, while the UAVs operate ahead of it and manage predefined route safety conditions. When the UAV system enters a detection region associated with an unsafe waypoint, the aerial vehicles are redirected towards a candidate safe point. Once this location has been reached and verified, its coordinates are transmitted to the ship and used to update the maritime waypoint sequence.

The proposed framework is developed in the MATLAB®/Simulink® environment and integrates the fixed-wing multi-UAV model, the Mariner class cargo ship model, the bidirectional information-exchange architecture and the mission-management logic. Three different route geometries are considered to provide a functional verification of waypoint reassignment, ship rerouting and the simplified UAV return and capture procedure. The results confirm the correct interaction among the integrated subsystems and demonstrate the role of bidirectional information exchange in coordinated maritime–aerial mission execution.

Al mio fratellone Pino e alla mia sorellina Ale,
i miei più grandi sostenitori,
oggi come sempre.

Acknowledgements

I would like to express my sincere gratitude to my academic supervisor, Prof. Alessandro Rizzo, for his guidance, valuable feedback and support throughout the development of this thesis.

I would also like to thank my company supervisor, Eng. Silvio Massimino, for his technical advice, availability and assistance during the different stages of this work. His expertise and suggestions were particularly valuable in the development and integration of the multidomain simulation framework.

I am grateful to Brain Technologies for giving me the opportunity to carry out this thesis in a professional environment and to work on a challenging and multidisciplinary project. I would also like to thank all the colleagues who supported me and contributed, directly or indirectly, to this experience.

Finally, I would like to thank my family and the people closest to me for their constant encouragement, patience and support throughout my academic journey. Their presence has been fundamental in helping me reach this important milestone.

Contents

Acknowledgements **iv**

Acronyms and Symbols **xv**

1 Introduction **1**

1.1 Background and Motivation 1

1.2 Problem Description 2

1.3 Research Objectives and Contributions 3

1.4 Thesis Structure 4

2 State of the Art **6**

2.1 The Role of Information Exchange in Aerial-Maritime Cooperation . . 6

2.2 Networking and Interoperability in Cooperative Maritime Systems . . . 9

2.3 Fixed-wing vs. Multirotor UAVs 11

2.4 Positioning of the Present Work 14

3 Fixed-wing UAV Swarm Modelling **16**

3.1 Simulink Model of a Single Fixed-wing UAV 16

3.2 Transition from Single-instance to Multi-instance Architecture 20

3.3 Simulink Model of the Multi-UAV Fixed-wing System 22

3.3.1 Waypoint Generation and Multi-UAV Guidance 23

3.3.2 Heading Control Subsystem 25

3.3.3 Guidance Model 27

3.3.4 Environmental Subsystem 28
3.3.5 Signal Formatting and Coordinate Handling 28
3.3.6 3D Visualisation Subsystem 29
3.3.7 Marker Generation Subsystem 30
3.4 Overall Working Principle 32
3.5 UAV Swarm Model Validation 33

4 Maritime Simulation Framework **35**
4.1 Selection of the Ship Model 35
4.2 Simulink Model of the Mariner Cargo Ship 38
4.2.1 Path Definition and LOS Guidance 39
4.2.2 LOS Observer and Reference Smoothing 40
4.2.3 Course Autopilot and Rudder Command 41
4.2.4 Mariner Vessel Model 42
4.2.5 COG Reconstruction and EKF 44
4.2.6 Overall Interpretation 46
4.3 Validation of the Cargo Ship Model 47

5 Integrated Multidomain Model **51**
5.1 Integration of Message-Based Communication in the UAV Swarm Model 52
5.2 Integration of Message-Based Communication in the Mariner Cargo Ship Model 61
5.3 MultiDomain Integration Model 66
5.3.1 Uav to Ship Communication 69
5.3.2 Ship to UAV Communication 72

6 Simulation Results of the Integrated Multidomain Model **76**
6.1 Simulation Setup and Scenario Definition 77
6.2 Reference Scenario 79
6.2.1 Overall Mission Evolution 79

6.2.2 Route Adaptation 80
6.2.3 Return and Simplified Capture Phase 82
6.2.4 Outcome of the Reference Scenario 83
6.3 Alternative Route Geometry 84
6.4 Demanding Route Geometry 87
6.5 Comparative Discussion 90

7 Conclusions and Outlook **92**
7.1 Conclusions 92
7.2 Limitations 93
7.3 Outlook 94

A Source Code Listings **96**
A.1 Dynamic Waypoint Manager Function 96
A.2 Return and Dock Supervisor Function 100
A.3 Ship Waypoint Manager Function 101
A.4 Build Drone Messages Function 103

List of Figures

2.1 Cooperative aerial-maritime search scenario involving multiple UAVs and USVs [1]. 7

2.2 Physical architecture of a cooperative USV-UAV system [12]. 9

2.3 Basic network architecture for UAV aided maritime communication [5]. 10

2.4 Comparison between multirotor and fixed-wing UAV platforms. Cropped and adapted from HOVERAir Official, "Different Types of Drones and Uses", available online. 12

2.5 USV equipped with a battery hot-swap system and UAV landing pad [13]. 13

3.1 `fixedWingPathFollowing` Simulink Model. Source: MathWorks [18]. . 16

3.2 `UAV Waypoint Follower` block in the MATLAB/Simulink environment. 17

3.3 `Heading Controller` block in the MATLAB/Simulink environment. . . 18

3.4 `UAV Guidance Model` block in the MATLAB/Simulink environment. . . 18

3.5 `UAV Animation` block in the MATLAB/Simulink environment. 19

3.6 `Multi-Instance Guidance Model` block in the MATLAB/Simulink environment. 21

3.7 Simulink model `formationFlightSimulationblock` for the multi-UAV fixed-wing guidance and visualisation architecture. 22

3.8 `Multi-Instance Waypoint Follower` subsystem in the MATLAB/Simulink environment. 23

3.9 `Multi-Instance Heading Control` block in the MATLAB/Simulink environment. 25

3.10 `Multi-Instance Aircraft Guidance Model` subsystem in the MATLAB/Simulink environment. 27

3.11 `Windless` subsystem in the MATLAB/Simulink environment. 28

3.12 `UAV Animation` subsystem for multi-UAV 3D visualisation in the MATLAB/Simulink environment. 30

3.13 Semicircular turnaround trajectories of the five UAVs. Each coloured dotted line represents the trajectory followed by a different UAV: red for UAV 1, purple for UAV 2, yellow for UAV 3, light blue for UAV 4 and green for UAV 5. 33

3.14 Zigzag trajectories with altitude variation of the five UAVs. Each coloured dotted line represents the trajectory followed by a different UAV: red for UAV 1, purple for UAV 2, yellow for UAV 3, light blue for UAV 4 and green for UAV 5. 34

4.1 Simulink block of the Mariner class cargo ship model available in the MSS library [20]. 36

4.2 Simulink model `demoMarinerPathFollowingCourseControl` for path following and course control of the Mariner class cargo ship, available in the MSS library [20]. 38

4.3 `LOS guidance` and `LOS observer` blocks implemented in the MATLAB/Simulink environment. 39

4.4 `LOS observer` subsystem implemented in the MATLAB/Simulink environment. 40

4.5 `Course autopilot` block implemented in the MATLAB/Simulink environment. 42

4.6 Mariner ship model and kinematic reconstruction of the COG implemented in the MATLAB/Simulink environment. 43

4.7 `Crab angle` computation subsystem implemented in the MATLAB/Simulink environment. 44

4.8 `Discrete-time EKF for SOG and COG` block implemented in the MATLAB/Simulink environment [20]. 45

4.9 Reference waypoint sequence and corresponding ship trajectory in the North-East plane. The light-blue markers indicate the waypoints through which the ship is required to pass, while the continuous line represents the trajectory followed by the vessel. 47

4.10 True and estimated Speed Over Ground (SOG) during ship path following. The blue line represents the true SOG, while the red line represents the SOG estimated by the Extended Kalman Filter. 48

4.11 Commanded and actual rudder angles during waypoint transitions. The blue line represents the commanded rudder angle, while the red line represents the actual rudder angle of the ship. 49

4.12 Desired, actual and estimated COG during ship path-following manoeuvres. The blue line represents the desired COG, the red line represents the actual COG of the ship and the yellow line represents the COG estimated by the Extended Kalman Filter. 50

5.1 Modified UAV swarm architecture with communication and mission management functionalities. 52

5.2 Simulink block `DynamicWaypointManager`. 53

5.3 Simulink block `ReturnAndDockSupervisor`. 58

5.4 Modified Mariner cargo ship model used for message based communication with the UAV swarm. 62

5.5 `Multidomain` Simulink model. 67

5.6 Simulink subsystem `DroneToShip_comm`. 70

5.7 Simulink subsystem `ShipToDrone_comm`. 73

5.8 Simulink subsystem `drone_msg`. 74

6.1 Overall trajectory evolution obtained from the first integrated simulation in the MATLAB/Simulink environment. The red trajectory represents the motion of the five UAVs, whereas the blue trajectory represents the ship path. 80

6.2 First route adaptation obtained from the integrated MATLAB/Simulink simulation. The red trajectory represents the UAV motion, the blue trajectory represents the ship path, the green marker identifies the verified safe point and the purple marker denotes the predefined unsafe waypoint. 81

6.3 Second route adaptation obtained from the integrated MATLAB/Simulink simulation. The red trajectory represents the UAV motion, the blue trajectory represents the ship path, the green marker identifies the verified safe point and the purple marker denotes the predefined unsafe waypoint. 82

6.4 Final return and simplified capture phase obtained from the first integrated simulation in the MATLAB/Simulink environment. The red dotted trajectories represent the UAV return paths, whereas the blue trajectory represents the ship path. 83

6.5 Overall trajectory evolution obtained from the second integrated simulation in the MATLAB/Simulink environment. The red trajectory represents the motion of the five UAVs, whereas the blue trajectory represents the ship path. 85

6.6 First route adaptation obtained from the second integrated MATLAB/Simulink simulation. The green marker identifies `SAFE1`, whereas the purple marker denotes the corresponding predefined unsafe waypoint. 85

6.7 Second route adaptation obtained from the second integrated MATLAB/Simulink simulation. The green marker identifies `SAFE2`, whereas the purple marker denotes the corresponding predefined unsafe waypoint. 86

6.8 Return and simplified capture phase obtained from the second integrated simulation in the MATLAB/Simulink environment. The red dotted trajectories represent the UAV return paths, whereas the blue trajectory represents the ship path. 86

6.9 Overall trajectory evolution obtained from the third integrated simulation in the MATLAB/Simulink environment. The red trajectory represents the motion of the five UAVs, whereas the blue trajectory represents the ship path. 88

6.10 First route adaptation obtained from the third integrated MATLAB/Simulink simulation. The green marker identifies `SAFE1`, whereas the purple marker denotes the corresponding predefined unsafe waypoint. 88

6.11 Second route adaptation obtained from the third integrated MATLAB/Simulink simulation. The green marker identifies `SAFE2`, whereas the purple marker denotes the corresponding predefined unsafe waypoint. 89

6.12 Return and simplified capture phase obtained from the third integrated simulation in the MATLAB/Simulink environment. The red dotted trajectories represent the UAV return paths, whereas the blue trajectory represents the ship path. 89

List of Tables

6.1 Definition, purpose and observed outcome of the three simulation scenarios. 78

6.2 Qualitative comparison of the three integrated simulation scenarios. . . 90

List of Listings

3.1 Representative creation of one waypoint follower instance for each UAV, based on the MathWorks reference example [19]. 24
3.2 Representative multi-UAV iteration used by the waypoint follower object. 24
3.3 Representative initialization of the marker input matrix. 31
3.4 Representative assignment of waypoint marker positions. 31
3.5 Representative assignment of look-ahead point markers. 31
5.1 Persistent variables used to store the mission state. 54
5.2 Example of transition from nominal navigation to the first rerouting phase. 56
5.3 Example of safe point verification. 56
5.4 Selection of the active waypoint sequence. 56
5.5 Example of safe point communication logic. 57
5.6 Persistent variable used to store the return mode of each UAV. 58
5.7 Transition from nominal mission to return phase. 59
5.8 Representative computation of the return course towards the ship. . . . 59
5.9 Transition from return phase to capture condition. 60
5.10 Representative assignment of the UAV position during capture. 60
5.11 Persistent variables used to store the ship rerouting state. 63
5.12 Activation of the first ship rerouting stage. 64
5.13 Activation of the second ship rerouting stage. 64
5.14 Update of the ship waypoint sequence using the verified safe points. . . 64
5.15 Initialization of the drone message matrix. 71
5.16 Representative assignment of the message fields for each UAV. 71
A.1 MATLAB Implementation of the `DynamicWaypointManager` function. . 96
A.2 MATLAB implementation of the `ReturnAndDockSupervisor` function. 100
A.3 MATLAB implementation of the `ShipWaypointManager` function. . . . 101
A.4 MATLAB implementation of the `BuildDroneMessages` function. 103

Acronyms and Symbols

Acronyms

SAR	Search and Rescue
UAV	Unmanned Aerial Vehicle
USV	Unmanned Surface Vehicle
UGV	Unmanned Ground Vehicle
UMV	Unmanned Marine Vehicle
UUV	Unmanned Underwater Vehicle
MASS	Maritime Autonomous Surface Ship
CNN	Convolutional Neural Network
VHF	Very High Frequency
LOS	Line of Sight
RL	Reinforcement Learning
3D	Three-Dimensional
RGB	Red, Green and Blue
R2D	Radians to Degrees
MSS	Marine Systems Simulator
EKF	Extended Kalman Filter
SOG	Speed Over Ground
COG	Course Over Ground
GNSS	Global Navigation Satellite System
DOF	Degrees of Freedom
GNC	Guidance, Navigation and Control
WP	Waypoint
ZYX	Euler rotation sequence about the (z), (y), and (x) axes

Symbols

θ	Pitch angle
γ	Flight path angle
α	Angle of attack
$V_{a,i}$	Airspeed of the i-th UAV
$V_{g,i}$	Ground speed of the i-th UAV
ψ_i	Heading angle of the i-th UAV
$\psi_{c,i}$	Commanded heading angle of the i-th UAV
$\psi_{\text{yaw},i}$	Wind corrected yaw angle of the i-th UAV
$\phi_{c,i}$	Commanded roll angle of the i-th UAV
$e_{\psi,i}$	Heading tracking error of the i-th UAV
P_ψ	Heading control gain
g_i	Gravity term associated with the i-th UAV
$\mathbf{W}_i$	Wind vector associated with the i-th UAV
$W_{N,i}$	North wind component of the i-th UAV
$W_{E,i}$	East wind component of the i-th UAV
$W_{D,i}$	Down wind component of the i-th UAV
b_i	First intermediate coefficient used in the ground speed computation
c_i	Second intermediate coefficient used in the ground speed computation
Δ	Look-ahead distance used in the LOS guidance law
R_{switch}	Switching radius used for waypoint transition
χ_{ref}	Reference course generated by the LOS guidance block
χ	Course over ground of the vessel
β_c	Crab angle of the vessel
ψ	Heading angle of the vessel
u	Surge velocity of the vessel
v	Sway velocity of the vessel
r	Yaw rate of the vessel
x_{pos}	North position of the vessel
y_{pos}	East position of the vessel
δ_r	Rudder angle
U_0	Nominal forward speed of the vessel
x_0	Initial state vector of the vessel
K_f	Observer gain used in the LOS observer

χ_d	Desired course angle provided to the course autopilot
$\omega_{\chi,d}$	Desired course rate provided to the course autopilot
ω_χ	Course over ground rate estimated by the EKF
δ_c	Commanded rudder angle
$R_{\text{detect},1}$	Detection radius associated with the first unsafe waypoint
$R_{\text{detect},2}$	Detection radius associated with the second unsafe waypoint
$R_{\text{safe},1}$	Verification radius associated with the first safe point
$R_{\text{safe},2}$	Verification radius associated with the second safe point
R_{end}	Mission-completion radius used to activate the UAV return phase
R_{capture}	Capture radius used to identify the final UAV recovery condition
$d_{\text{end},i}$	Distance between the i-th UAV and the final mission waypoint
$d_{\text{ship},i}$	Distance between the i-th UAV and the ship during the return phase
ΔN_i	North position error between the ship and the (i)-th UAV during the return phase
ΔE_i	East position error between the ship and the (i)-th UAV during the return phase
N_{ship}	North position of the ship
E_{ship}	East position of the ship
$N_{\text{UAV},i}$	North position of the (i)-th UAV
$E_{\text{UAV},i}$	East position of the (i)-th UAV
$\chi_{\text{return},i}$	Return course reference of the (i)-th UAV towards the ship

Chapter 1

Introduction

1.1 Background and Motivation

Modern maritime operations increasingly rely on heterogeneous unmanned platforms operating across multiple domains to ensure situational awareness, safe navigation and mission effectiveness in complex and dynamic environments. The growth of Maritime Autonomous Surface Ships (MASS) has intensified the need for systems capable of monitoring the surrounding environment, anticipating hazards along a planned route and adapting navigation decisions without continuous human supervision. In this context, the integration of autonomous cargo ships with aerial drone swarms represents a promising approach to extend the perception range of the maritime platform, support cooperative decision making and increase the reliability of autonomous navigation in offshore scenarios.

Cargo ships are particularly relevant for this application because of their size, limited manoeuvrability and long planned trajectories. Unlike smaller vessels, a large cargo ship cannot react quickly to obstacles or unsafe conditions discovered at close range. It is therefore advantageous to inspect the planned route in advance, before the ship reaches a given segment, so that corrective actions can be planned with sufficient time margins. This forward-looking inspection task is well suited to aerial platforms, which can cover the route ahead of the ship at higher speed and with a wider field of view than any sensor mounted directly on the vessel.

Among the available aerial platforms, fixed-wing UAVs offer a particularly favourable tradeoff for this type of mission. Compared to multirotor configurations, fixed-wing aircraft provide significantly longer endurance, higher cruise speeds and better aerodynamic efficiency, at the cost of requiring forward motion and being unable to hover.

These characteristics are well aligned with the requirements of route monitoring along extended maritime trajectories, where the aerial agents must cover large segments repeatedly and maintain a speed advantage over the ship. A swarm of such vehicles can divide the inspection task among multiple agents, increasing coverage and robustness against individual failures.

Traditionally, maritime connectivity has relied on land based cellular networks, Very High Frequency (VHF) radios and satellite links. Although these technologies are widely used, they present several limitations in offshore areas, including restricted coverage, high latency and low data rates. These limitations become critical when autonomous platforms need to exchange navigation data, coordinate their actions and react to environmental changes in near real time. In the proposed simulation scenario, communication between the ship and the swarm is not intended to provide broadband access, but to support a local information loop in which the aerial agents report the safety status of the route ahead and the ship responds with updated position and trajectory data.

Designing behavioural models and communication strategies for such heterogeneous systems remains a challenging task. The ship and the drones operate under fundamentally different motion constraints, have different energy and communication limitations and must synchronise their actions based on position and safety information exchanged through a link that is subject to range constraints and variable channel conditions. For this reason, simulation plays a key role in analysing the behaviour of such systems before real deployment.

The interaction between the cargo ship and the fixed-wing UAV swarm is modelled and simulated within the MATLAB/Simulink environment. Established dynamic models are used for both platforms, while a custom communication and coordination logic enables the UAV swarm to assess route safety and trigger trajectory adaptations when required. The communication architecture also allows the ship to provide its position to the UAVs, so that the swarm can update its return direction and execute the final return and docking phase.

1.2 Problem Description

The present study addresses the behavioural modelling and simulation of a multidomain scenario involving an autonomous cargo ship and a swarm of fixed-wing UAVs. The two platforms operate in different physical domains and exchange navigation information through a local communication link, with the aim of improving route safety along a

predefined maritime trajectory.

The ship is assumed to follow a planned route, while the swarm is deployed to inspect the same trajectory in advance. The drones fly segments of the route ahead of the ship, assess whether conditions are safe and transmit the results back to the maritime platform. If a route segment is identified as unsafe, the UAV swarm does not simply report the hazard to the ship, but also verifies an alternative safe location that can be used to modify the maritime trajectory. Once a safe segment has been validated by the drones, the ship can adapt its path in order to avoid the affected area and continue the mission through the verified safe route. During the return phase, the drones rely on updated ship position data to align their approach with the moving vessel and perform recovery.

The problem is relevant because multidomain maritime systems require continuous interaction among platforms with different dynamics, operational constraints and communication capabilities. A cargo ship typically follows slower and more constrained trajectories, while fixed-wing drones can move more flexibly, cover larger areas and inspect specific segments of the route before the ship reaches them. At the same time, aerial platforms are limited by battery capacity, communication range and environmental conditions. These differences make coordination complex and require suitable models capable of representing both the individual behaviour of each platform and the information exchange among them.

The work is based on the selection and integration of dynamic models for the ship and the swarm, chosen from established libraries and adapted to the requirements of the considered scenario. These models are integrated into a Simulink simulation framework that represents coordinated navigation, route monitoring and information exchange as separate but interacting subsystems.

A communication strategy is defined to allow the platforms to exchange position and route safety updates, enabling trajectory adaptation when unsafe conditions are detected along the planned path.

1.3 Research Objectives and Contributions

The work does not aim to develop new high-fidelity aerodynamic or hydrodynamic models. Instead, established models available in the MATLAB/Simulink environment and in the Marine Systems Simulator library are selected, adapted and integrated into a common architecture.

The main contributions of the work can be summarised as follows:

- Development of an integrated MATLAB/Simulink simulation architecture combining a multi-instance fixed-wing UAV model with the Mariner class cargo ship model. The two subsystems retain their respective guidance and navigation structures while operating within a shared multidomain scenario.
- Development of the `DynamicWaypointManager` logic for the UAV system. This component supervises the transition between nominal navigation, unsafe-waypoint detection, safe-point verification and resumption of the original mission, while dynamically selecting the waypoint sequence assigned to the UAVs.
- Definition of a bidirectional information exchange architecture between the aerial and maritime domains. UAV-to-ship messages communicate the mission state, UAV position and verified safe-point coordinates, whereas ship-to-UAV messages provide updated vessel position information required during the return phase.
- Development of the `ShipWaypointManager` logic for the replacement of unsafe ship waypoints with safe locations verified by the UAV system. This enables the maritime platform to modify its predefined route according to the information received from the aerial agents.
- Development of the `ReturnAndDockSupervisor` logic for the coordinated return of the fixed-wing UAVs towards the moving ship. The supervisor computes the return course references using the updated vessel position and represents the final recovery phase through a simplified capture radius condition. The complete integrated architecture is then evaluated through three simulation scenarios characterised by different waypoint and route geometries.

1.4 Thesis Structure

The remainder of the document is organised as follows:

- **Chapter 2 – State of the Art** reviews the relevant literature on cooperative aerial-maritime systems, with particular attention to information exchange, communication architectures, interoperability issues, UAV platform selection and autonomous maritime navigation. It also identifies the research context and clarifies the positioning of the present work.

- **Chapter 3 – Fixed-wing UAV Swarm Modelling** presents the modelling process adopted for the fixed-wing UAV swarm. Starting from a reference single UAV path-following model, the chapter describes the transition to a multi-instance architecture and details the main subsystems used for waypoint following, heading control, guidance, environmental modelling, visualisation and validation.

- **Chapter 4 – Maritime Simulation Framework** describes the selection of the maritime model used. In particular, it focuses on the Mariner class cargo ship model, its Simulink architecture, the path-following logic and the estimation blocks used to reconstruct navigation quantities.

- **Chapter 5 – Integrated Multidomain Model** explains how the UAV swarm model and the cargo ship model are integrated into a common simulation framework. The chapter presents the communication based modifications introduced in both models and describes the information exchange logic between the aerial and maritime platforms.

- **Chapter 6 – Simulation Results of the Integrated Multidomain Model** presents the complete simulation scenario and discusses the results obtained from the integrated multidomain model. The chapter shows how the system behaves during coordinated aerial-maritime operations and route-safety evaluation.

- **Chapter 7–Conclusions and Outlook** summarizes the main results obtained, discusses the limitations of the current implementation and proposes possible future developments for extending and improving the proposed simulation framework.

Chapter 2

State of the Art

2.1 The Role of Information Exchange in Aerial-Maritime Cooperation

Recent studies on unmanned maritime systems no longer consider single vehicles acting independently, but rather coordinated groups of heterogeneous platforms that share information and cooperate during mission execution [1, 2, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15, 17]. In this context, unmanned marine vehicles (UMVs) have gradually evolved from remotely operated tools limited to inspection or damage assessment into autonomous agents capable of executing complex tasks with minimal human supervision [11]. This evolution has been enabled by improvements in onboard sensing, artificial intelligence and communication hardware, which naturally leads to architectures in which fleets of unmanned surface vehicles (USVs) and unmanned aerial vehicles (UAVs) are designed to work together instead of operating in isolation [12].

Maritime Search and Rescue (SAR) provides one of the clearest examples of these multidomain configurations. When operations involve only surface assets, the field of view remains constrained to sea level and the detection of distant objects or hazardous conditions can be slow and unreliable [4].

Introducing an aerial component changes this picture: UAVs can act as elevated sensing nodes, extending the perception range of the overall system and providing additional information that would not be available from the surface alone. As shown in Figure 2.1, in the cooperative framework proposed by Dong et al., the UAV swarm first scans the search area from above and generates probability maps indicating the regions where the target is more likely to be located. These maps are then transmitted to the USVs, which

use them to update their trajectories, focusing around the most promising regions. As the mission progresses, the information collected by the surface vehicles contributes to refining the search process, thereby closing the loop between aerial sensing and maritime navigation [1].

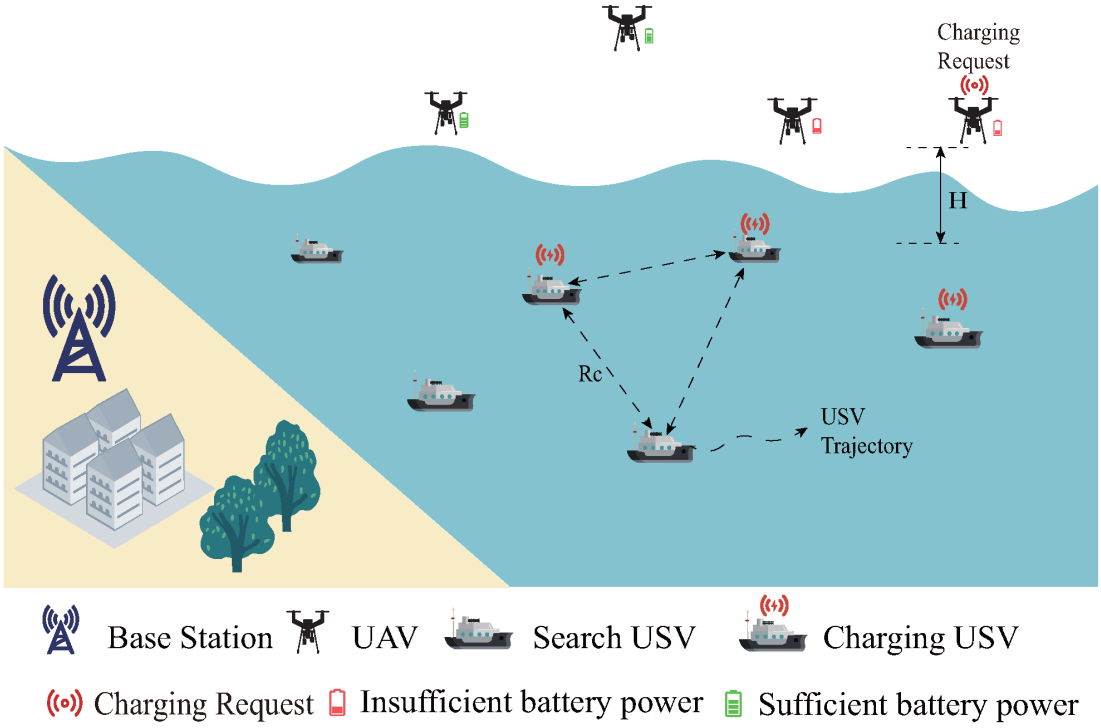


Figure 2.1: Cooperative aerial-maritime search scenario involving multiple UAVs and USVs [1].

Other contributions use convolutional neural networks (CNNs) to process images acquired from above, extracting quantities such as the location of floating objects or the pose and heading of the USV, that are subsequently passed to reinforcement learning (RL) controllers responsible for steering the surface vehicle in the presence of waves and disturbances [2].

Although the perception and control architectures differ across these works, they share a common objective: exploiting the complementary capabilities of aerial and surface platforms to improve mission awareness and decision making. The air segment gathers environmental data from above, while the maritime segment uses this information to adapt its behaviour and execute the mission [2, 1]. This coordination is particularly important in uncertain and dynamic conditions such as moving targets, changing sea states or unexpected obstacles.

The effectiveness of these cooperative configurations also depends on the mission-

planning strategy adopted by the aerial swarm. Checker et al. [3] provide a systematic review of multi-objective UAV swarm mission-planning systems, highlighting how the use of multiple aerial agents makes it possible to address concurrent objectives such as task assignment, path planning, coverage, energy management and operational safety. These capabilities are particularly relevant in complex missions, where the swarm must distribute tasks among its members while adapting its behaviour to changing environmental and operational conditions. At the same time, the review identifies several aspects that remain insufficiently addressed in existing systems, including real time mission observability, communication-link loss mitigation and the management of abnormal operating conditions.

Within an aerial-maritime scenario, these considerations suggest that information exchange should not be treated as an isolated communication function, but as an enabling element of mission coordination, allowing the UAVs to report route conditions, update their assigned trajectories and initiate recovery procedures when the inspection task has been completed.

A practical way to support these functions is to assign the surface vehicle a central supervisory role within the heterogeneous system. In this configuration, it can act as a coordination hub or "*mothership*", hosting navigation systems and communication hardware for the management of one or more UAVs [12, 13].

As represented in Figure 2.2, this concept can be implemented through an unmanned surface platform equipped with onboard sensing modules, communication antennas, control interfaces and a shipborne drone bay for UAV storage, launch, recovery and power replacement. In this role the USV does not only operate as a maritime vehicle, but also provides physical and informational support to the aerial segment by enabling data exchange, mission coordination and extended UAV operation. This support is particularly relevant because commercial multirotor UAVs are often constrained by short flight time and limited range, making continuous coverage difficult without an appropriate support strategy.

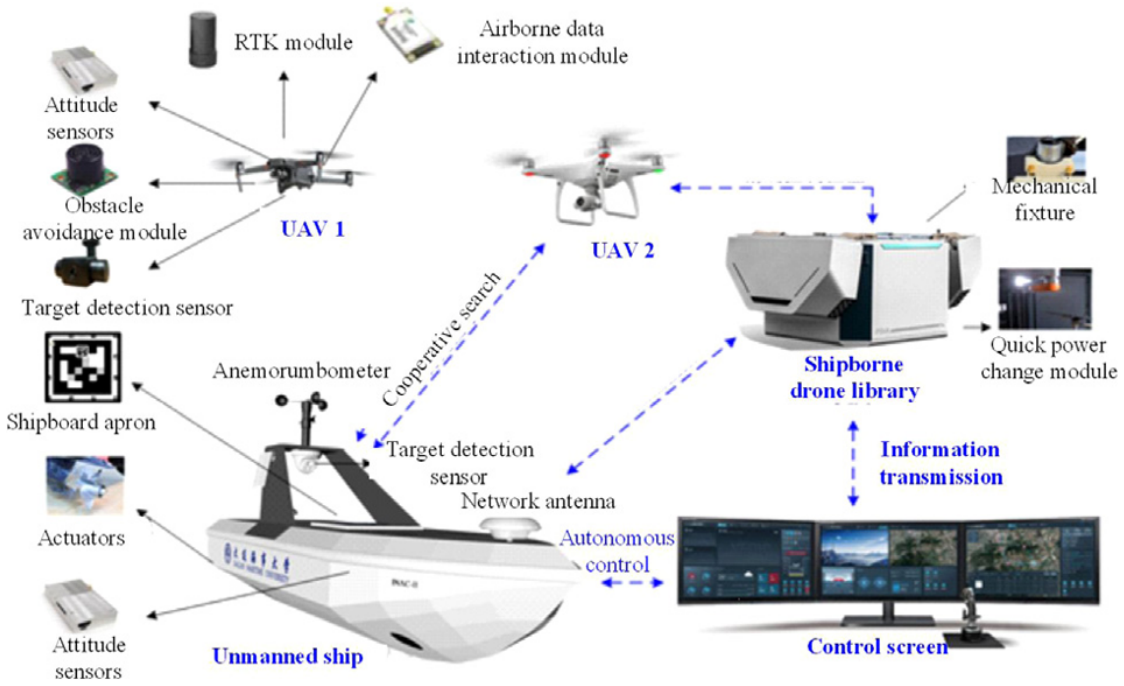


Figure 2.2: Physical architecture of a cooperative USV-UAV system [12].

2.2 Networking and Interoperability in Cooperative Maritime Systems

When aerial and surface vehicles share information over long distances, the quality of the communication infrastructure becomes a critical factor. Conventional maritime links based on VHF or satellite connections are known to suffer from high latency, limited throughput and significant operating costs [5]. To mitigate these limitations, several authors propose using UAVs as intermediate communication nodes or aerial relays, effectively extending network coverage into areas where traditional infrastructure is weak or absent [5].

As illustrated in Figure 2.3, the UAV operates as an aerial relay between the maritime control station and the surface vessels, forwarding information and extending the effective communication range of the network link [5]. In these works, particular attention is given to the modelling of air-to-sea channels, which differ from terrestrial wireless channels due to line of sight (LOS) dominance, sea surface reflections and the influence of Earth curvature. Two-ray and curved-Earth models are commonly adopted to capture these effects in a realistic way [5].

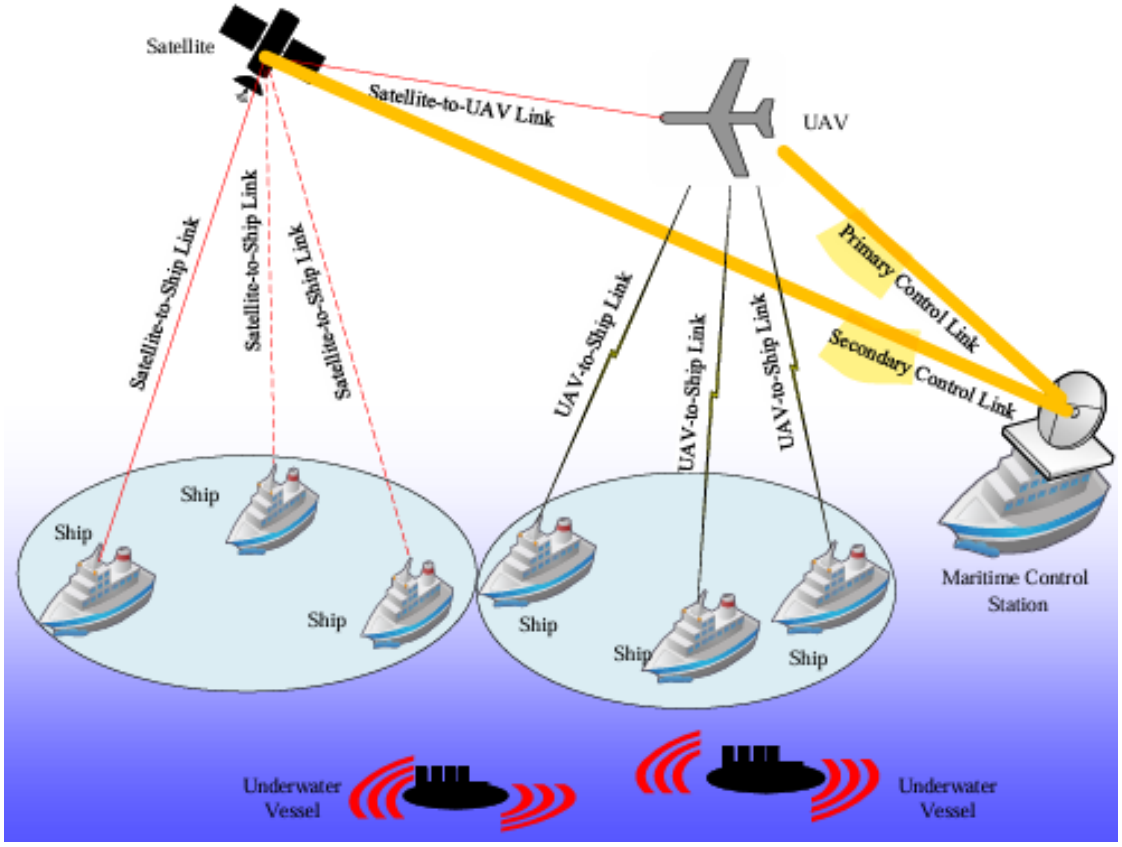


Figure 2.3: Basic network architecture for UAV aided maritime communication [5].

The characteristics of maritime wireless propagation are also strongly influenced by the condition of the sea surface. Li et al. [6] investigated a ship-to-ship wireless link through measurements conducted in the Pearl River Estuary and analysed the effects of different wind and sea surface conditions using a Round Earth Loss model. Their results show that maritime propagation cannot always be represented by a constant free-space link, since sea surface reflection, Earth curvature, wave-induced roughness and shadow fading can significantly modify the received signal characteristics. Although the study focuses on ship-to-ship communication, the identified propagation mechanisms are also relevant when defining more realistic communication models for cooperative aerial-maritime systems.

Beyond single relays, more recent contributions explore the use of multiple UAVs as a distributed communication layer. In this setting, several aerial nodes can share the burden of relaying information between shore stations and maritime users. One example is the use of UAV swarms, where some vehicles serve multiple users simultaneously through advanced multiple access schemes, while others listen to the ongoing transmissions and use packet identifiers to avoid unnecessary repetitions [10].

Treating the swarm as a flexible connectivity layer has two main consequences:

- Relay nodes can be selected dynamically depending on channel conditions;
- The system becomes more tolerant to individual failures, since the loss of a single UAV does not immediately compromise coverage.

However, even with sophisticated communication strategies, heterogeneous teams of unmanned platforms cannot operate effectively without a suitable software and architectural framework.

The concept of "*interoperability*", as discussed in recent works on unmanned maritime systems, goes beyond the definition of communication standards [14]. It also involves the way vehicles, control stations and software components are organised into modules and connected through stable interfaces. Service-oriented architectures are often proposed as a solution in which high level functionalities are implemented as separate services that can be mapped to different platforms [15].

European initiatives on interoperability highlight that such modularity is essential to integrate new sensors, vehicles or controllers without redesigning the entire system [15]. By decoupling the components and specifying clearly which information is exchanged between them, it becomes possible to scale from single vehicle scenarios to more complex multidomain operations. This philosophy is particularly relevant for the present work, where the cargo ship, the UAV swarm and the communication channel are treated as distinct elements that must interact in a coordinated and reliable way.

Taken together, these studies provide the conceptual foundation for the scenario considered here, while addressing research objectives that differ from the specific focus of the present work. Most of the existing literature aims at improving SAR search patterns, enabling low level visual navigation for single USVs or maximising communication performance for maritime users [2, 1, 4, 7, 10].

In contrast, the emphasis of this thesis is on route safety for a given cargo ship trajectory and on the use of aerial information as an additional decision layer for that trajectory. The communication links are not designed to provide general broadband access, but to support a local loop between the ship and its onboard UAV swarm.

2.3 Fixed-wing vs. Multirotor UAVs

Different aerial platforms offer different tradeoffs between manoeuvrability, endurance and operational complexity. These tradeoffs have a direct impact on the design of

cooperative maritime systems. Architectures that operate across space, air, ground and sea domains, such as those surveyed in [8], underline the distinction between fixed-wing aircraft and multirotor UAVs.

As shown in Figure 2.4, the two configurations differ significantly in terms of structure and operational capabilities. Multirotors can hover and hold a position with high accuracy, which is advantageous for tasks like station keeping or vertical inspection, but they usually exhibit limited flight time and a restricted effective range, often around 10 kilometres. Fixed-wing platforms, on the other hand, cannot hover and must maintain forward motion, yet they provide higher cruise speeds, better aerodynamic efficiency and significantly longer endurance [8].

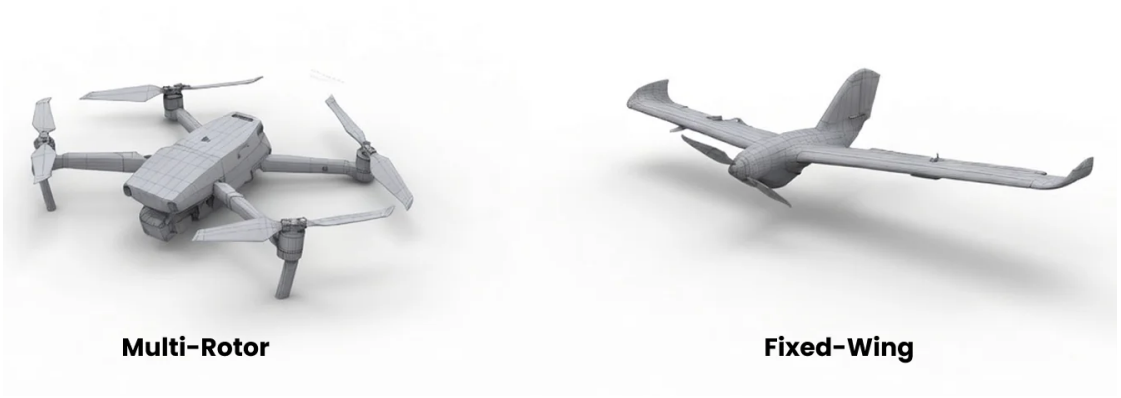


Figure 2.4: Comparison between multirotor and fixed-wing UAV platforms. Cropped and adapted from HOVERAir Official, "Different Types of Drones and Uses", available online.

Within the scope of the present work, the main objective of the aerial component is to inspect a portion of the route that lies ahead of a moving cargo ship. The UAVs are required to cover long segments and to revisit them within a reasonable time, so that potential hazards can be detected before the ship reaches them. This type of mission is well suited to a waypoint based coordination approach, in which each agent of the swarm follows a predefined sequence of positions derived from the planned ship route. Such strategies have been shown to enable efficient area coverage and distributed task execution in heterogeneous multirobot systems, including configurations where aerial and surface agents cooperate in parallel [9].

Under these conditions, the advantages of fixed-wing aircraft become particularly relevant. A swarm of fixed-wing UAVs can sweep a larger area in a shorter time and maintain higher average speeds, making it more suitable for forward scouting and route safety assessment than an equivalent multirotor configuration.

Selecting fixed-wing platforms does not remove the need for a suitable energy management strategy. Continuous operation still requires a surface platform that can act as a base for launch, recovery and data exchange. In some implemented systems, this role is played by a USV equipped with docking mechanisms and battery replacement hardware, effectively functioning as a mothership for a multirotor UAV [13].

As shown in Figure 2.5, the maritime platform can support UAV launch, recovery and battery swapping, enabling persistent aerial operations. Although the referenced implementations rely on vertical landing procedures and target different UAV types, they highlight the advantage of using a surface vehicle as a mobile base. The proposed study adopts the same principle by treating the cargo ship as a central node from which the UAV swarm can operate and periodically return for recovery.



Figure 2.5: USV equipped with a battery hot-swap system and UAV landing pad [13].

Operating fixed-wing UAVs from a moving maritime platform, however, introduces additional constraints. Since these aircraft cannot descend vertically, they must adjust their trajectories to align with the ship motion during the return phase. This requires a reliable exchange of telemetry between the swarm and the vessel, including updated ship positions and possibly relative orientation information. The UAVs use this data to compute their approach paths and to coordinate their arrival with the ship trajectory. These communication and guidance aspects are explicitly represented in the simulation, so that the interaction between the fixed-wing swarm and the cargo ship can be analysed under realistic conditions.

2.4 Positioning of the Present Work

The scenario investigated in this work builds upon the ideas discussed above, but introduces a specific operational focus. Rather than concentrating on generic SAR missions, low level control of a single USV or broadband service provision, the work targets the safety of a predefined route for an autonomous cargo ship. Maritime Autonomous Surface Ships increasingly rely on automated decision making, which requires systematic methods to identify hazards, estimate their likelihood and assess the potential consequences for navigation [17]. In this context, route planning is closely linked to risk assessment, since the chosen trajectory should avoid areas or situations where the probability of collision, loss of control or system failure becomes unacceptable [17].

A swarm of fixed-wing UAVs is employed as an additional sensing layer that follows a path aligned with the ship trajectory but shifted in time, so that the aerial agents can inspect the route before the vessel reaches it. Segments of the path that appear unsafe from the UAV perspective can then be flagged and used to support a possible adaptation of the ship trajectory. In this way, the safety oriented perspective adopted in MASS risk assessment is extended to a multidomain scenario where aerial information contributes directly to route evaluation.

From a modelling point of view the cargo ship, the UAV swarm and the communication logic are represented in Simulink as separate subsystems that exchange signals through clearly specified interfaces [14, 15]. The fixed-wing swarm is not intended to explore the environment freely, but to execute waypoint based missions derived from the planned ship route, acting as an auxiliary sensing layer [10]. By exploiting the higher speed and extended coverage of fixed-wing vehicles [8], the UAVs can scan the path ahead and provide early indications of potential hazards.

Whenever the swarm detects an unsafe area, the corresponding information is transmitted to the cargo ship through the local communication link. The ship can then reconsider the planned path and, if necessary, select an alternative trajectory that avoids the unsafe segment. During the return phase, the UAVs rely on updated ship coordinates to compute their approach and synchronise with the moving vessel, in line with the mothership concepts discussed in [13].

The overall simulation framework therefore enables an initial behavioural validation of a multidomain system in which a cargo ship and a fixed-wing swarm cooperate by sharing navigation and environment related data, closing the gap between multi UAV coordination and the embedded control logic required for autonomous maritime operations.

By combining behavioural modelling with route-safety assessment, the present work extends beyond previous studies that primarily address communication architectures, perception algorithms or isolated platform-level functions. In particular, it considers not only how the heterogeneous vehicles exchange information and react to detected hazards, but also how these interactions influence mission execution, waypoint reassignment and the adaptation of the ship trajectory within an integrated maritime–aerial scenario.

Chapter 3

Fixed-wing UAV Swarm Modelling

3.1 Simulink Model of a Single Fixed-wing UAV

Before introducing the proposed multi-UAV architecture, it is useful to consider a simpler reference model based on a single fixed-wing UAV. A suitable baseline is the Simulink® model `fixedWingPathFollowing`, presented in the MathWorks example `Tuning Waypoint Following Controller for Fixed-Wing UAV` [18]. This model combines a Waypoint Follower block with a fixed-wing Guidance Model block and shows how trajectory references can be transformed into control actions for a single aircraft, as shown in Figure 3.1.

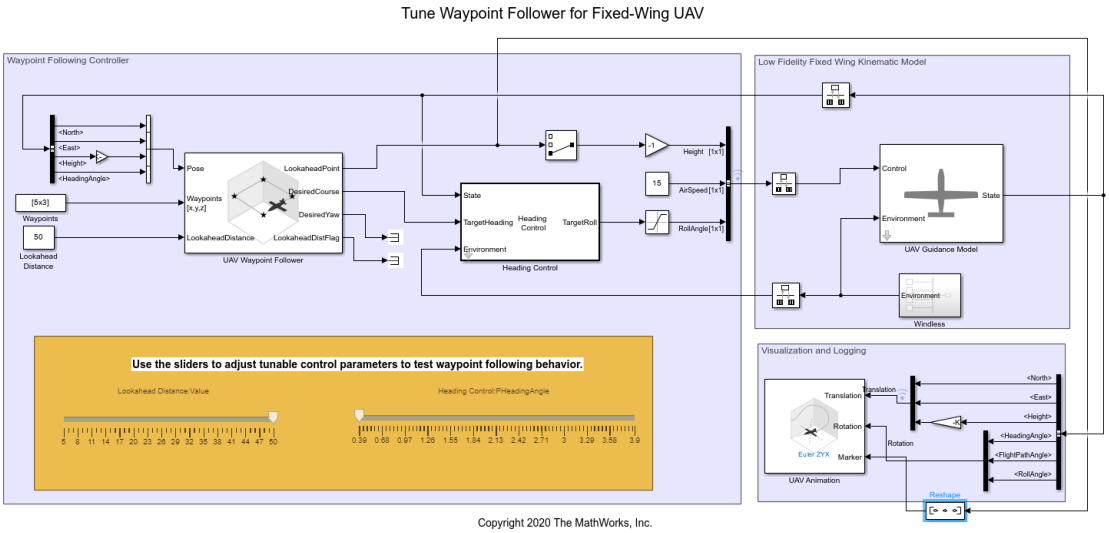


Figure 3.1: `fixedWingPathFollowing` Simulink Model. Source: MathWorks [18].

The purpose of this reference model is to describe the essential guidance loop of a fixed-

wing UAV through a simplified but effective representation. The fixed-wing guidance block does not reproduce a high fidelity aerodynamic model, instead it approximates the behaviour of an aircraft controlled by an autopilot under stable flight conditions. This level of abstraction is suitable for preliminary simulations aimed at evaluating the waypoint-following logic and the overall trajectory tracking behaviour.

As shown in Figure 3.1, the yellow panel in the lower part of the model contains interactive sliders for tuning selected parameters, such as the look-ahead distance and the heading control gain. Since the present work focuses on extending the guidance loop to multiple UAVs, this interactive tuning interface is not used in the subsequent modelling.

From a structural point of view, the model is organised around three main components:

- the `UAV Waypoint Follower` block, shown in Figure 3.2, uses the current UAV pose, the assigned waypoint sequence and the selected look-ahead distance to compute a look-ahead point along the path.

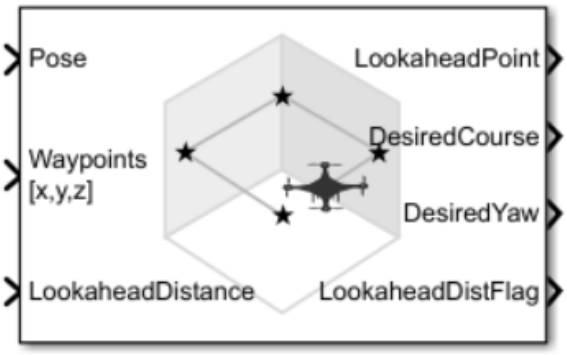


Figure 3.2: `UAV Waypoint Follower` block in the MATLAB/Simulink environment.

  From this information, the block provides the reference course and yaw commands required by the following control stage.

- the `Heading Controller`, shown in Figure 3.3, is implemented through a proportional control action. It receives the current UAV state, the target heading, the environmental information and computes the corresponding roll angle command required to reduce the heading tracking error.

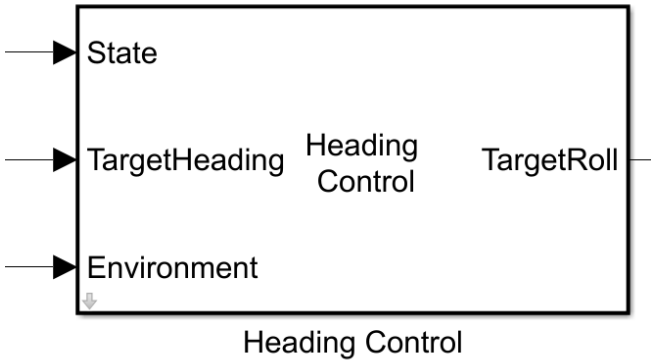


Figure 3.3: `Heading Controller` block in the MATLAB/Simulink environment.

- the `UAV Guidance Model` block, shown in Figure 3.4, represents the simplified aircraft dynamics and updates the UAV state as a function of the control commands and external inputs.

  The block receives the roll command and the other guidance-related quantities, such as the flight path angle, airspeed, wind components and propagates the UAV state during the simulation.

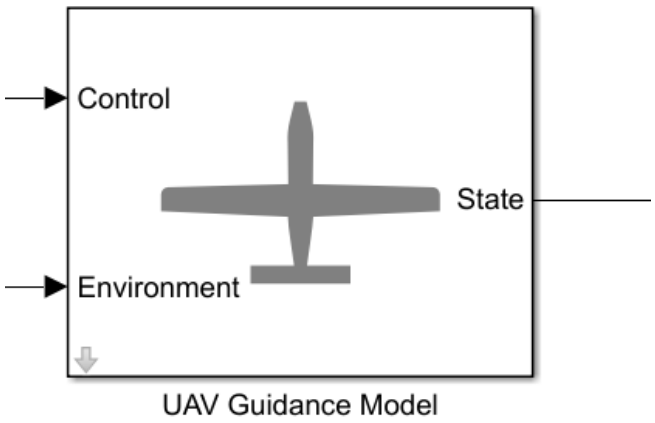


Figure 3.4: `UAV Guidance Model` block in the MATLAB/Simulink environment.

More specifically, the waypoint-following logic does not force the vehicle to aim directly at the next waypoint. Instead, it computes a look-ahead point and a desired heading, so that the aircraft follows a smoother path and reaches each waypoint within a specified transition radius. This anticipative behaviour is important in fixed-wing applications, because such vehicles cannot perform the abrupt directional changes that are more feasible for other aerial platforms, such as multirotor drones.

The heading controller compares the current heading with the desired heading and produces the roll angle command required to steer the UAV towards the target path. As a result, the guidance architecture forms a closed loop: the aircraft state is continuously updated by the guidance model and fed back to the waypoint-following and heading-control stages.

Another important element of this model is the `UAV Animation` block, shown in Figure 3.5, which displays the vehicle trajectory, heading and look-ahead point during the simulation.

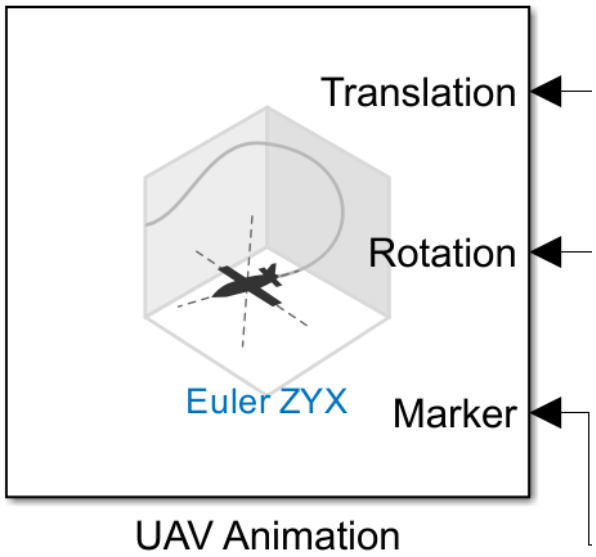


Figure 3.5: `UAV Animation` block in the MATLAB/Simulink environment.

In this block, the aircraft attitude is represented through a simplified relation between the pitch angle θ, the flight path angle γ and the angle of attack α:

$$\theta = \gamma + \alpha$$

Since the angle of attack is assumed to be negligible in the considered operating conditions, it is set to zero:

$$\alpha = 0$$

Therefore, the pitch angle is approximated as

$$\theta = \gamma$$

This simplification is used only to represent the UAV attitude in the 3D animation and does not correspond to a high fidelity aerodynamic model.

Overall, the `fixedWingPathFollowing` example represents an appropriate baseline for this work, since it includes the main elements that are later extended in the proposed architecture: waypoint based navigation, heading regulation, environmental inputs, state propagation and 3D visualisation [18].

For this reason, it provides a clear conceptual starting point for extending the analysis from the guidance of a single UAV to a coordinated simulation framework involving multiple vehicles.

3.2 Transition from Single-instance to Multi-instance Architecture

Starting from the single fixed-wing path-following model, the transition to a multi-instance architecture consists of extending the same guidance logic from one UAV to multiple UAVs simulated simultaneously.

In the single-instance case one set of state variables, one set of control inputs and one trajectory reference are associated with a single aircraft.

In the multi-instance case the same logic is applied in parallel to a group of vehicles, each characterized by its own state and its own waypoint sequence.

This extension is directly supported in UAV Toolbox by the `Multi-Instance Guidance Model` block, shown in Figure 3.6.

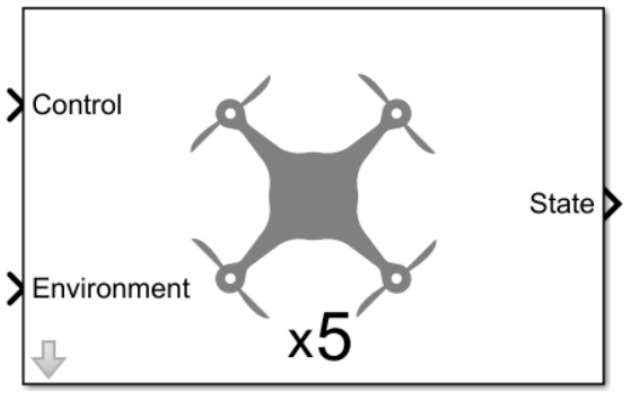


Figure 3.6: `Multi-Instance Guidance Model` block in the MATLAB/Simulink environment.

The block allows multiple UAVs to be simulated within the same model using simplified fixed-wing or multirotor kinematic models together with autopilot controllers. For this reason, it provides a suitable starting point for representing a UAV swarm in the proposed simulation framework.

Conceptually, the main change concerns how signals are organised and how the system is scaled. In the single-UAV model variables such as commands, states and environmental quantities are handled for one vehicle only. In the multi-instance model, instead, these quantities are arranged in vector form, allowing the same guidance logic to be applied simultaneously to all UAVs. The fixed-wing path-following approach is therefore maintained but extended to multiple aircraft instances within the same simulation framework.

This passage from single-instance to multi-instance modelling is particularly relevant for swarms. A single fixed-wing model is sufficient to study the behaviour of one vehicle following a predefined path, but it cannot directly represent the simultaneous evolution of multiple aircraft with distinct trajectories, shared initial conditions or coordinated navigation objectives. By contrast, the multi-instance framework makes it possible to simulate multiple UAVs within one unified Simulink architecture, while maintaining the same basic guidance concepts used in the single UAV case.

For this reason, the single fixed-wing path-following model can be interpreted as the conceptual baseline, while the multi-instance model represents its scalable extension towards coordinated multi-UAV simulation.

This transition is important because it extends the analysis from a single UAV following predefined waypoints to multiple fixed-wing UAVs operating at the same time within the same simulation framework.

3.3 Simulink Model of the Multi-UAV Fixed-wing System

The Simulink model `formationFlightSimulationblock`, shown in Figure 3.7, represents a multi-UAV fixed-wing guidance and visualisation architecture. It is designed to simulate the motion of five autonomous aerial vehicles, each following a different waypoint sequence. The overall structure is organised as a closed loop guidance system, where the current state of each vehicle is fed back to the waypoint-following and control blocks in order to continuously update the commanded motion.

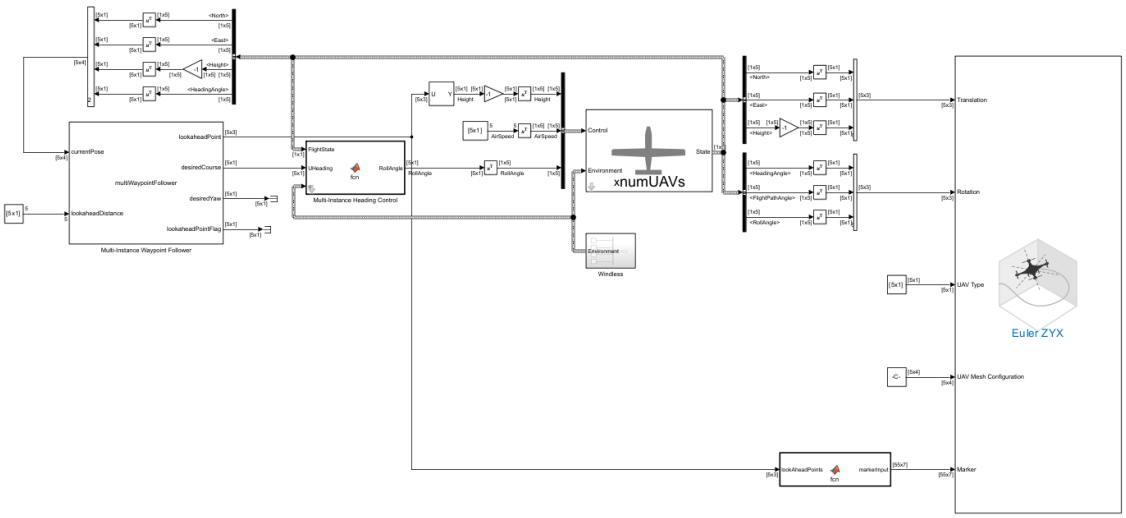


Figure 3.7: Simulink model `formationFlightSimulationblock` for the multi-UAV fixed-wing guidance and visualisation architecture.

The model is composed of five main functional sections:

1) Waypoint-based Trajectory Generation;

2) Heading Control;

3) Vehicle Guidance Dynamics;

4) Environmental Modelling;

5) 3D Animation and Graphical Marker Visualisation.

3.3.1 Waypoint Generation and Multi-UAV Guidance

The simulation starts from the `Multi-Instance Waypoint Follower` block, shown in Figure 3.8. This subsystem generates the reference trajectory information for all UAVs simultaneously, using as inputs the current pose of each vehicle and the selected look-ahead distance.

The current pose is obtained from the vehicle state variables by extracting the north position, east position, height and heading angle. The *North* and *East* components define the UAV position in the local horizontal reference frame, *Height* represents the vertical position, while the *Heading Angle* describes the vehicle orientation in the horizontal plane. These quantities are rearranged through transpose and concatenation operations so that the waypoint follower receives the input data in the matrix format required for the five UAVs.

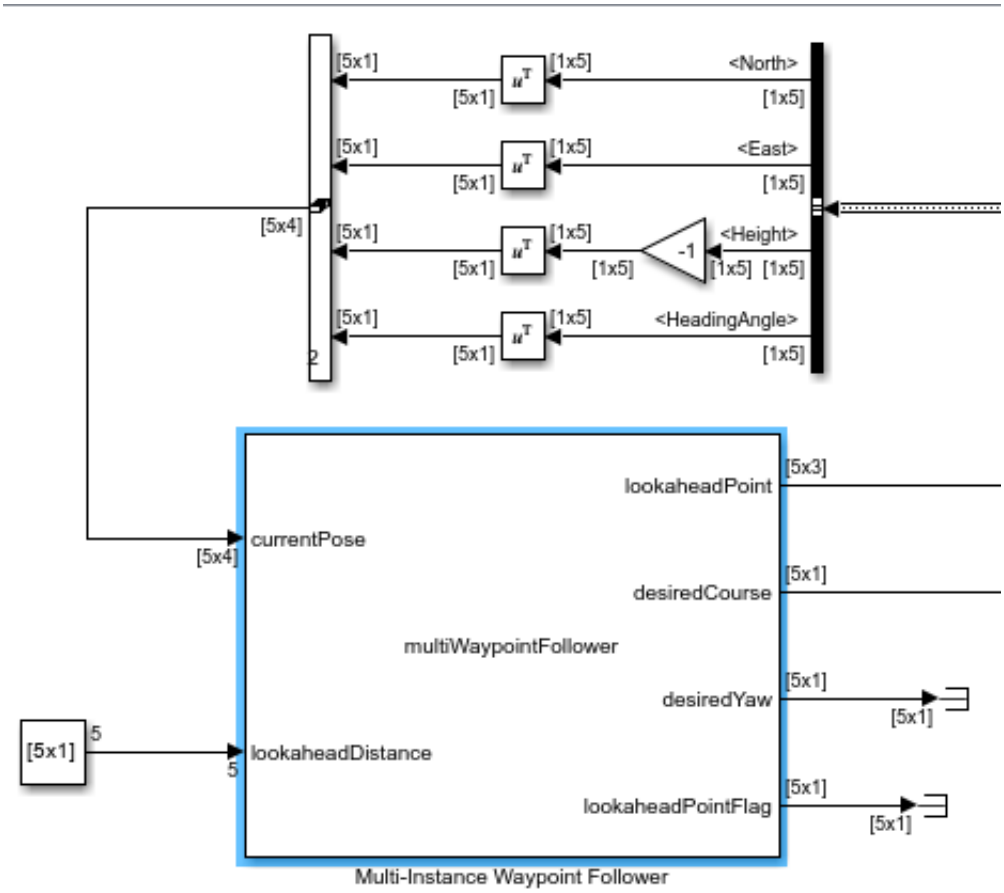


Figure 3.8: `Multi-Instance Waypoint Follower` subsystem in the MATLAB/Simulink environment.

For each vehicle, the waypoint-following logic computes a look-ahead point, a desired

course, a desired yaw angle and a transition flag. Instead of forcing the UAV to aim directly at the next waypoint, the look-ahead strategy defines an intermediate target point along the path. This produces smoother trajectory tracking and reduces abrupt heading variations, which is particularly relevant for fixed-wing aircraft.

In the adopted implementation, this functionality is based on the MATLAB System object `multiWaypointFollower`, provided in the MathWorks `multi-UAV Simulink reference` example [19]. Since this object was not developed from scratch in this thesis, its complete source code is not reported in the appendix. The discussion therefore focuses on its role within the adopted architecture and on how it extends the waypoint-following logic from a single UAV to multiple vehicles.

The `multiWaypointFollower` object manages a collection of standard `uavWaypoint Follower` instances, assigning one follower to each UAV. This structure avoids duplicating the complete waypoint-following subsystem for every aircraft, while allowing each vehicle to track its own waypoint sequence within a single multi-instance simulation framework. A representative instruction showing the association between each UAV and its corresponding waypoint follower is reported in Code 3.1.

```
obj.WaypointFollowers{i} = uavWaypointFollower(...);
```

Code 3.1: Representative creation of one waypoint follower instance for each UAV, based on the MathWorks reference example [19].

At each simulation step, the object processes the UAVs sequentially. The corresponding waypoint follower receives the current pose and the look-ahead distance and returns the reference quantities required by the following control stage. This multi-instance behaviour can be represented as the iteration shown in Code 3.2.

```
for i = 1:obj.NumUAVs
...
% Apply the waypoint-following logic to the i-th UAV
...
end
```

Code 3.2: Representative multi-UAV iteration used by the waypoint follower object.

Therefore, the `Multi-Instance Waypoint Follower` preserves the guidance principle of the single fixed-wing UAV model, but applies it to several vehicles within the same simulation architecture. In this thesis, this block is used as part of the reference multi-UAV framework, while the custom contribution is introduced later through the additional mission-management and communication logic.

3.3.2 Heading Control Subsystem

The output of the waypoint follower is then processed by the `Multi-Instance Heading Control` block, shown in Figure 3.9. This subsystem is based on the heading control logic adopted in the MathWorks multi-UAV reference architecture [19]. Since the corresponding implementation was not developed or modified in this thesis, its complete source code is not reported in the appendix.

For each UAV the subsystem receives the current flight state, the commanded heading reference generated by the waypoint follower, the heading control gain and the environmental parameters associated with wind and gravity. Its objective is to convert the heading tracking error into a roll command suitable for the fixed-wing guidance model.

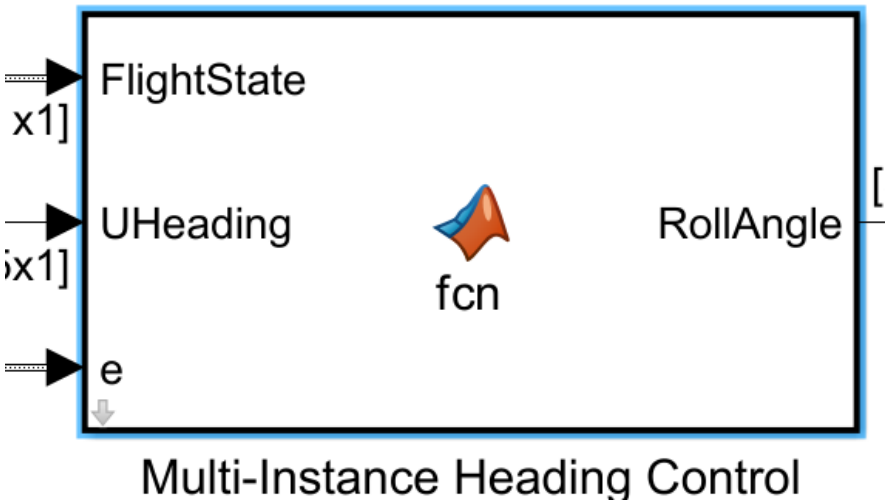


Figure 3.9: `Multi-Instance Heading Control` block in the MATLAB/Simulink environment.

For each vehicle, the heading control logic uses the current airspeed $V_{a,i}$, the heading angle ψ_i and the flight path angle γ_i, extracted from the flight state structure.

The index i identifies the generic UAV within the swarm.

The environmental contribution is represented by the wind vector

$$\mathbf{W}_i = \begin{bmatrix} W_{N,i} \\ W_{E,i} \\ W_{D,i} \end{bmatrix},$$

where $W_{N,i}$, $W_{E,i}$ and $W_{D,i}$ denote the north, east and down wind components acting on the i-th UAV.

The first relevant quantity is the wind corrected yaw angle, denoted as $\psi_{\text{yaw},i}$. This angle accounts for the effect of the horizontal wind components on the relation between the vehicle heading and its actual direction of motion. It can be expressed as

$$\psi_{\text{yaw},i} = \psi_i - \arcsin\left(\frac{1}{V_{a,i}}\begin{bmatrix} W_{N,i} & W_{E,i}\end{bmatrix}\begin{bmatrix}-\sin\psi_i & \cos\psi_i\end{bmatrix}\right).$$

The corresponding ground speed $V_{g,i}$ is then obtained from the airspeed and wind components through a kinematic correction. In the adopted formulation, it is expressed as

$$V_{g,i} = -b_i + \sqrt{b_i^2 - c_i},$$

where b_i and c_i are intermediate terms computed from the flight direction vector and the wind vector. This correction allows the heading controller to account for the difference between air-relative motion and ground-relative motion.

The heading tracking error is evaluated through the wrapped angular difference

$$e_{\psi,i} = \text{angdiff}(\psi_i, \psi_{c,i}),$$

where $\psi_{c,i}$ is the commanded heading generated by the waypoint follower. The use of the wrapped angular difference avoids discontinuities when the angles cross the $\pm\pi$ boundary.

Finally, the commanded roll angle is computed as

$$\phi_{c,i} = \text{atan2}\left(P_\psi e_{\psi,i} V_{g,i}, g_i \cos(\psi_i - \psi_{\text{yaw},i})\right).$$

where P_ψ is the heading-control gain and g_i is the local gravity term.

This relation converts the heading error into a roll angle command while considering both the ground speed and the difference between the heading angle and the wind corrected yaw angle. Although the subsystem follows a multi-instance architecture, the same heading-control law is applied independently to each UAV. The resulting roll angle commands are then collected in a single output vector and provided to the multi-instance guidance model.

Therefore, the `Multi-Instance Heading Control` block represents the intermediate stage between waypoint-based trajectory generation and vehicle motion propagation.

It receives the desired heading information from the waypoint follower and transforms it into the roll commands required to steer the fixed-wing UAVs along their assigned trajectories.

3.3.3 Guidance Model

The roll command produced by the heading controller is combined with other control variables and is passed to the central guidance model. This block represents the central dynamic component of the simulation and propagates the motion of all UAVs over time. In Figure 3.10 this block is shown as a multi-instance aircraft model labeled with `xnumUAVs`, indicating that the same guidance logic is replicated across the fleet. The guidance model receives two main inputs:

- the *Control* input, containing the commanded motion variables;
- the *Environment* input, containing the external environmental conditions.

Its output is the updated state of each UAV.

This state includes the main navigation and attitude variables, such as North position, East position, Height, Heading Angle, Flight Path Angle and Roll Angle. These state variables are then fed back into the waypoint follower and heading controller, closing the guidance loop.

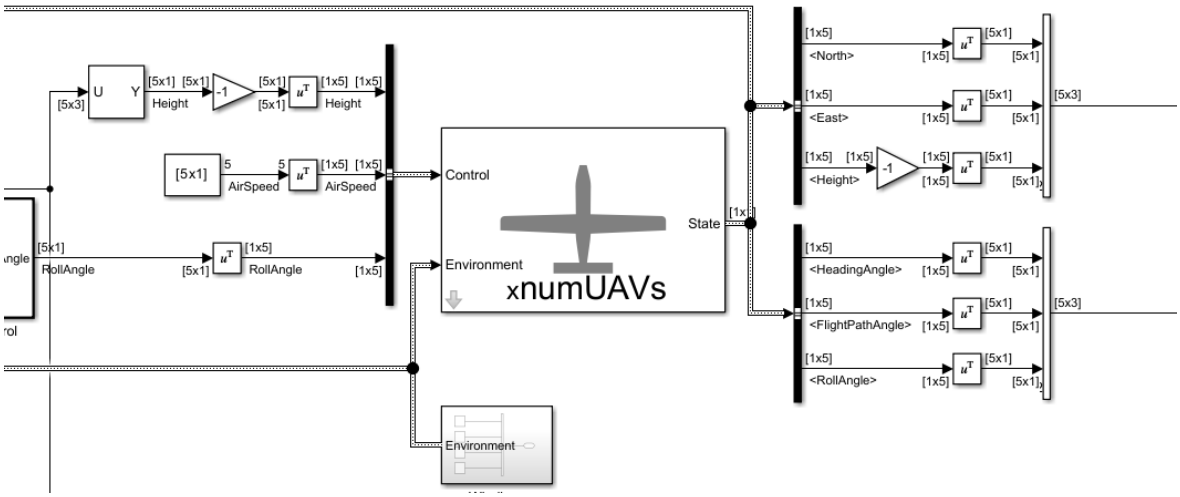


Figure 3.10: `Multi-Instance Aircraft Guidance Model` subsystem in the MATLAB/Simulink environment.

3.3.4 Environmental Subsystem

The model includes a subsystem labeled `Windless`, which provides the environmental input to the guidance model. This means that the simulation is currently carried out under ideal atmospheric conditions, with no wind disturbance acting on the vehicles. The use of a windless environment is particularly useful in the initial validation phase, because it allows the behaviour of the guidance and waypoint-following algorithms to be analyzed without the additional complexity introduced by external perturbations.

Although simplified, this subsystem is important because it provides a modular interface through which more realistic environmental effects, such as wind components or turbulence, could be introduced in future developments. The corresponding environmental input subsystem is shown in Figure 3.11.

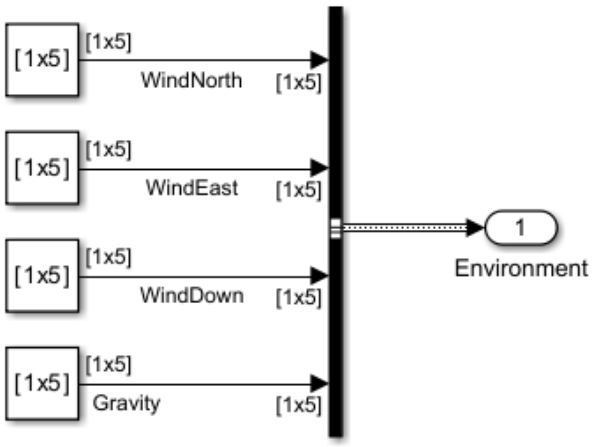


Figure 3.11: `Windless` subsystem in the MATLAB/Simulink environment.

3.3.5 Signal Formatting and Coordinate Handling

Several intermediate blocks are used to process the state signals before sending them either to the controller or to the visualisation subsystem.

In particular, transpose blocks are used to convert vectors between row-wise and column-wise formats, while gain blocks with value -1 are used to change the sign of the altitude related variable. This is necessary because different subsystems may adopt different conventions for vertical position, such as height versus down coordinates.

Therefore, the model includes these conversion steps to maintain consistency between the dynamic computation and the graphical animation.

3.3.6 3D Visualisation Subsystem

The `UAV Animation` block provides a three-dimensional visualisation of the simulated formation. This block receives:

- the *Translation* input derived from the North, East and Height positions of the UAVs;
- the *Rotation* input derived from Heading Angle, Flight Path Angle and Roll Angle;
- the *UAV Type* input which specifies the type of vehicle to be displayed in the animation;
- the *UAV Mesh Configuration* input which defines the graphical properties of the UAV model used in the 3D scene, such as its geometric mesh, visual appearance and rendering configuration;
- the *Marker* input which is used to display waypoint related visual elements.

The animation uses the Euler ZYX rotation convention to represent the attitude of each UAV. Consequently, each vehicle is displayed with both its correct position and orientation in space.

The corresponding `UAV Animation` subsystem is shown in Figure 3.12.

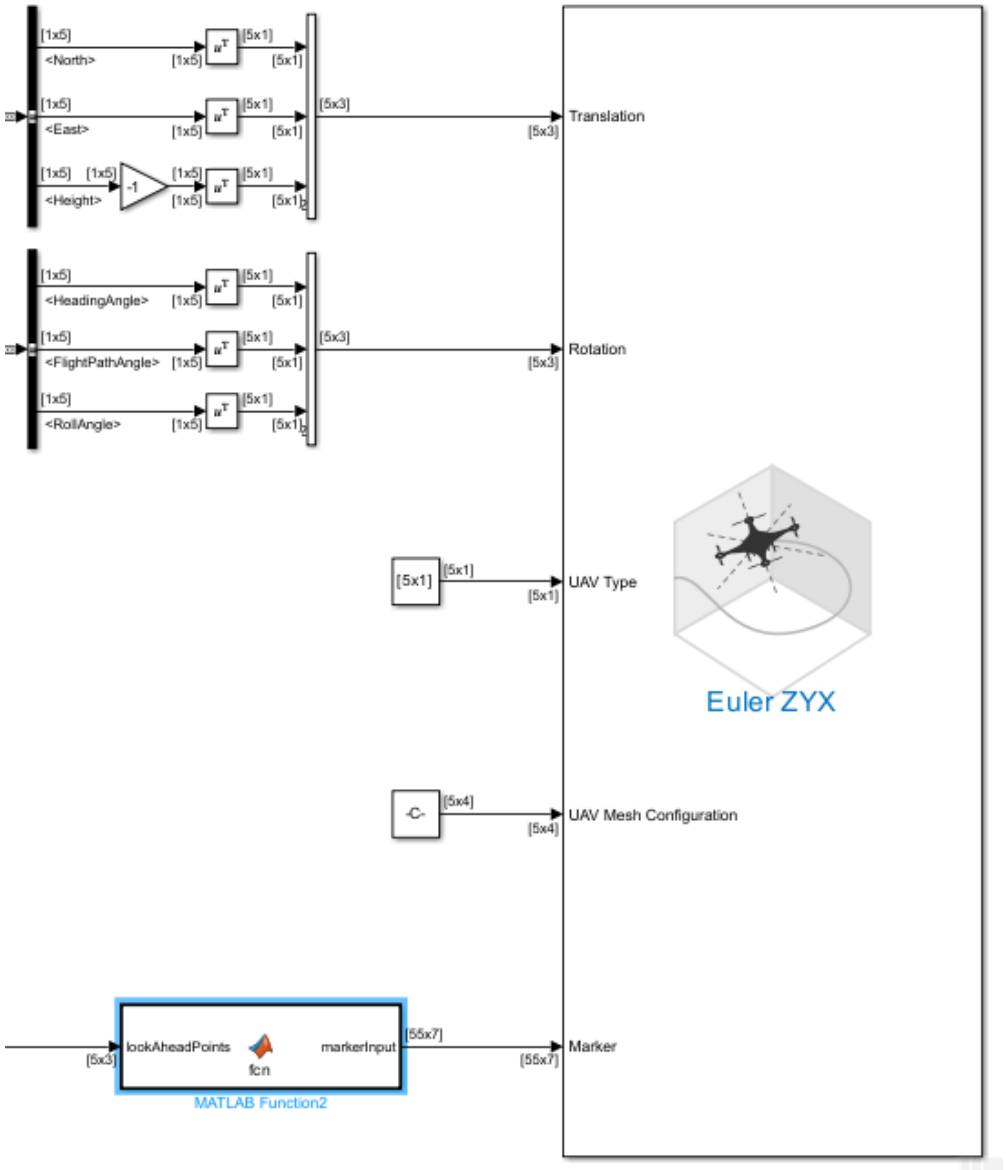


Figure 3.12: `UAV Animation` subsystem for multi-UAV 3D visualisation in the MATLAB/Simulink environment.

3.3.7 Marker Generation Subsystem

In the same `UAV Animation` subsystem shown in Figure 3.12, the MATLAB Function block located at the bottom of the model generates the variable *markerInput*, which is then provided to the animation block. This variable defines the graphical markers associated with the waypoint-following process, allowing both the predefined waypoint locations and the current look-ahead points to be displayed in the 3D scene.

The function receives the current look-ahead points and the waypoint array associated with the UAV swarm. Since the waypoint array is organised as a three-dimensional matrix, the number of UAVs and the number of waypoints are obtained directly from its dimensions. The output matrix *markerInput* is then initialized so that each row corresponds to a graphical marker, while the seven columns contain the marker position, size and RGB colour components.

```
markerInput = zeros(numWayPoints*numUAVs + numUAVs, 7);
```

Code 3.3: Representative initialization of the marker input matrix.

The first portion of *markerInput* is used to store the waypoint markers. For each UAV, the corresponding waypoint positions are assigned to the first three columns of the matrix. The remaining columns are used to define the marker size and colour, so that the waypoint set of each vehicle can be displayed in the 3D animation.

```
markerInput((i-1)*numWayPoints+1:i*numWayPoints, 1:3) = ...
wayPoints(:,:,i);
```

Code 3.4: Representative assignment of waypoint marker positions.

The final rows of the matrix are reserved for the current look-ahead points. These points represent the instantaneous guidance targets generated by the waypoint-following algorithm and are displayed together with the fixed waypoint markers.

```
markerInput(numWayPoints*numUAVs+1:numWayPoints*numUAVs+numUAVs,
    1:3) = ...
lookAheadPoints(:,:);
```

Code 3.5: Representative assignment of look-ahead point markers.

Therefore, the marker generation subsystem acts as a visualisation interface for the guidance process. By displaying both the assigned waypoint trajectories and the instantaneous look-ahead points, it improves the interpretation of the multi-UAV path-following behaviour and supports the visual validation of the simulated formation. In this thesis, the subsystem is used as part of the reference visualisation architecture, while the custom mission logic is introduced separately through the communication and coordination blocks.

3.4 Overall Working Principle

From a system level perspective, the model operates through a sequential information flow. The current UAV states are first extracted and converted into pose information, which is provided to the Waypoint Follower. Based on the assigned waypoint sequence, the Waypoint Follower generates the local navigation references, including the look-ahead point, desired course and desired yaw for each UAV. These references are then processed by the Heading Control subsystem, which computes the roll angle command required to guide each vehicle along its trajectory.

The resulting control commands, together with the environmental inputs, are passed to the multi-instance guidance model, which updates the flight state of each UAV over time. The updated states are then reused by the guidance and control blocks and are also sent to the 3D visualisation subsystem. In this way, the architecture provides a complete framework for tracking the assigned trajectories of multiple UAVs through continuous feedback from the simulated flight states.

This organization is particularly suitable for the study of cooperative autonomous systems, since it integrates trajectory generation, control, multi-vehicle scalability and real time visualisation within a single simulation environment. Moreover, its modular structure allows future extensions, such as the introduction of communication logic between platforms, disturbance rejection strategies or hardware in the loop validation.

3.5 UAV Swarm Model Validation

To validate the proposed UAV swarm model, different simulation scenarios were performed. The validation process was based on the MathWorks reference example for simulating multiple fixed-wing UAVs in a waypoint-following mission using Simulink System objects [19]. Starting from this reference configuration, additional trajectory profiles were introduced to evaluate the tracking performance of the proposed multi-UAV architecture under more complex spatial maneuvers.

The first scenario considers the UAVs following parallel U-shaped paths. Each trajectory includes an initial straight segment, followed by a semicircular turn where the altitude first increases and then decreases in a symmetric way. This scenario was used to evaluate whether the UAVs can turn in a coordinated way while maintaining their formation. The resulting trajectory is shown in Figure 3.13.

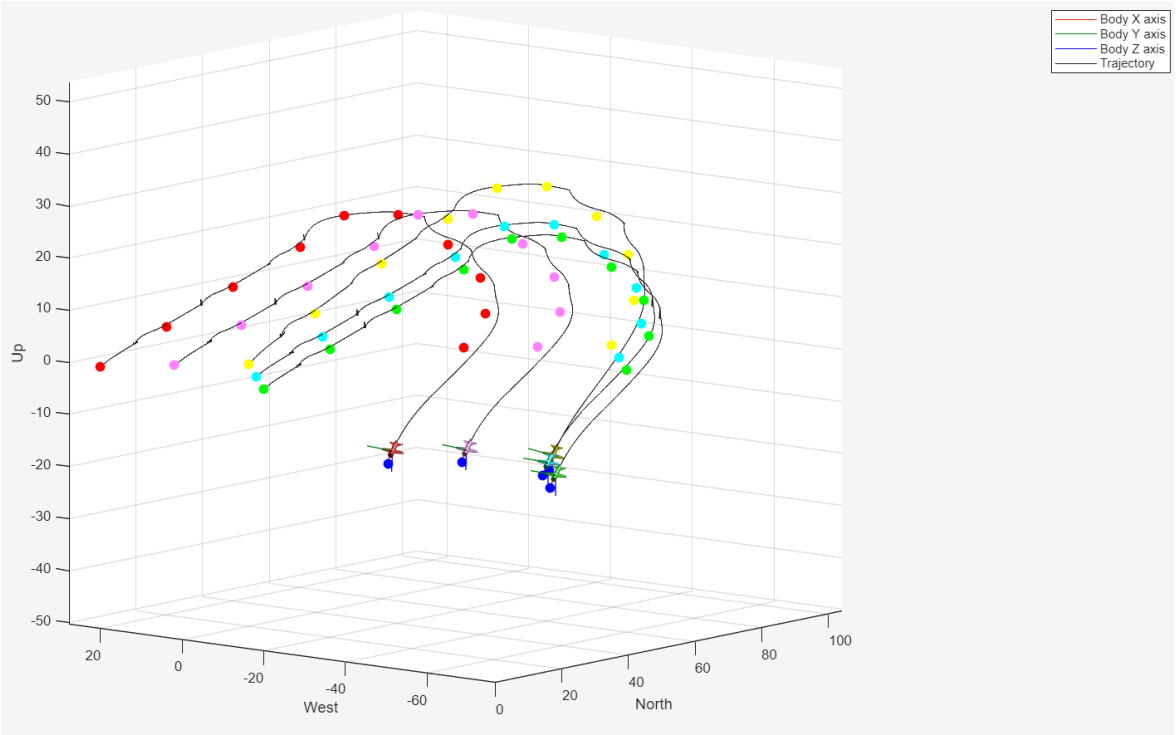


Figure 3.13: Semicircular turnaround trajectories of the five UAVs. Each coloured dotted line represents the trajectory followed by a different UAV: red for UAV 1, purple for UAV 2, yellow for UAV 3, light blue for UAV 4 and green for UAV 5.

The second scenario considers a zigzag trajectory with varying altitude. In this case, the UAVs follow a path characterized by alternating direction changes and height variations. This test was used to assess the swarm response to a less regular trajectory than the previous one, as reported in Figure 3.14.

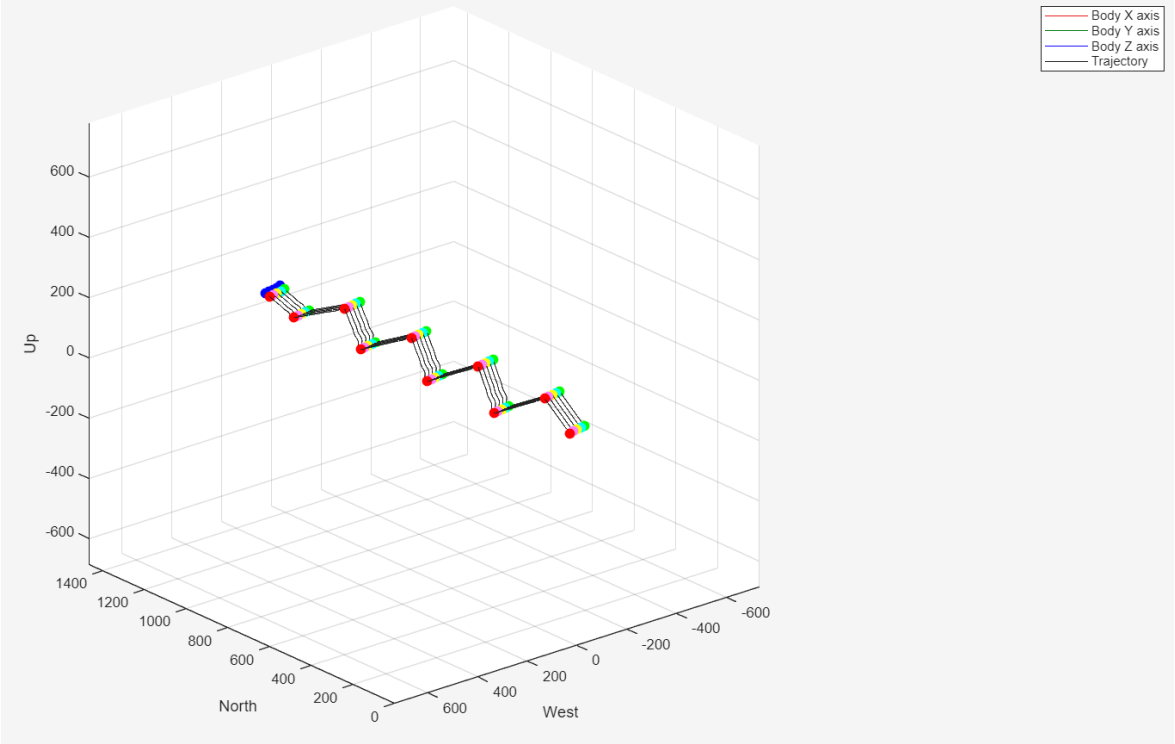


Figure 3.14: Zigzag trajectories with altitude variation of the five UAVs. Each coloured dotted line represents the trajectory followed by a different UAV: red for UAV 1, purple for UAV 2, yellow for UAV 3, light blue for UAV 4 and green for UAV 5.

The simulation results show that the implemented framework is capable of handling a wide range of trajectories, including both linear and nonlinear paths. At the same time, they emphasize a key design constraint related to waypoint generation: the spacing between consecutive waypoints must be sufficiently large to allow the UAVs to respond dynamically and reach the desired reference in a physically consistent manner. When waypoints are too closely spaced, the vehicles may be forced to perform abrupt maneuvers, with correspondingly high velocity demands and degraded tracking accuracy.

Chapter 4

Maritime Simulation Framework

The Marine Systems Simulator (MSS) library was selected as the reference framework for the maritime component because it provides an established MATLAB/Simulink environment for the modelling, simulation and control of marine systems. MSS includes several vessel models, control algorithms and documented simulation examples, offering a solid basis for the development of maritime applications [20, 21].

A relevant advantage of MSS is its direct compatibility with MATLAB/Simulink. This aspect is particularly important in the present work, since the UAV subsystem and the communication architecture are also developed within the same simulation environment. As a result, the maritime platform can be included without introducing a separate software framework, making the connection between the different parts of the system more straightforward.

Another reason for selecting MSS is the balance it offers between model accuracy and practical implementation. The aim of this thesis is not only to reproduce the motion of a ship, but also to study the exchange of information, the cooperative behaviour and the coordination strategies between heterogeneous platforms. For this reason, the use of validated marine models and preconfigured simulation demos reduces the time required to build the maritime subsystem from the beginning. This allows more attention to be devoted to the development of the communication logic, the interaction with the UAV swarm and the testing of the overall cooperative architecture.

4.1 Selection of the Ship Model

Among the vessel models available in MSS, the Mariner class cargo ship was selected as the maritime platform for this work. According to the MSS documentation, `mariner.m`

describes a 160 m cargo ship and models its manoeuvring behaviour in the horizontal plane [20]. This model is particularly suitable for the proposed simulation framework because MSS also provides Simulink demonstrations based on the same vessel, including examples for heading autopilot and line-of-sight (LOS) path-following control [20, 21], while the corresponding Simulink block is shown in Figure 4.1.



Figure 4.1: Simulink block of the Mariner class cargo ship model available in the MSS library [20].

The choice of the Mariner class cargo ship was motivated by several considerations:

- The Mariner class cargo ship is coherent with the operational context considered in this thesis. The proposed scenario involves the cooperation between a swarm of five UAVs and a large maritime unit. In this context, a cargo ship represents a realistic reference platform for activities such as maritime monitoring, surveillance, inspection and coordination. Smaller vessels, underwater vehicles or more specialized platforms available in MSS would be less representative of the type of maritime asset considered in this work.

- Its size and nature make it suitable for a cooperative scenario involving aerial vehicles. A large ship provides a meaningful maritime platform around which the UAV swarm can operate, exchange information and perform coordinated tasks. This aspect is particularly relevant in the implemented mission logic, since the aerial vehicles are not only used for reconnaissance and communication support, but are also required to return to the ship during the final return-and-docking phase.

  In this context, the physical dimensions of the maritime platform become an important modelling aspect. Since the simulated swarm consists of five UAVs, the ship must represent a sufficiently large target for the recovery phase to be meaningful. Even though the docking process is modelled in a simplified way,

the UAVs are assumed to have non negligible physical dimensions and cannot realistically be recovered by a small or point-like platform. The use of a large cargo vessel therefore provides a more consistent reference scenario, where the ship can act both as the main maritime unit of the mission and as the recovery platform for the aerial vehicles.

This choice is useful for studying how the UAVs interact with the maritime platform, how position data are exchanged and how the overall cooperative system evolves during the mission, from the reconnaissance phase to the final return and recovery procedure.

- The model offers a good compromise between realism and computational simplicity. Although MSS includes more complex vessel models with a higher number of degrees of freedom, the 3 DOF formulation of the Mariner model is sufficient to describe the main manoeuvring behaviour of the ship in the horizontal plane. This level of detail is adequate for the objectives of the developed simulation framework, where the ship dynamics must be realistic enough to support the cooperative scenario, but not so complex as to make the overall simulation unnecessarily demanding.

- The computational efficiency of the Mariner model is also important for the development of the complete architecture. Since the maritime platform has to be connected with the UAV subsystem, the communication blocks and the supervision logic, a simpler but still representative ship model makes the simulation easier to manage. This is especially useful when additional modules are introduced, such as position exchange or coordination strategies.

- The availability of existing Simulink demonstrations provides a practical starting point for the development of the maritime subsystem. Since the model is already included in documented MSS examples, it can be extended more easily. This reduces the risk of implementation errors and supports a more structured development process.

For these reasons, the Mariner class cargo ship was considered more appropriate than the other MSS vessel models examined during the preliminary phase.

The alternative models were not adopted because they were either less representative of the selected operational context or unnecessarily detailed for the level of complexity required in this study. The Mariner model therefore provides a suitable basis for representing the maritime component while keeping the focus on the cooperative interaction between the ship, the UAV swarm and the communication architecture.

4.2 Simulink Model of the Mariner Cargo Ship

The Simulink model used for the maritime platform is based on the MSS demonstration model `demoMarinerPathFollowingCourseControl`, which implements a path following and course control architecture for a Mariner class cargo vessel. As shown in Figure 4.2, the original scheme implements a line-of-sight (LOS) path-following algorithm together with a course autopilot for the maritime platform.

This model was used as a reference starting point for the development of the maritime subsystem. However, as for the UAV subsystem, the original architecture was not used without modifications. It was adapted and extended in order to meet the specific requirements of the proposed cooperative scenario, including the interaction with the UAV swarm, the exchange of position information and the connection with the communication and supervision layers.

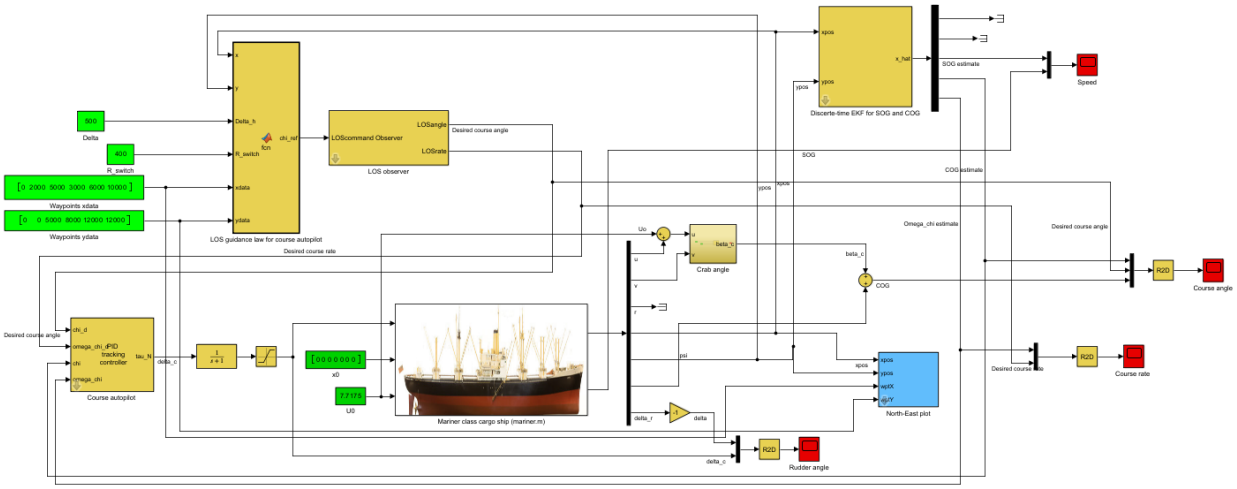


Figure 4.2: Simulink model `demoMarinerPathFollowingCourseControl` for path following and course control of the Mariner class cargo ship, available in the MSS library [20].

The control structure adopted in the MSS path-following example is related to the approach presented in [16]. In that work, the vessel follows a predefined path by combining a LOS based reference generation strategy with a course control loop. In addition, an extended Kalman filter is used to obtain navigation information, such as speed over ground (SOG) and course over ground (COG), starting from GNSS position measurements.

In the following sections, the description of the block connections and parameter choices is based on the direct analysis of the customized Simulink model developed for this

thesis. The role of the main MSS subsystems is instead interpreted with reference to the MSS documentation, the MSS overview paper and the path-following formulation proposed by Fossen [16, 20, 21].

4.2.1 Path Definition and LOS Guidance

The model begins with the geometric definition of the reference path through the constant blocks *Waypoints xdata* and *Waypoints ydata*, which store the North and East coordinates of the waypoint sequence. Since the ship motion is described in the horizontal plane, no vertical position component is required. These two signals are provided both to the LOS guidance subsystem and to the North-East plotting block. Therefore, the waypoint vectors have a dual role in the model: they define the path to be followed by the ship and provide the reference geometry used for trajectory visualisation.

As shown in Figure 4.3, the parameters $\Delta = 500$ and $R_{\text{switch}} = 400$ are connected to the LOS guidance block. The look-ahead distance Δ contributes to the generation of the desired course and affects the way in which the vessel converges towards the reference path. Smaller values generally lead to a faster and more aggressive convergence, whereas larger values produce a smoother but less reactive trajectory [16]. In this work, $\Delta = 500$ was selected to obtain a suitable compromise between responsiveness in path convergence and smoothness of the vessel motion.

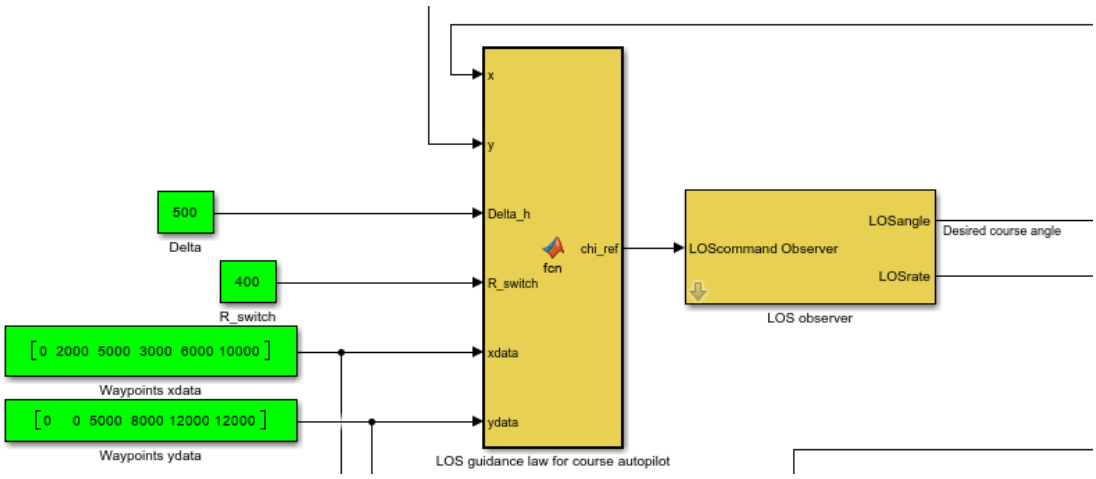


Figure 4.3: `LOS guidance` and `LOS observer` blocks implemented in the MATLAB/Simulink environment.

The parameter R_{switch} defines the distance threshold used to switch from the current waypoint to the next one. In the customized model, it was set to $R_{\text{switch}} = 400$, so that

the transition to the next path segment occurs before the waypoint is reached exactly. This anticipatory switching helps reduce abrupt changes in the course reference and supports smoother manoeuvring between consecutive path segments.

The LOS guidance subsystem receives the vessel position, the waypoint coordinates, the look-ahead distance Δ and the switching radius R_{switch} as input signals. These quantities are processed by the `LOSchi` function, which generates the course reference χ_{ref} required to guide the vessel along the desired path.

The purpose of the LOS guidance block is to convert the path defined by the waypoints into a course command for the vessel. This is consistent with the line of sight path-following method described in [16], where the desired course depends on the direction of the path, the distance of the vessel from the path and the selected look-ahead distance.

For this reason, the guidance block represents the first step of the path-following process. It does not directly control the ship actuators, but provides the reference course to be followed by the vessel. This signal is then sent to the observer and to the inner control loop, which complete the control structure [16, 20, 21].

4.2.2 LOS Observer and Reference Smoothing

The reference generated by the guidance law is processed by the observer shown in Figure 4.4. This subsystem acts as an intermediate stage between the path-following logic and the course autopilot, transforming the original command into smoother reference signals for the controller.

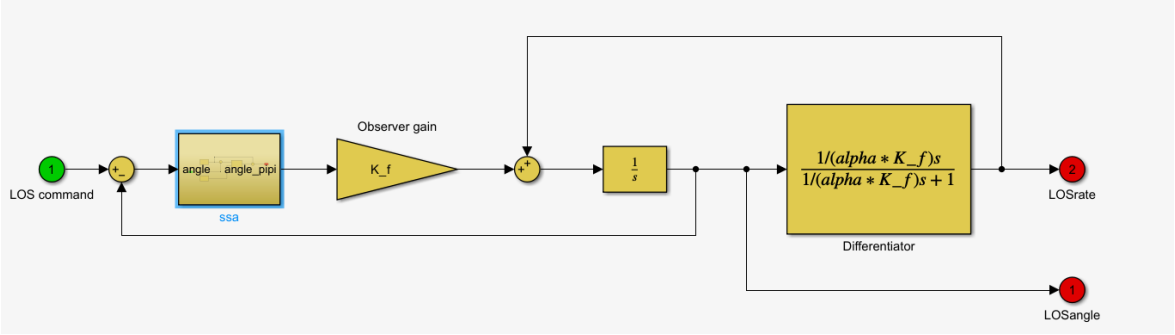


Figure 4.4: `LOS observer` subsystem implemented in the MATLAB/Simulink environment.

The input signal, denoted as *LOS command*, is compared with the filtered course reference generated inside the block. The resulting angular error is passed through a function used to avoids discontinuities when the angle crosses the $\pm\pi$ boundary. The

error is then multiplied by the observer gain K_f, which determines how quickly the observer follows the LOS command.

After this correction, the signal is integrated to obtain the desired course angle, denoted as *LOSangle*. In parallel, a filtered differentiating action is used to generate the desired course rate, denoted as *LOSrate*. As a result, the autopilot receives both the course reference and information about how fast this reference is changing.

This structure helps reduce abrupt variations in the command sent to the `Course Autopilot`, especially during waypoint transitions. This is consistent with the LOS path-following approach in [16], where filtering is used to avoid discontinuities in the reference signals during waypoint switching.

As a result, the controller is provided with smoother and more continuous inputs, supporting a more stable path-following behaviour of the vessel.

4.2.3 Course Autopilot and Rudder Command

The inner loop of the model is the `Course autopilot`, which receives four signals: the desired course angle (*chi_d*), the desired course rate coming from the LOS observer (*omega_chi_d*), the estimated course over ground (*chi*) and the rate of change of the course over ground (*omega_chi*) coming from the estimator. Its output is the commanded rudder angle *delta_c*.

The course control level of the path-following architecture is represented by the `Course autopilot`, as depicted in Figure 4.5. In the reference approach, the control system is organised in two main levels: a guidance level and a course control level, the first one generates the desired course and the second one computes the rudder command required to follow it [16]. Therefore, the autopilot compares the desired course information with the estimated motion of the vessel and generates the commanded rudder angle.

The internal implementation of the `Course autopilot` block is not reported in the appendix, since it belongs to the original MSS architecture and was not modified in this thesis. Instead, the block is described from a functional point of view, focusing on its role within the adopted path-following scheme.

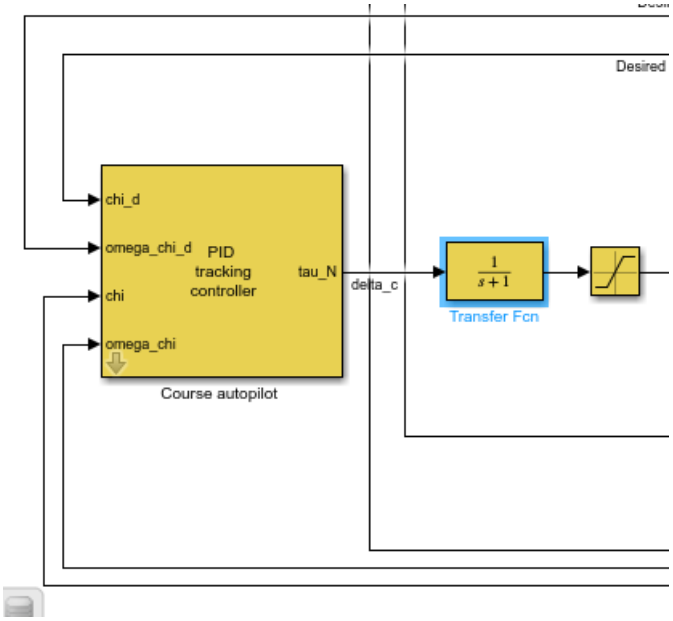


Figure 4.5: `Course autopilot` block implemented in the MATLAB/Simulink environment.

In the Simulink model, the internal equations of the controller are contained inside the MSS `Course autopilot` block. However, its role is clear: it receives the desired course angle and course rate, compares them with the estimated course variables and produces the rudder command needed to reduce the tracking error. After this block, the command passes through a first order transfer function and a saturation block limited to $\pm 40°$. These two blocks condition the rudder command before it is applied to the ship model: the transfer function makes the command smoother, while the saturation block keeps the rudder angle within a physically admissible range.

4.2.4 Mariner Vessel Model

The vessel dynamics are represented by the `mariner.m` file, which implements the Mariner class cargo ship model included in MSS [20]. This block receives three inputs: the rudder command, the initial state vector x_0 and the nominal forward speed U_0.

In the adopted simulation x_0 is set to zero while U_0 is set to 7.7175 m/s because it defines the reference operating speed of the vessel.

As shown in Figure 4.6, the first output of the Mariner block is demultiplexed into seven signals: u, v, r, x_{pos}, y_{pos}, ψ, and δ_r. The second output is *SOG*.

These signals provide the body fixed velocities, the planar position, the heading angle, the actual rudder angle and the SOG required by the other parts of the architecture.

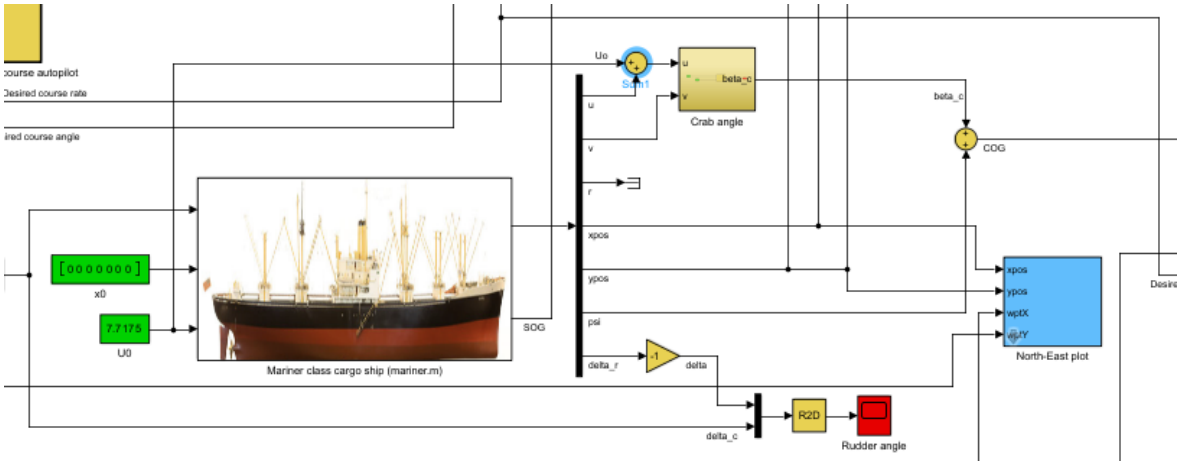


Figure 4.6: Mariner ship model and kinematic reconstruction of the COG implemented in the MATLAB/Simulink environment.

The function describes the ship motion in the horizontal plane by considering three main components: surge, sway and yaw. This 3 DOF representation is sufficient for the objectives of the developed simulation framework, since the maritime platform is mainly required to follow a path and interact with the UAV swarm at the guidance and communication level. At the same time, the model remains simple enough to be integrated with the other subsystems of the proposed architecture [20, 21].

From a computational perspective, the function evaluates the state derivatives based on the current vehicle state and the applied control inputs. The state vector is composed of the body fixed velocities (surge u, sway v and yaw rate r), the Earth fixed navigation coordinates (North x, East y and heading ψ) and the actual rudder deflection angle δ_r.

At each simulation step, these quantities are updated by combining the vessel dynamic equations with the kinematic transformation between the body fixed and Earth fixed reference frames. Therefore, the function provides the evolution of the ship state over time and allows the updated variables to be used by the guidance, control, estimation and communication subsystems [20, 21].

The model also includes the effect of the rudder dynamics. The steering system is represented in a simplified way, so that the rudder does not change instantaneously and its angle remains within the physical limits of the vessel. This behaviour is also represented by the first order transfer function and the saturation block applied before the command enters the ship model.

4.2.5 COG Reconstruction and EKF

A particularly important part of the model is the kinematic reconstruction of the actual COG, shown in Figure 4.7. The longitudinal velocity signal u is first summed with U_0 in the block `Sum1` and then sent, together with the lateral velocity v, to the `Crab angle` block.

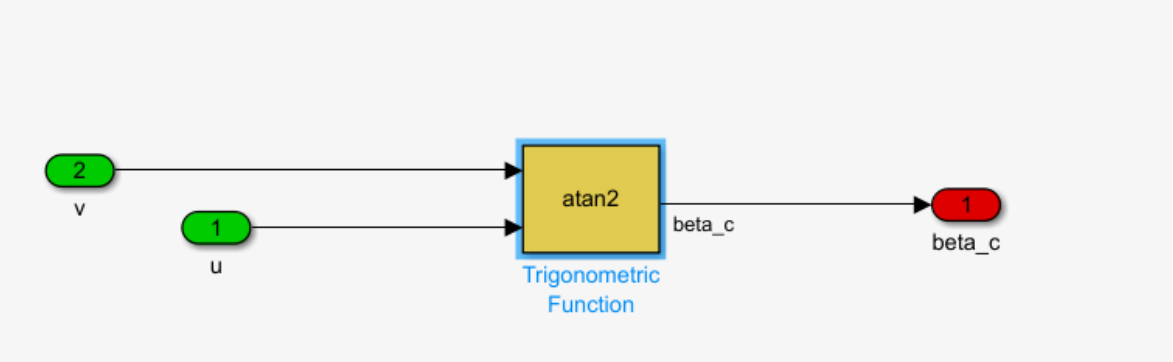


Figure 4.7: `Crab angle` computation subsystem implemented in the MATLAB/Simulink environment.

The `Crab angle` block produces the signal β_c. In marine navigation, the crab angle represents the difference between the vessel heading and the actual direction of motion. Therefore, the course over ground χ can be obtained by combining the heading angle ψ with the crab angle β_c [16]:

$$\chi = \psi + \beta_c.$$

This operation is implemented in the `Sum2` block, where ψ and β_c are added to obtain the signal labeled *COG*. This branch reconstructs the actual direction of travel of the vessel from its heading and body fixed velocity components. This step is important because the adopted control strategy is based on course following rather than on heading control alone [16, 20].

The navigation and state estimation stage of the model is implemented by the block labeled `Discrete-time EKF for SOG and COG`, shown in Figure 4.8. This block receives the vessel position signals x_{pos} and y_{pos} as inputs and estimates the navigation quantities required by the feedback control loop.

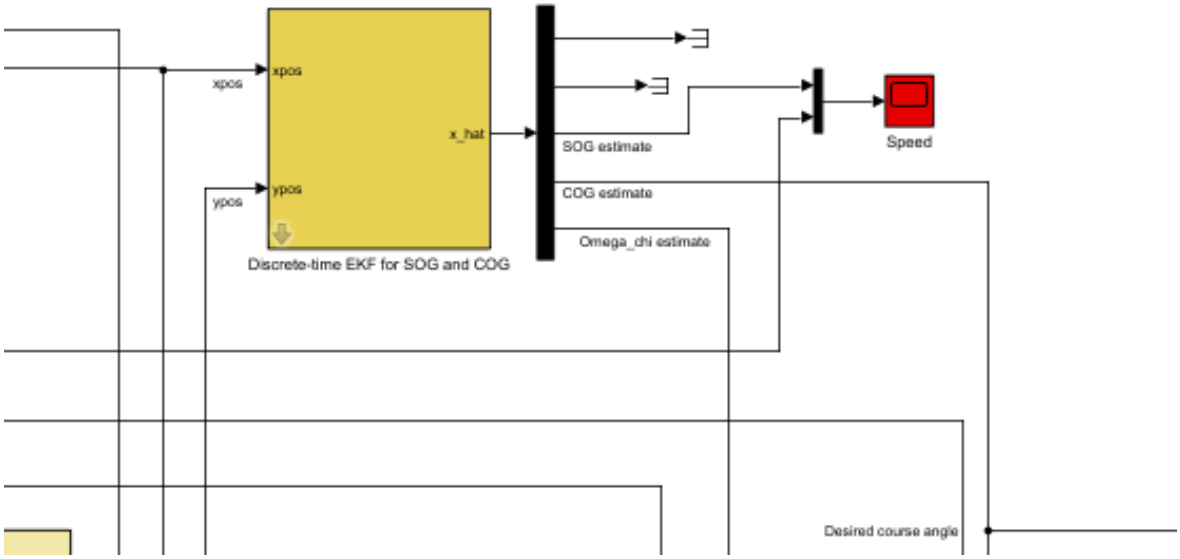


Figure 4.8: `Discrete-time EKF for SOG and COG` block implemented in the MATLAB/Simulink environment [20].

This structure is consistent with the reference approach in [16], where North and East GNSS position measurements are used to estimate the vessel position, SOG, COG and course rate. This approach allows the main navigation variables to be reconstructed from position measurements, reducing the need for direct course measurements from additional sensors, such as a gyrocompass.

From a practical point of view, the EKF processes the time evolution of the measured vessel position and provides filtered estimates of the quantities required by the Course autopilot. Therefore, this block plays an important role in closing the control loop, since it supplies the estimated motion information used to compare the actual vessel behaviour with the desired course reference.

The EKF output is then demultiplexed into five signals. The first two correspond to the filtered estimates of the vessel North and East positions. These quantities are not used further in the adopted control architecture because the measured position signals are already available for trajectory representation and waypoint-based guidance, while the Course Autopilot does not require position estimates as direct feedback inputs. For this reason, the first two outputs do not contribute to the subsequent control computations and are terminated. The remaining three outputs are explicitly labelled *SOG estimate*, *COG estimate* and *Omega_ chi estimate*, since they represent the navigation quantities required by the feedback control loop.

The remaining blocks of the model are mainly dedicated to monitoring and interpre-

tation of the closed loop behaviour. The North-East plot receives the actual ship position together with the waypoint coordinates and is used to show how closely the vessel trajectory follows the prescribed path.

The Mux blocks collect signals for comparison in the scopes. For example, the Speed scope compares the true SOG from the vessel model with the SOG estimate from the EKF, while the Course angle scope compares the desired course, the estimated course and the reconstructed true course. The Course rate scope plays the same role for the desired and estimated course rate signals.

In the branch related to the rudder signals, the commanded and actual rudder angles are combined and converted from radians to degrees (R2D) before visualisation. The gain of -1 applied to *delta_r* is only introduced to match the sign convention adopted in the plot.

4.2.6 Overall Interpretation

Taken as a whole, the model can be interpreted as an integrated marine guidance, navigation and control architecture. The waypoint coordinates define the desired path to be followed by the vessel, while the `LOS guidance` block converts this geometric information into a course command. This command is then processed by the `LOS observer`, which provides a smoother desired course angle and the corresponding course rate. These reference signals are necessary to avoid abrupt changes in the command sent to the controller, especially during waypoint transitions.

In parallel, the `EKF` estimates the main navigation variables starting from the vessel position. These estimated quantities are used by the `Course autopilot` to compare the desired motion with the actual motion of the ship and to compute the corresponding rudder command. Before this command is applied to the vessel model, it is processed by the actuator conditioning blocks, which account for the dynamic response and physical limitations of the rudder system. The `Mariner vessel` model then updates the ship motion, while the monitoring blocks allow the user to analyse the trajectory, the estimation performance and the rudder activity during the simulation.

This structure confirms the suitability of the selected model for the objectives of this thesis. The architecture is not limited to a simple kinematic path generator, but includes all the main elements required to represent a realistic marine control system: path definition, guidance logic, navigation estimation, course control, actuator limitations and vessel dynamics.

4.3 Validation of the Cargo Ship Model

The predefined Simulink scenario was used as an initial reference case to verify the correct operation of the model. This validation step was carried out before integrating the maritime platform with the UAV swarm, in order to check the behaviour of the ship path-following architecture independently from the cooperative scenario.

The waypoint sequence adopted for this baseline test is shown in Figure 4.9. The reference path is defined in the North-East plane and is composed of the following points:

[(0;0), (0;2000), (5000;5000), (8000;3000), (12000;6000), (10000,12000)]

This trajectory includes both straight segments and changes in direction, making it suitable for testing the response of the guidance, estimation and control blocks during waypoint transitions.

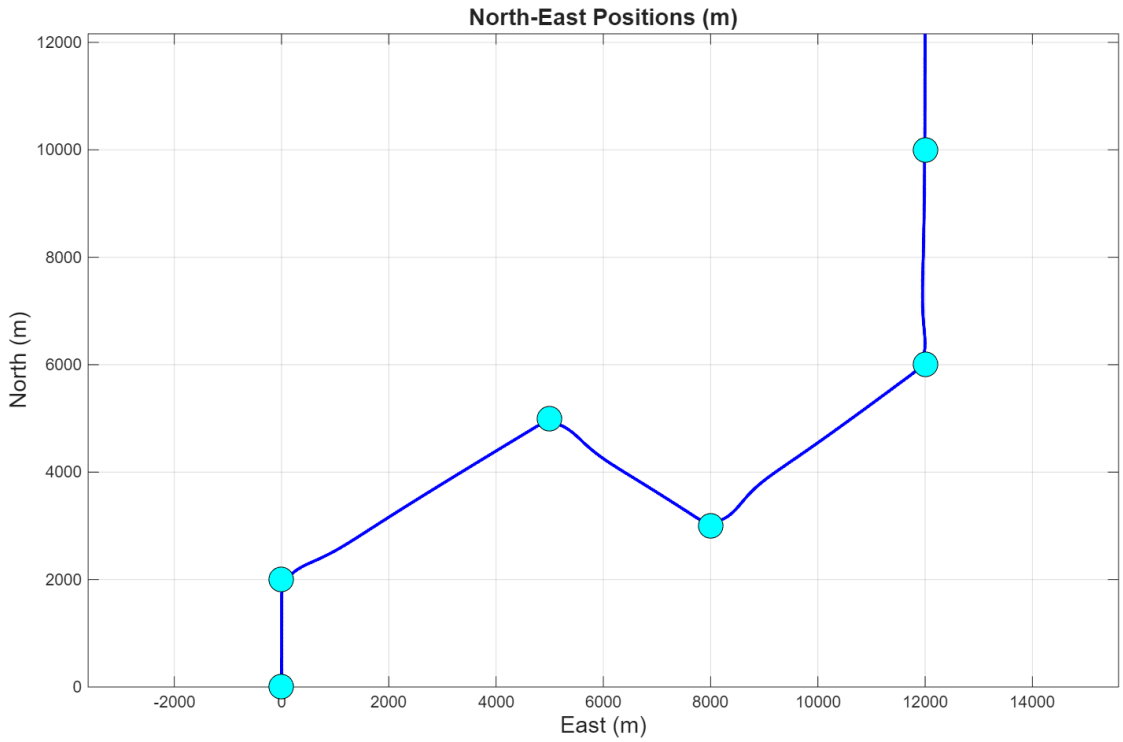


Figure 4.9: Reference waypoint sequence and corresponding ship trajectory in the North-East plane. The light-blue markers indicate the waypoints through which the ship is required to pass, while the continuous line represents the trajectory followed by the vessel.

The first validation output concerns the speed over ground (SOG), shown in Figure 4.10. The plot compares the true SOG provided by the vessel model with the estimated SOG generated by the EKF. The two curves show very good agreement over

the whole simulation, indicating that the estimator is able to reconstruct the vessel speed from the available position measurements. The small variations visible during the simulation are consistent with the manoeuvring phases of the vessel, especially when the path direction changes.

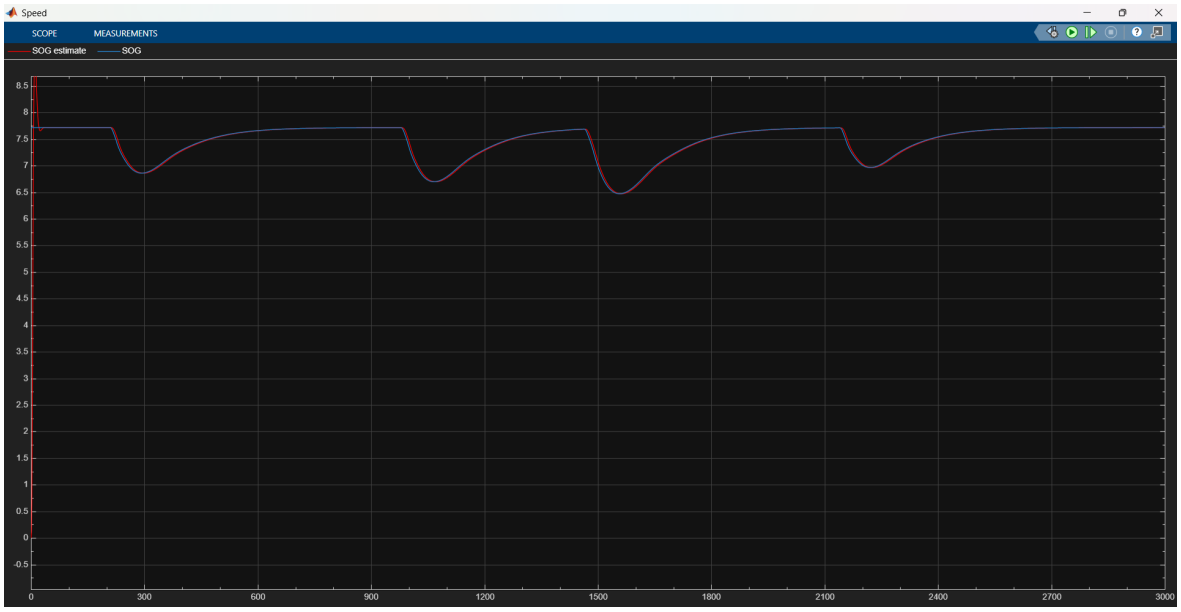


Figure 4.10: True and estimated Speed Over Ground (SOG) during ship path following. The blue line represents the true SOG, while the red line represents the SOG estimated by the Extended Kalman Filter.

Figure 4.11 reports the commanded and actual rudder angles during the same maneuver. The largest variations occur near waypoint transitions, where the vessel has to modify its course in order to follow the next path segment. The actual rudder angle follows the commanded signal while remaining within the admissible range imposed by the saturation block. This confirms that the rudder dynamics and the command conditioning blocks operate consistently with the expected behaviour of the steering system.

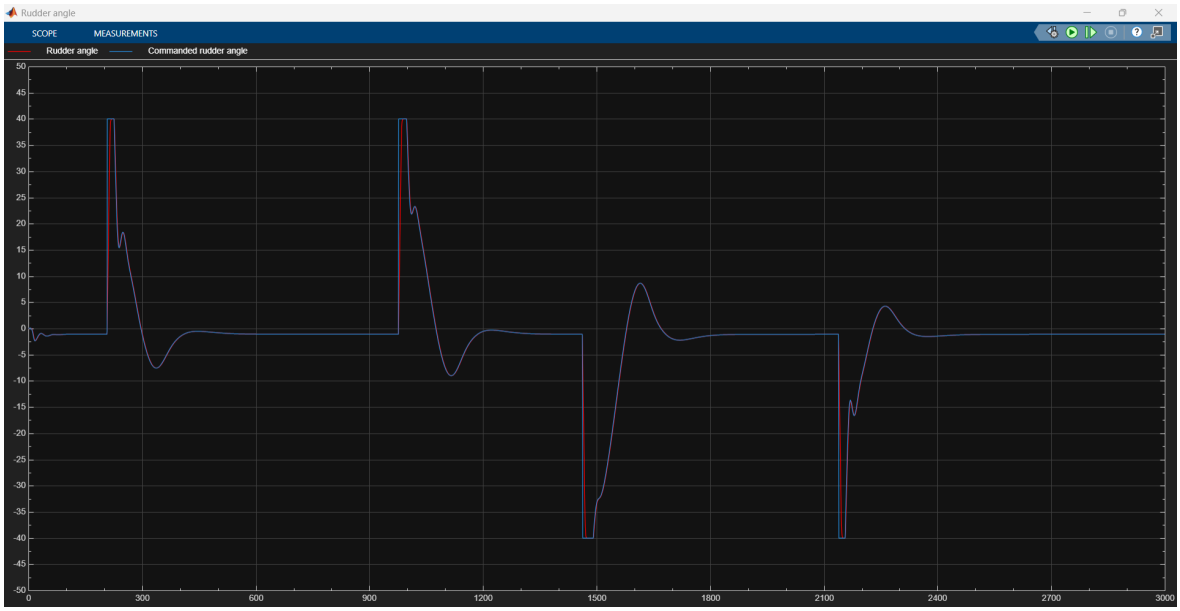


Figure 4.11: Commanded and actual rudder angles during waypoint transitions. The blue line represents the commanded rudder angle, while the red line represents the actual rudder angle of the ship.

The course response is shown in Figure 4.12. This plot compares the desired course over ground with the estimated course provided by the EKF. The estimated course follows the desired reference with limited transient deviations, mainly occurring when the vessel switches from one path segment to the next. This behaviour is expected, since the ship cannot change direction instantaneously and the course reference is affected by the smoothing action of the observer and by the vessel dynamics.



Figure 4.12: Desired, actual and estimated COG during ship path-following manoeuvres. The blue line represents the desired COG, the red line represents the actual COG of the ship and the yellow line represents the COG estimated by the Extended Kalman Filter.

Overall, the simulation results confirm the correct operation of the standalone maritime path-following model. The vessel is able to follow the prescribed waypoint sequence, the EKF provides reliable estimates of the main navigation variables and the rudder response is consistent with the required course changes.

This preliminary validation represents an important step before the integration of the ship model into the complete cooperative architecture involving the UAV swarm and the communication layer.

Chapter 5

Integrated Multidomain Model

The UAV swarm model and the cargo ship model described in the previous chapters were initially developed and analysed as separate simulation architectures. This separation was useful to understand the behaviour of each subsystem independently and to verify the correct operation of the main guidance, control, estimation and visualisation blocks.

However, the objective of this thesis is not limited to the independent simulation of the aerial and maritime platforms. The main goal is to create a cooperative multidomain scenario in which the UAV swarm and the cargo ship are included within the same simulation framework and are able to exchange information during the mission. For this reason, a new Simulink model was developed by combining the two previously described models and extending them with additional communication and coordination functionalities.

In this multidomain architecture, the UAV subsystem and the maritime subsystem are not treated as isolated models. Instead, they are connected through message based communication blocks, so that relevant information can be exchanged between these two platforms during the simulation. In particular, the UAVs can communicate mission related information to the ship, such as the verification of safe points, while the ship position can be made available to the UAVs for supervision and return-to-ship logic.

To obtain this behaviour, the original models were modified by adding dedicated functions, supervisory logic and communication interfaces. These additions allow the simulation to represent not only the motion of the vehicles, but also the flow of information between them. As a result, the multidomain model provides a more complete framework for analysing cooperation between heterogeneous autonomous platforms operating in the same scenario.

5.1 Integration of Message-Based Communication in the UAV Swarm Model

The UAV swarm model was modified by extending the original multi-UAV waypoint-following architecture into a cooperative framework, as shown in Figure 5.1. The updated model includes dynamic mission management, communication with the ship and return-to-ship logic. These additions allow the UAV swarm to adapt its behaviour during the mission instead of following only a fixed sequence of waypoints.

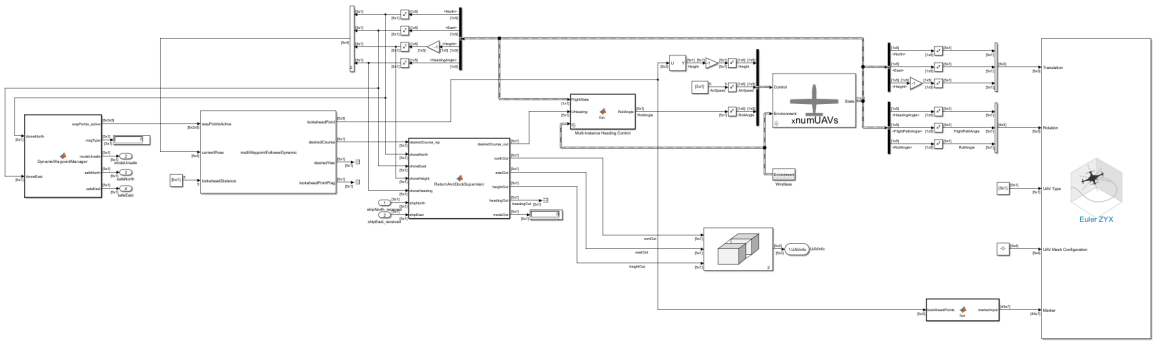


Figure 5.1: Modified UAV swarm architecture with communication and mission management functionalities.

The main modification is the introduction of the `DynamicWaypointManager` block before the Waypoint Follower, as shown in Figure 5.2. In the original configuration, the UAVs followed a fixed waypoint sequence, defined before the start of the simulation and not modified during the mission. As a consequence, each vehicle simply tracked the assigned waypoints without reacting to the possible presence of unsafe regions along the route.

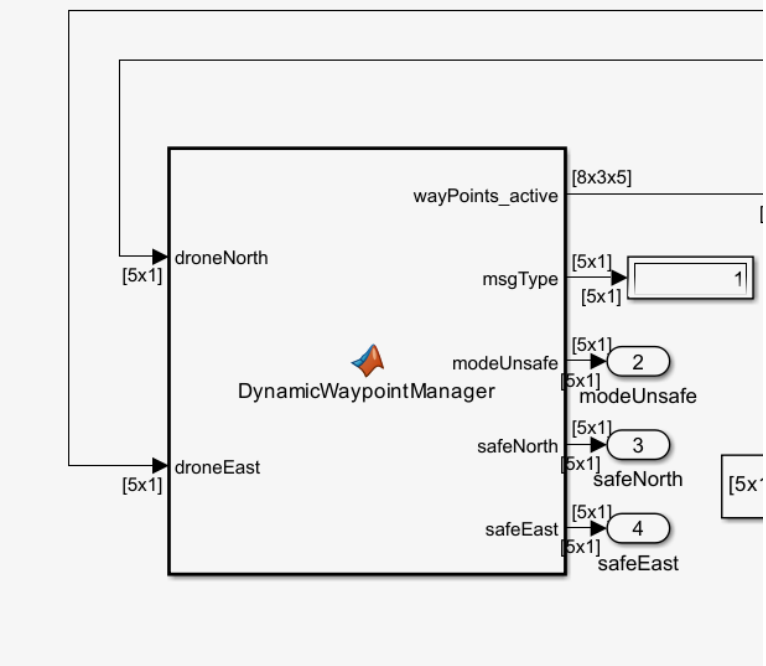


Figure 5.2: Simulink block `DynamicWaypointManager`.

In the modified model, the waypoint sequence is no longer treated as a purely fixed reference defined before the beginning of the simulation. Instead, the active trajectory is updated according to the mission state and to the current position of each UAV. This modification introduces a higher level of mission awareness into the waypoint management logic, since the UAVs are not only required to follow a prescribed path, but also to react to specific events occurring along the nominal route.

In particular, two waypoints of the original mission, denoted as *WP3* and *WP5*, are treated as potentially unsafe locations. These points represent critical areas along the planned route of the maritime platform, where an unsafe condition is assumed to be detected during the mission. When a UAV reaches the neighborhood of one of these locations, the waypoint manager modifies its active reference sequence and redirects the vehicle towards a corresponding candidate safe point. The candidate safe locations, denoted as *SAFE1* and *SAFE2*, represent alternative areas that may be considered suitable for the subsequent adaptation of the ship trajectory.

In the present Simulink implementation, the candidate safe points are defined a priori. This modelling choice allows the cooperative logic to be tested in a controlled and repeatable simulation environment, without introducing an additional online search or optimization layer. In a more general operational scenario, the UAVs could be required to explore the surrounding area of the unsafe region in order to identify suitable safe locations autonomously. However, for the purposes of this implementation, the safe

points are assumed to be known candidate locations, so that the focus can be placed on the interaction between detection, validation, communication and rerouting.

Although *SAFE1* and *SAFE2* are predefined in the model, they are not automatically accepted by the supervisory logic. Each candidate safe point must first be reached by the UAV and validated through a verification radius. This means that the point becomes available for the ship rerouting logic only after the UAV has physically reached the corresponding area and confirmed it within the simulation framework. In this way, the model preserves the essential sequence of the cooperative decision process: the UAV detects the unsafe region, moves towards the candidate safe point, verifies it and then provides the relevant information to the maritime subsystem.

Once the validation condition is satisfied, the UAV can transmit the corresponding information to the ship through the communication architecture. The maritime platform can then use the received data to update its own waypoint sequence and adapt its trajectory accordingly. Therefore, the `DynamicWaypointManager` block represents a key element of the proposed architecture, since it connects the local mission state of the UAVs with the higher level cooperative rerouting strategy of the ship.

The block is placed before the waypoint-following subsystem because the active trajectory must be updated before the next look-ahead point and the corresponding guidance commands are computed. The complete MATLAB implementation of the `DynamicWaypointManager` function is reported in Appendix A.1, Listing A.1. Since this function was developed specifically for the proposed cooperative architecture, the appendix includes the full code, while this section focuses on its functional role within the mission logic.

The system operates according to a mode-based logic. The persistent variable *mode* stores the current operating condition of each UAV during the simulation, so that the mission state is preserved from one simulation step to the next. The variables *sentSafe1* and *sentSafe2* are also persistent and are used to avoid sending the same safe point to the ship more than once.

```
1 persistent mode
2 persistent sentSafe1
3 persistent sentSafe2
```

Code 5.1: Persistent variables used to store the mission state.

The mode-based logic defines a finite set of operating conditions, each associated with a specific phase of the UAV mission. At each simulation step, the current value of *mode* determines which waypoint sequence must be followed and whether the UAV has

to continue along the nominal route, move towards a candidate safe point or resume the mission after validation. Accordingly, the transition between nominal navigation, unsafe waypoint detection, safe point verification and mission continuation is managed explicitly within the `DynamicWaypointManager` function.

The operating modes are defined as follows:

- `mode = 0`: the UAV follows the nominal waypoint sequence;
- `mode = 1`: the first unsafe waypoint *WP3* has been detected and the UAV is redirected towards *SAFE1*;
- `mode = 2`: *SAFE1* has been reached, verified and the UAV continues the mission along the updated trajectory;
- `mode = 3`: the second unsafe waypoint *WP5* has been detected and the UAV is redirected towards *SAFE2*;
- `mode = 4`: *SAFE2* has been reached and verified.

The transition between these modes is based on distance thresholds. The detection radii $R_{\text{detect},1}$ and $R_{\text{detect},2}$ are set to 60 m, while the verification radii $R_{\text{safe},1}$ and $R_{\text{safe},2}$ are set to 40 m. These values were selected by considering the scale of the simulated mission. The waypoints are defined with relatively large separations, because the same operational scenario also includes the cargo ship, whose maneuvering capability is much more limited than that of the UAVs. Unlike an aerial vehicle, the ship cannot change direction quickly and therefore the reference path must avoid very short segments or abrupt consecutive turns.

For this reason, a detection radius of 60 m is small with respect to the overall spacing of the mission waypoints, but it is still large enough to activate the rerouting logic before the UAV reaches the exact unsafe location. This avoids relying on a perfectly exact waypoint crossing, which would be unrealistic in a dynamic simulation.

The verification radius is instead set to 40 m because the confirmation of a safe point must be more selective than the detection of an unsafe region. In this case, the UAV is required to approach the safe point more closely before the point is considered verified and communicated to the ship. This distinction allows the model to combine an early detection of critical areas with a more precise confirmation of the safe locations.

For example, the first unsafe waypoint is detected when the distance from *WP3* becomes smaller than the detection radius. At that moment, the operating mode changes from nominal navigation to the first rerouting phase:

```
if mode(i) == 0 && d_wp3 <= R_detect1
mode(i) = 1;
end
```

Code 5.2: Example of transition from nominal navigation to the first rerouting phase.

Similarly, *SAFE1* is considered verified only when the UAV reaches the verification area around it:

```
if mode(i) == 1 && d_safe1 <= R_safe1
mode(i) = 2;
end
```

Code 5.3: Example of safe point verification.

Based on the current value of *mode*, the function selects the waypoint sequence assigned to each UAV. When the UAV is still in nominal navigation, the original waypoint sequence is used. Once an unsafe point has been detected, the alternative sequence including the safe points is activated.

```
if mode(i) == 0
wayPoints_active(:,:,i) = WP_nom;
else
wayPoints_active(:,:,i) = WP_safe_complete;
end
```

Code 5.4: Selection of the active waypoint sequence.

As a result, the value of *mode* acts as an internal mission-state indicator, determining which waypoint sequence is active at each simulation step and ensuring the correct transition between nominal navigation, safe point verification and mission continuation.

Since *mode* is an internal persistent variable of the `DynamicWaypointManager` function, its value is exported to the Simulink model through the output *modeUnsafe*. This output provides the surrounding subsystems with a compact representation of the current UAV mission state, enabling its use in the communication and rerouting logic.

In addition to the waypoint update logic, the function also generates auxiliary communication-related outputs. Among these, the variable *msgType* is introduced to classify the message condition generated by the UAV logic. In the current implementation, this variable is mainly used for monitoring and validation purposes, since it allows the communication condition to be inspected during the simulation. For this reason, *msgType* is visualized through a Display block, as shown in Figure 5.2, in order to verify

when a normal communication condition is active and when a verified safe point event is generated.

Only two values of *msgType* are considered:

- `msgType = 1`: this value represents the default communication condition. It is assigned at each simulation step when no newly verified safe point is generated by the UAV logic;
- `msgType = 3`: this value is assigned when a candidate safe point has been reached and verified by the UAV. In particular, it is generated when either *SAFE1* or *SAFE2* becomes available and has not already been communicated.

The persistent variables *sentSafe1* and *sentSafe2* ensure that each verified safe point is marked as communicated only once. Once the corresponding flag is set to true, the same safe point does not generate a new `msgType = 3` condition at every simulation step. This mechanism avoids repeated event generation and supports the validation of the communication logic.

```
if mode(i) == 2 && sentSafe1(i)
msgType(i) = 3;
safeNorth(i) = SAFE1(1);
safeEast(i)  = SAFE1(2);
sentSafe1(i) = true;
end
```

Code 5.5: Example of safe point communication logic.

In addition to the waypoint update logic, the UAV subsystem was extended to make the position of the aerial vehicles available to the communication layer. For this purpose, the relevant UAV state information is collected in the *UAVinfo* signal, which contains the position data required by the rest of the multidomain model.

The return to ship logic is managed by a separate supervisory block, implemented through the `ReturnAndDockSupervisor` function. This function controls the final phase of the UAV mission after the exploration and safe point verification stages. Its role is to determine when each UAV has completed its assigned trajectory, redirect it towards the ship and represent the final recovery phase on the ship deck.

Figure 5.3 shows the `ReturnAndDockSupervisor` block used in the Simulink model. The block receives the desired course generated by the waypoint follower, the current UAV position, heading and the ship position received through the communication layer.

Its outputs provide the updated desired course, the updated UAV position variables and the current mission mode of each UAV.

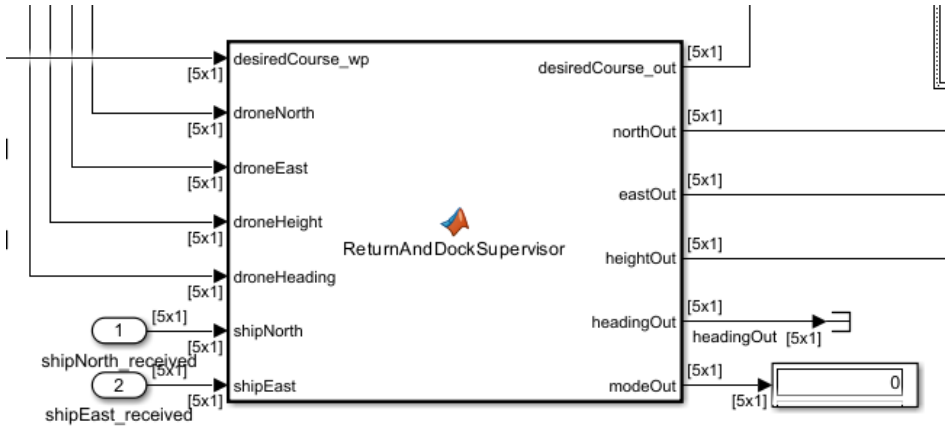


Figure 5.3: Simulink block `ReturnAndDockSupervisor`.

The complete MATLAB implementation of the `ReturnAndDockSupervisor` function is reported in Appendix A.2, Listing A.2.

The function uses a mode based logic for each UAV. The current phase of each vehicle is stored in the persistent variable *mode*, so that the state of the mission is preserved during the simulation:

```
1 persistent mode
```

Code 5.6: Persistent variable used to store the return mode of each UAV.

The operating logic is divided into three main modes, each corresponding to a specific phase of the return-and-docking procedure:

- `mode = 0`: this mode corresponds to the nominal mission phase. The UAV is still following the assigned waypoint sequence and the guidance command is provided by the waypoint-following subsystem. In this condition, no return-to-ship action is active and the desired course remains the one computed from the current waypoint tracking logic;
- `mode = 1`: this mode is activated when the UAV has completed its assigned path and must return towards the maritime platform. The guidance objective is therefore switched from waypoint tracking to ship tracking. In this phase, the desired course is computed with respect to the current ship position, so that the UAV is progressively guided towards the moving platform;

- `mode = 2`: this mode represents the capture or docking phase. Once the UAV enters the predefined capture region around the ship, it is considered to have reached the maritime platform. Its position is then constrained to the ship deck, allowing the simulation to represent the final docking condition and preventing the UAV from continuing independent flight after capture.

During the nominal mission phase, the supervisor evaluates the distance between each UAV and the final mission waypoint. When the UAV enters a radius R_{end} around this final point, the mission path is considered completed and the return phase is activated:

```
if mode(i) == 0 && d_end <= R_end
mode(i) = 1;
end
```

Code 5.7: Transition from nominal mission to return phase.

The parameter R_{end} is set to 60 m. This value avoids requiring the UAV to reach the final waypoint exactly, which would not be suitable for the simulated mission conditions. Instead, the vehicle is considered to have completed the trajectory when it enters a reasonable neighborhood of the final point.

During the return phase, the desired course is no longer provided by the `Waypoint Follower`. Instead, it is computed directly from the relative position between each UAV and the ship. For the i-th UAV, the North and East position errors with respect to the maritime platform are defined as

$$\Delta N_i = N_{\text{ship}} - N_{\text{UAV},i}, \qquad \Delta E_i = E_{\text{ship}} - E_{\text{UAV},i}.$$

The return course reference is then obtained as

$$\chi_{\text{return},i} = \operatorname{atan2}\left(\Delta E_i, \Delta N_i\right).$$

This formulation generates a course reference directed from the UAV towards the ship. The use of the `atan2` operator allows the correct quadrant of the angle to be determined from the relative North and East components, providing a consistent course reference over the entire operating area.

```
desiredCourse_out(i) = atan2(shipEast_i - droneEast(i), ...
shipNorth_i - droneNorth(i));
```

Code 5.8: Representative computation of the return course towards the ship.

Since the ship position is received through the communication layer and updated during the simulation, the reference is recomputed according to the instantaneous relative geometry between the UAV and the maritime platform. Therefore, the UAV does not return to a fixed predefined point, but continuously adapts its course to the moving ship.

The transition to the capture phase occurs when the distance between the UAV and the ship becomes smaller than R_{capture}. In this case, the UAV is considered to have reached the ship deck:

```
if mode(i) == 1 && d_ship <= R_capture
mode(i) = 2;
end
```

Code 5.9: Transition from return phase to capture condition.

The capture radius R_{capture} is set to 85 m. This threshold is selected larger than R_{end}, since the capture condition is evaluated with respect to a moving target rather than to a fixed waypoint. During the simulation, the ship continues to move while the UAV is returning; therefore, imposing an exact coincidence between the UAV position and the ship coordinates would be unrealistic and numerically too restrictive. A wider capture radius provides a more robust recovery criterion, allowing the UAV to be considered captured once it enters a suitable neighborhood of the maritime platform.

The selected value is also consistent with the physical scale of the ship adopted in the maritime model. The MSS Mariner class cargo vessel used in the simulation has an overall length of approximately 160 m. Consequently, a capture radius of 85 m is comparable with the characteristic dimensions of the vessel and can be interpreted as a reasonable recovery region around the ship. This choice avoids modelling the ship as a single geometric point and provides a practical approximation of the final approach area, while remaining compatible with the simplified docking representation implemented in the Simulink model.

Once the UAV enters the capture mode, its North and East positions are set equal to the current ship position, while its height is set to the deck height:

```
northOut(i)  = shipNorth_i;
eastOut(i)   = shipEast_i;
heightOut(i) = deckHeight;
```

Code 5.10: Representative assignment of the UAV position during capture.

The output *modeOut* provides the current return mode of each UAV to the rest of the

model. This makes it possible to distinguish between UAVs that are still following the mission path, UAVs that are returning to the ship and UAVs that have already been captured.

Overall, the `ReturnAndDockSupervisor` function extends the UAV model by adding three main functionalities: mission completion detection, return to ship guidance and a simplified representation of the docking phase. Together with the `DynamicWaypoint-Manager` function and the message based communication signals exchanged with the ship, this block allows the UAV swarm to operate within the multidomain cooperative scenario, rather than as a set of independent waypoint following vehicles.

The core components of the original UAV model remain substantially unchanged. In particular, the `Heading Controller`, the `Multi-instance Guidance Model` block `xnumUAVs`, the `Windless` environmental subsystem and the `UAV Animation` block are preserved from the reference architecture. The main contribution of the modified model is therefore not the replacement of the basic flight guidance structure, but its extension with mission supervision and communication logic.

As a result, the second architecture evolves from a standard path following simulation into a cooperative simulation framework. In this configuration, the UAV swarm can interact with the maritime platform, update its behaviour according to the mission state, communicate verified safe points and return towards the ship at the end of the mission.

5.2 Integration of Message-Based Communication in the Mariner Cargo Ship Model

After introducing the communication logic on the UAV side, the Mariner class cargo ship model was also extended in order to react to the information received from the aerial swarm. In the original ship model, the reference path is defined only by a fixed sequence of waypoints provided to the `LOS guidance` block. Under this configuration, the vessel behaves as an independent platform: it follows the assigned route according to the LOS guidance law, without considering external information generated by other agents operating in the same environment.

In the proposed multidomain architecture, this structure is modified by introducing an additional decision layer between the original waypoint definition and the LOS guidance subsystem. The purpose of this layer is to make the maritime trajectory dependent on the information exchanged with the UAV subsystem, while preserving the original

path-following structure of the ship model. The main modification consists in the introduction of the `ShipWaypointManager` block, shown in Figure 5.4, which manages the waypoint sequence provided to the guidance block according to the safe point information received from the UAV leader.

The resulting ship trajectory is therefore no longer limited to the initial predefined route. Instead, selected portions of the path can be replaced when a candidate safe point has been verified by the UAV and transmitted to the maritime platform through the communication layer. This allows the ship to adapt its route in response to information collected by the aerial subsystem, while maintaining a deterministic and controlled rerouting logic.

It is important to observe that the route modification is not performed autonomously by the ship in isolation. The maritime platform does not generate new safe locations by itself, nor does it continuously optimize its trajectory online. Rather, the update is triggered only when validated information is received from the UAV leader. In this way, the rerouting process remains explicitly linked to the cooperative mission logic: the UAV detects and verifies a candidate safe point, the information is communicated to the ship and the ship waypoint sequence is then updated accordingly.

This modification allows the information collected by the UAVs to influence the path followed by the ship. The `ShipWaypointManager` block acts as an interface between the communication architecture and the LOS-based ship guidance system, translating the received safe point information into an updated waypoint sequence. As a consequence, the ship model is able to preserve the validated guidance and control structure of the original MSS implementation, while becoming responsive to the cooperative information flow generated within the multidomain simulation framework.

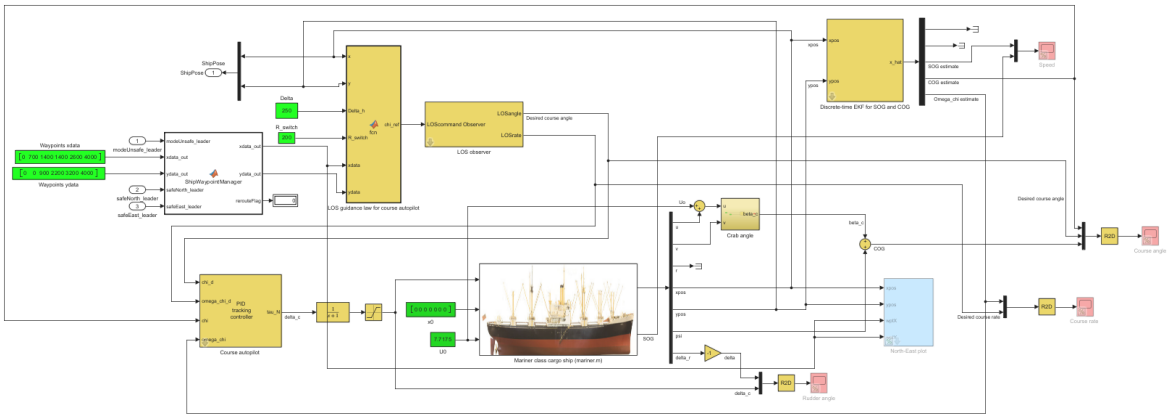


Figure 5.4: Modified Mariner cargo ship model used for message based communication with the UAV swarm.

The complete MATLAB implementation of the `ShipWaypointManager` function is reported in Appendix A.3, Listing A.3.

The block receives as inputs the signal *modeUnsafe_leader*, which represents the current mission state of the UAV leader and the coordinates *safeNorth_leader* and *safeEast_leader*, corresponding to the safe point communicated by the UAV after verification. In addition, the inputs *waypoint_x_data* and *waypoint_y_data* define the nominal ship trajectory through the coordinates of the reference waypoints. These waypoints constitute the original path provided to the LOS guidance subsystem. However, when a safe point is validated by the UAV leader, the `ShipWaypointManager` block can update the waypoint sequence by replacing the corresponding unsafe portion of the route with the received safe location.

Within the proposed simulation framework, the leader is selected as the first UAV of the swarm, but this choice is not fixed by the logic itself and could be modified if a different communication strategy were adopted.

As described in the previous chapter, the safe points are defined in the simulation, but they are not directly used by the ship from the beginning. They are considered valid for rerouting only after the UAV leader has reached and verified them. Therefore, the ship modifies its trajectory only after receiving confirmed information from the aerial platform.

The rerouting logic is based on the persistent variable *rerouteStage*, which stores the current rerouting condition of the ship during the simulation. This memory is necessary because the ship must preserve the information about the safe points already verified by the UAV leader and must not return to the initial route at the next simulation step.

```
1 persistent rerouteStage
2 persistent safe1_North safe1_East
3 persistent safe2_North safe2_East
```

Code 5.11: Persistent variables used to store the ship rerouting state.

The variable *rerouteStage* can assume three main values:

- *rerouteStage = 0*: no safe point has been used yet and the ship follows the nominal route;
- *rerouteStage = 1*: the first safe point has been verified by the UAV leader and the ship route is updated through *SAFE1*;
- *rerouteStage = 2*: both safe points have been verified and the ship route is

updated through *SAFE1* and *SAFE2*.

The first rerouting condition is activated when *modeUnsafe_leader* becomes greater than or equal to 2. This means that the UAV leader has reached and verified the first safe point. At this moment, the coordinates received from the UAV are stored and the rerouting stage is updated:

```
if modeUnsafe_leader >= 2 && rerouteStage < 1
rerouteStage = 1;
safe1_North = safeNorth_leader;
safe1_East  = safeEast_leader;
end
```

Code 5.12: Activation of the first ship rerouting stage.

Similarly, the second rerouting condition is activated when *modeUnsafe_leader* becomes greater than or equal to 4. This condition indicates that the second safe point has also been verified by the UAV leader. The new safe coordinates are then stored and the rerouting stage is increased:

```
if modeUnsafe_leader >= 4 && rerouteStage < 2
rerouteStage = 2;
safe2_North = safeNorth_leader;
safe2_East  = safeEast_leader;
end
```

Code 5.13: Activation of the second ship rerouting stage.

In this implementation, two unsafe points were introduced as a design choice in order to verify the correct behaviour of the rerouting logic of the ship. This number is not a fixed limitation of the general approach: depending on the mission scenario the number of unsafe areas and corresponding safe points could be increased or reduced. In the proposed work, the use of two unsafe points was considered sufficient to test the progressive update of the ship trajectory and to show how the vessel can modify its path after receiving verified information from the UAV swarm.

Once a safe point has been verified, the corresponding ship waypoint is replaced by the safe coordinates received from the UAV. In particular, after the verification of *SAFE1*, the third waypoint of the ship path is replaced, while after the verification of *SAFE2*, the fifth waypoint is replaced:

```
if rerouteStage >= 1
xdata_out(3) = safe1_North;
```

```
ydata_out(3) = safe1_East;
end

if rerouteStage >= 2
xdata_out(5) = safe2_North;
ydata_out(5) = safe2_East;
end
```

Code 5.14: Update of the ship waypoint sequence using the verified safe points.

The output waypoint vectors *xdata_out* and *ydata_out* are then sent to the `LOS guidance` block of the ship. Therefore, the reference path followed by the Mariner vessel is updated according to the validated information received from the UAV swarm. The ship trajectory is therefore not changed simply because alternative safe points exist in the model, but only after the UAV leader has confirmed them and communicated their coordinates.

The block also provides the output *rerouteFlag*, which is set equal to the current value of *rerouteStage*. While *rerouteStage* is used internally by the `ShipWaypointManager` function to preserve the rerouting state during the simulation, *rerouteFlag* makes this information available outside the block. Therefore, it acts as an external indicator of the route adaptation stage reached by the ship within the integrated Simulink model.

Overall, the `ShipWaypointManager` block introduces a communication based rerouting mechanism in the Mariner cargo ship model. Together with the UAV mission management logic, it allows the maritime platform to adapt its path according to the information verified by the aerial swarm, making the ship model part of the multidomain cooperative scenario.

The updated model also includes the output *ShipPose*, which provides the current position of the ship. This information can be transmitted back to the UAV subsystem, allowing the drones to know the updated location of the maritime platform.

This is particularly important for the return and docking phase, since the UAVs need the ship position in order to return to the vessel after completing their mission.

Furthermore, the ship trajectory and some guidance parameters were also adapted to the new cooperative scenario.

The LOS guidance parameters were also tuned to match the spatial scale of the cooperative UAV-ship scenario. In particular, the look-ahead distance *Delta* was reduced from 500 m to 250 m, while the waypoint switching radius R_switch was reduced from 400 m to 200 m. This modification does not change the physical ship model or its

dynamic behaviour, it only adapts the guidance logic to a more compact operational area, consistent with the trajectory followed by the UAV swarm.

These reduced values allow the vessel to follow the reference route with a level of responsiveness more appropriate for the simulated multidomain mission. Since both parameters were scaled in the same proportion, the overall LOS guidance behaviour is preserved, while the path-following logic becomes better matched to the UAV trajectory scale.

The ship is now able to receive information from the swarm, process verified safe point coordinates, update its own route and share its position with the UAVs. This modification transforms the maritime model from an isolated path-following system into an integrated multidomain simulation framework based on information exchange and coordinated decision making.

5.3 MultiDomain Integration Model

The final Simulink architecture, named `multidomain`, represents the complete integration of the two models developed in the previous parts of this thesis: the Mariner cargo ship model and the UAV swarm model. These models were initially implemented and tested separately in order to verify the behaviour of each subsystem independently. The ship model was used to simulate the motion of the maritime platform along a predefined path, while the UAV swarm model was used to represent the coordinated flight of multiple aerial vehicles following their assigned waypoint sequences.

The purpose of the `multidomain` model is to combine these two simulation environments into a single top level architecture. This integration provides a common simulation framework in which the evolution of the ship and the UAV swarm can be evaluated at the same time, while preserving the internal structure of the two original subsystems.

In particular, the `multidomain` architecture introduces communication links between the two models. Through these links, the UAVs can transmit mission related information, such as the verification of safe points, while the ship can use this information to update its reference trajectory. At the same time, the ship position can be made available to the UAVs, enabling return to ship and docking logic at the end of the aerial mission.

Therefore, the final model extends the original standalone simulations into a cooperative multidomain framework. The resulting architecture allows the UAV swarm and the Mariner cargo ship to interact during the simulation, modifying their behaviour

according to the mission state and to the information exchanged between the two domains.

From a structural point of view, the `multidomain` model is composed of two main subsystems: the ship subsystem and the UAV swarm subsystem. As shown in Figure 5.5, these two parts are connected through two dedicated communication interfaces, named `ShipToDrone_Comm` and `DroneToShip_Comm`. The first interface transfers information from the maritime platform to the aerial vehicles, while the second one sends mission related information from the UAV swarm to the ship.

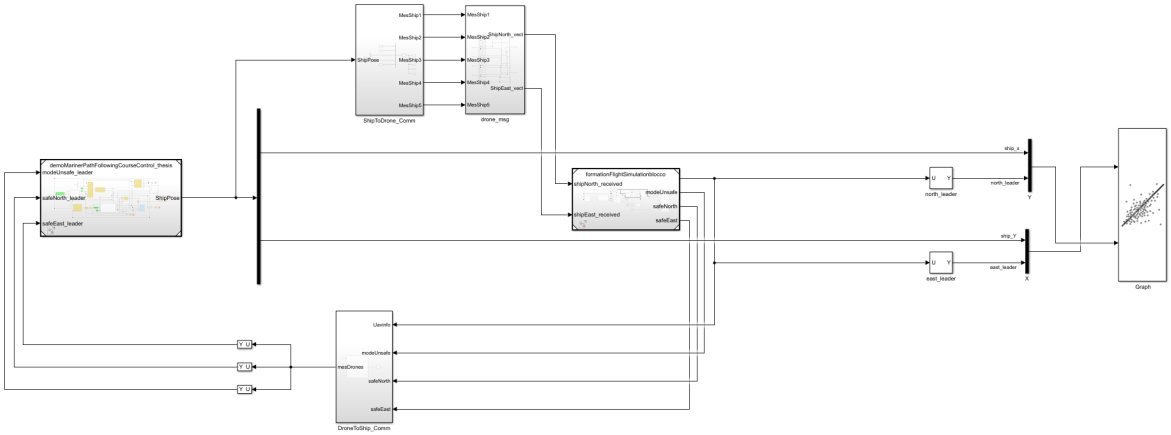


Figure 5.5: `Multidomain` Simulink model.

The architecture shown in Figure 5.5 highlights the separation between the two physical domains and the communication layer introduced to connect them. Both subsystems preserve the main structure of the models described in the previous sections. However, they were extended with additional communication and mission management logic in order to exchange information during the simulation and to support the cooperative behaviour of the multidomain architecture. The additional communication blocks are placed between these two subsystems and are used to exchange only the variables required by the cooperative mission logic.

Signal selectors and bus management blocks are used to extract the required variables from each subsystem and route them towards the corresponding communication interface. This organization avoids sending the complete set of internal variables from one subsystem to the other. Instead, only the signals required for coordination are selected and transmitted.

In the ship to drone direction, the ship subsystem provides its current position through the *ShipPose* output. This information is selected, processed and transmitted to the UAV swarm through the `ShipToDrone_Comm` interface.

On the UAV side, the received position is made available through the signals *shipNorth_received* and *shipEast_received*. These signals represent the updated North and East coordinates of the cargo ship and are used by the UAV supervisory logic during the final phase of the mission. In particular, they allow the UAVs to identify the current position of the ship and to use it as the moving target for the return to ship and docking logic.

In the drone to ship direction, the UAV swarm transmits information related to the mission state and safe point verification. The relevant signals are *modeUnsafe*, *safeNorth* and *safeEast*. The signal *modeUnsafe* describes the current state of the UAV mission management logic, while *safeNorth* and *safeEast* contain the coordinates of the safe point verified by the UAVs.

Since the rerouting strategy implemented in this thesis does not require the ship to process the complete state information of all UAVs, the communication architecture is based on a leader based approach. In the present implementation, the first UAV of the swarm is selected as the leader and its transmitted message is used by the maritime platform as the reference input for the route adaptation logic. This choice provides a deterministic decision making mechanism and avoids ambiguities in the trajectory update process of the ship.

In the considered simulation scenario, all UAVs follow the same mission logic and are assigned to verify the predefined critical areas and the corresponding candidate safe points. However, the implemented MATLAB/Simulink architecture is not intrinsically limited to this configuration. The multi-UAV model and the associated waypoint management structure can be extended to assign independent waypoint sequences to the different vehicles, allowing each UAV to inspect a distinct region of the operational area and to provide the maritime platform with the information required for route assessment and possible trajectory adaptation.

Such an extension would require the introduction of a higher level coordination strategy. If multiple UAVs were allowed to provide different candidate safe locations or alternative route suggestions simultaneously, the maritime platform would need a specific information selection, prioritization or fusion criterion before updating its trajectory. Indeed, since the ship can physically follow only one route at a time, conflicting inputs from different UAVs could lead to an ambiguous or inconsistent rerouting decision if not properly managed.

For this reason, a leader based communication strategy was adopted to ensure a controlled and consistent validation of the proposed cooperative architecture. By selecting the first UAV as the representative agent of the swarm, the ship receives a single co-

herent set of information for the activation of the rerouting logic. This assumption makes it possible to focus on the integration between the aerial subsystem, the maritime subsystem, the communication structure and the route adaptation mechanism, while preserving the possibility of extending the framework in future developments towards distributed UAV trajectories, multi-agent information fusion and more advanced swarm coordination strategies.

In the present implementation, the leader UAV provides the signals *modeUnsafe_ leader*, *safeNorth_ leader* and *safeEast_ leader*. These signals are transmitted to the ship subsystem through the `DroneToShip_Comm` interface and are used by the `ShipWaypoint-Manager` block to activate the communication based rerouting strategy. When the leader UAV verifies a safe point, the corresponding coordinates are communicated to the ship, which can then update its waypoint sequence accordingly.

The model also includes signal selection and visualisation blocks used to monitor the cooperative behaviour of the system.

In particular, the positions of the ship and of the UAV leader are extracted and displayed on the graph, allowing the relative motion between the maritime platform and the aerial subsystem to be analyzed during the simulation.

Overall, the multidomain model enables bidirectional information exchange: the ship provides its position to support UAV recovery, while the UAV swarm provides verified safe point information to support ship rerouting. This structure forms the basis for the cooperative multidomain simulation framework developed in this work.

5.3.1 Uav to Ship Communication

The `DroneToShip_Comm` block manages the communication flow from the UAV swarm to the ship subsystem. As shown in Figure 5.6, this block collects the information generated by the UAV mission logic and organizes it into a structured message that can be used by the ship model.

In particular, the block transmits the mission state of the UAV leader and the coordinates of the verified safe points. These signals are then used by the ship subsystem to activate the waypoint update logic when validated information is received from the UAV swarm.



Figure 5.6: Simulink subsystem `DroneToShip_comm`.

The block receives four main inputs:

- *Uavinfo*, which contains the position information of the UAVs;
- *modeUnsafe*, which represents the current mission mode of each drone;
- *safeNorth* and *safeEast*, which are the coordinates associated with the verified safe points.

Inside the block, the MATLAB Function `BuildDroneMessages` builds the matrix *MesDrones*, which represents the message packet transmitted from the UAV swarm to the ship subsystem. The complete implementation of this function is reported in Appendix A.4, Listing A.4; in this section only the main logic is described.

The matrix *MesDrones* has seven rows and five columns. Each column is associated with one UAV of the swarm, while each row contains a specific type of information. This structure allows the information of all five UAVs to be grouped into a single compact communication matrix.

The function first initializes the output matrix as a zero matrix:

```
MesDrones = zeros(7,5);
```

Code 5.15: Initialization of the drone message matrix.

Then, a loop is used to fill one column of the matrix for each UAV. For the i-th UAV, the North and East coordinates are extracted from the signal *Uavinfo*. These values are then inserted into the corresponding message column together with the UAV identifier, the current mission mode and the safe point coordinates:

```
MesDrones(1,i) = 1;                % valid message
MesDrones(2,i) = i;                % sender identifier
MesDrones(3,i) = north_i;          % UAV North position
MesDrones(4,i) = east_i;           % UAV East position
MesDrones(5,i) = modeUnsafe(i);    % mission mode
MesDrones(6,i) = safeNorth(i);     % safe point North coordinate
MesDrones(7,i) = safeEast(i);      % safe point East coordinate
```

Code 5.16: Representative assignment of the message fields for each UAV.

The first row is a validity flag and is set equal to 1 to indicate that the message is active, meaning that the communication channel is assumed to be always available and that the transmitted message is correctly received by the ship subsystem. Therefore, possible communication impairments, such as packet losses, corrupted data or delays are not explicitly modelled at this stage.

This assumption allows the communication architecture to be tested under ideal transmission conditions, ensuring that the information exchange between the UAV leader and the maritime platform is always established. In this way, the focus is placed on the correctness of the message structure, the integration between subsystems and the activation of the rerouting logic once validated information is available. However, in a real operational scenario, the reliability of the communication link cannot be taken for granted and should be addressed through additional mechanisms for message validation, fault detection or resilient data exchange.

The second row contains the sender identifier, which corresponds to the UAV index. The third and fourth rows contain the current North and East coordinates of the UAV, extracted from *Uavinfo*. The fifth row contains the value of *modeUnsafe*, which describes the current mission condition of the UAV. Finally, the sixth and seventh rows contain the North and East coordinates of the safe point associated with that UAV.

Therefore, the output *MesDrones* provides a structured communication packet that includes both navigation information and mission status information for the whole

UAV swarm. This packet is then made available to the ship side of the model, where the relevant leader UAV information is selected and used for the communication based rerouting logic.

A unit delay block is placed after the MATLAB Function to introduce a discrete time delay in the transmitted message. This delay gives a simple representation of message based communication in discrete time and also avoids algebraic loops between the UAV and ship subsystems in the integrated Simulink model.

5.3.2 Ship to UAV Communication

The `ShipToDrone_Comm` block manages the communication flow from the ship subsystem to the UAV swarm. Its purpose is to transmit the current position of the maritime platform to the UAVs, so that each drone can use this information during the return to ship and docking phase.

As shown in Figure 5.7, the input of the block is the signal *ShipPose*, which contains the current position of the ship. From this signal, the North and East coordinates of the ship are extracted and inserted into a structured message. The message also includes a validity flag, a sender identifier and a message type field.

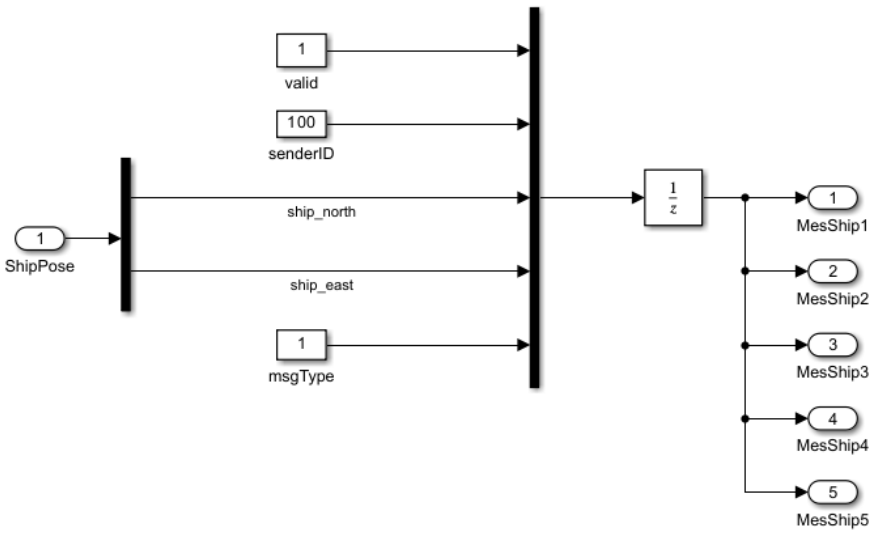


Figure 5.7: Simulink subsystem `ShipToDrone_comm`.

The validity flag is set equal to 1 to indicate that the ship message is active. Under this assumption, this communication channel is considered always available and the transmitted information is assumed to be correctly received by the UAV subsystem.

The sender identifier is used to distinguish the ship from the UAVs in the communication structure. In the proposed implementation, the value 100 is assigned to the ship, while the UAVs use their own numerical identifiers.

The message type field specifies that the transmitted content is a standard position update.

The generated command message is passed through a Unit Delay before being provided to the UAV subsystem. This choice allows the command sent by the ship to be updated at the next simulation instant, rather than being applied instantaneously within the same time step. As a result, the exchange of information is treated as a sequential process between the maritime and aerial subsystems, which is more consistent with the adopted discrete time simulation structure. The presence of the delay also improves the numerical robustness of the integrated model by avoiding instantaneous dependencies between interconnected signals.

The same ship message is then distributed to the five UAVs through the outputs *MesShip1*, *MesShip2*, *MesShip3*, *MesShip4* and *MesShip5*. Consequently, each UAV has access to the updated ship position required by the supervisory and return to ship logic.

The received ship messages are then processed inside the UAV side of the communication architecture by the `drone_msg` block, as shown in Figure 5.8. This block extracts the information required by the UAV swarm from the messages received from the ship. Each input, from *MesShip1* to *MesShip5*, corresponds to the ship message received by one UAV. Since the same ship message is transmitted to all drones, each message contains the same information fields: validity flag, sender identifier, ship North position, ship East position and message type.

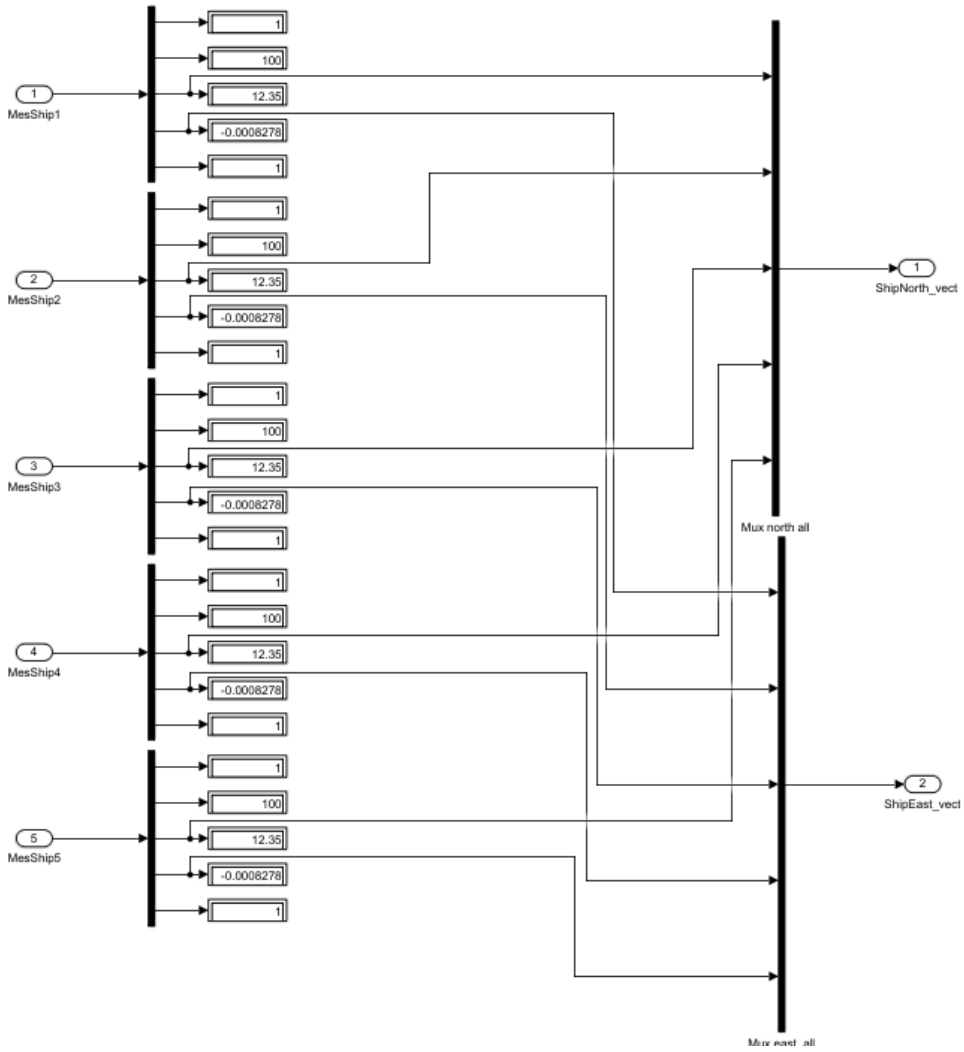


Figure 5.8: Simulink subsystem `drone_msg`.

The demultiplexer blocks separate the individual components of each received message.

In particular, the ship North and East coordinates are extracted from each message and then collected through mux blocks to build two vectors: *ShipNorth_ vect* and *ShipEast_ vect*. These vectors provide the ship position in a format compatible with the multi-instance structure of the UAV swarm model.

At the output of the `drone_msg` block, the vectors *ShipNorth_ vect* and *ShipEast_ vect* are sent to the `Formation Flight Simulation` block. As a result, the updated ship position becomes available inside the UAV swarm model and can be used by the supervisory logic during the return to ship phase.

This structure is consistent with the multi-instance organization of the UAV swarm model, where each UAV is associated with its own input signals. Even if the transmitted ship position is the same for all UAVs in the present implementation, the vector form makes the model more scalable and suitable for future extensions, such as different communication delays, packet losses or UAV specific received information.

This part of the model also provides a useful way to verify the correct operation of the communication structure. By observing the content of the received messages and the extracted vectors, it is possible to check that the ship position is correctly transmitted, received and made available to all UAVs during the simulation. Therefore, the `drone_msg` block is not only used for signal extraction, but also supports the validation of the integrated multidomain communication architecture.

Chapter 6

Simulation Results of the Integrated Multidomain Model

This chapter presents the results obtained from the complete multidomain simulation model developed in the MATLAB/Simulink environment. The purpose of the analysis is to verify the functional behaviour of the integrated architecture after combining the fixed-wing multi-UAV system, the Mariner class cargo ship model, the bidirectional information exchange structure and the supervisory functions introduced in the previous chapters.

The simulations evaluate the complete cooperative mission sequence under three different route geometries. In all the scenarios, the UAVs initially follow a waypoint sequence located ahead of the ship. When the UAV system approaches a predefined unsafe waypoint, the mission-management logic redirects the aerial vehicles towards the corresponding candidate safe point. The safe coordinates are considered valid only after the UAVs enter the associated verification radius.

Once verified, the coordinates of the safe point are transmitted to the maritime platform. The ship rerouting logic then replaces the corresponding unsafe waypoint with the received safe location and updates the reference route. After the route-inspection task has been completed, the ship transmits its current position to the UAVs, allowing them to compute a continuously updated return direction towards the moving maritime platform.

The simulations are intended to provide a functional verification of the implemented information flows, state transitions and waypoint-management logic. They do not represent an experimental validation of a real communication channel, an onboard hazard-detection system or a physical fixed-wing landing procedure.

6.1 Simulation Setup and Scenario Definition

All three simulations use the same mission-management architecture and the same detection and verification parameters. The unsafe locations and the associated candidate safe points are predefined in the simulation environment. Therefore, the term "detection" refers to the condition in which the UAV system enters the detection radius associated with a predefined unsafe waypoint, rather than to the output of an onboard perception algorithm.

The detection radius is set to

$$R_{\text{detect}} = 60 \text{ m}, \tag{6.1}$$

whereas the safe point verification radius is set to

$$R_{\text{safe}} = 40 \text{ m}. \tag{6.2}$$

In every scenario, `WP3` and `WP5` are associated with the two unsafe route segments. The additional waypoints `WP7` and `WP8` are set equal to `WP6`, so that the final part of the waypoint sequence remains fixed at the terminal location.

Table 6.1 summarises the waypoint coordinates, the safe point locations and the specific purpose of each simulation. All coordinates are expressed in the North-East-Down reference frame.

Table 6.1: Definition, purpose and observed outcome of the three simulation scenarios.

Case	**Nominal waypoint sequence**	**Unsafe and safe locations**	**Geometry, purpose and outcome**

1	`WP1`: $[0, 0, 0]$
	`WP2`: $[700, 0, -120]$
	`WP3`: $[1400, 900, -220]$
	`WP4`: $[1400, 2200, -160]$
	`WP5`: $[2600, 3200, -300]$
	`WP6`: $[4000, 4000, -140]$
		`UNSAFE1`: `WP3`
		`SAFE1`:
		$[1000, 900, -220]$
		`UNSAFE2`: `WP5`
		`SAFE2`:
		$[3000, 2800, -300]$
			Geometry: Reference trajectory with two separated route deviations.
			Purpose: Detailed verification of the complete mission sequence.
			Outcome: Both safe points are verified, two ship rerouting stages are activated and the final UAV capture condition is reached.

2	`WP1`: $[0, 0, 0]$
	`WP2`: $[900, 300, -120]$
	`WP3`: $[1800, 900, -220]$
	`WP4`: $[2500, 1700, -160]$
	`WP5`: $[3400, 2500, -300]$
	`WP6`: $[4700, 3300, -140]$
		`UNSAFE1`: `WP3`
		`SAFE1`:
		$[1500, 1300, -220]$
		`UNSAFE2`: `WP5`
		`SAFE2`:
		$[3800, 2200, -300]$
			Geometry: Oblique route with modified waypoint spacing and safe point directions.
			Purpose: Verification that the implemented logic is not restricted to the reference geometry.
			Outcome: The same communication, rerouting and return sequence is completed with the modified waypoint arrangement.

Case	Nominal waypoint sequence	Unsafe and safe locations	Geometry, purpose and outcome
3	WP1: $[0, 0, 0]$ WP2: $[900, 700, -120]$ WP3: $[1800, 1500, -220]$ WP4: $[3000, 2200, -160]$ WP5: $[3900, 1200, -300]$ WP6: $[6500, -500, -140]$	UNSAFE1: WP3 SAFE1: $[2000, 2300, -220]$ UNSAFE2: WP5 SAFE2: $[4300, 1900, -300]$	**Geometry:** Wider route variation with more pronounced changes in the maritime trajectory. **Purpose:** Evaluation of the waypoint-replacement logic under a more demanding route configuration. **Outcome:** Both rerouting stages remain feasible and the ship completes the adapted trajectory before the UAV return phase.

The common mission sequence is described in detail for the first simulation. The second and third cases are subsequently discussed by focusing on the differences introduced by their waypoint geometry and on the resulting maritime trajectory.

6.2 Reference Scenario

The first simulation is used as the reference case for the complete cooperative mission. It includes nominal waypoint following, the verification of two alternative safe locations, two consecutive modifications of the maritime route and the final return of the UAVs towards the moving ship.

6.2.1 Overall Mission Evolution

Figure 6.1 shows the complete trajectory evolution in the North-East plane. In the first simulation, the nominal UAV route is defined by the following waypoint sequence:
WP1 $= [0, 0, 0]$,

WP2 $= [700, 0, -120]$,

WP3 $= [1400, 900, -220]$,

WP4 $= [1400, 2200, -160]$,

$\texttt{WP5} = [2600, 3200, -300]$,

$\texttt{WP6} = [4000, 4000, -140]$.

The UAV system initially follows this nominal waypoint sequence ahead of the maritime platform. The aerial trajectory is subsequently modified in correspondence with the two predefined unsafe locations, namely `WP3` and `WP5`. The corresponding alternative safe locations are defined as $\texttt{SAFE1} = [1000, 900, -220]$ and $\texttt{SAFE2} = [3000, 2800, -300]$, respectively.

After the verification of `SAFE1` and `SAFE2`, their coordinates are transmitted to the ship. The maritime waypoint sequence is then updated by replacing `WP3` with `SAFE1` and `WP5` with `SAFE2`.

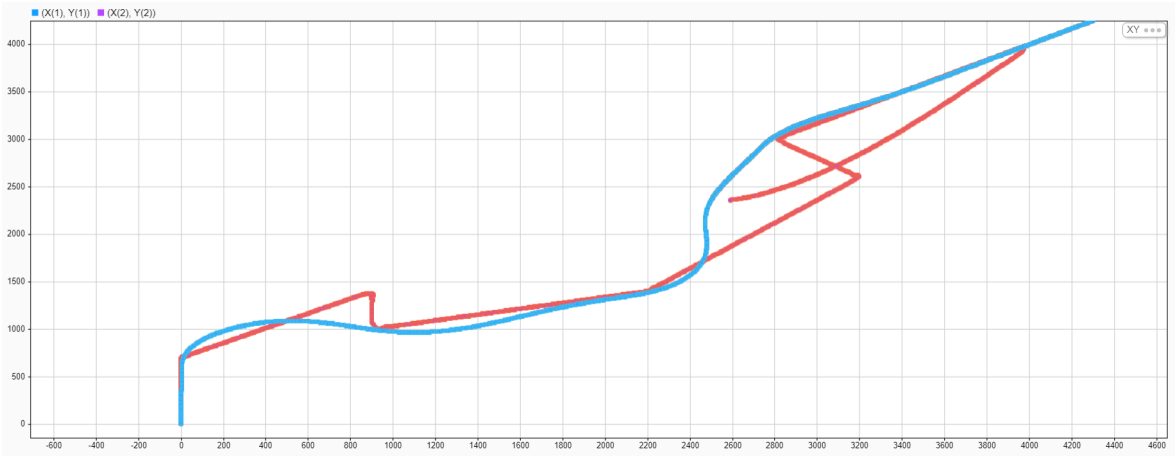


Figure 6.1: Overall trajectory evolution obtained from the first integrated simulation in the MATLAB/Simulink environment. The red trajectory represents the motion of the five UAVs, whereas the blue trajectory represents the ship path.

The overall result shows that the aerial and maritime subsystems do not evolve independently. The route followed by the ship depends on the information generated by the UAV mission-management logic, whereas the final UAV return direction depends on the continuously updated ship position.

6.2.2 Route Adaptation

The first route adaptation occurs when the UAV system enters the detection region associated with `WP3`, located at $[1400, 900, -220]$.

The `DynamicWaypointManager` redirects the UAVs towards `SAFE1`, whose coordinates are $[1000, 900, -220]$.

The safe location is not immediately communicated to the ship. It becomes valid only after the UAV system enters the corresponding verification radius. Once this condition is satisfied, the message validity field is activated and the verified coordinates are made available to the maritime subsystem.

Figure 6.2 shows the resulting trajectory modification. The ship does not react only to the presence of the unsafe waypoint. Instead, its reference route is modified after a valid alternative location has been received.

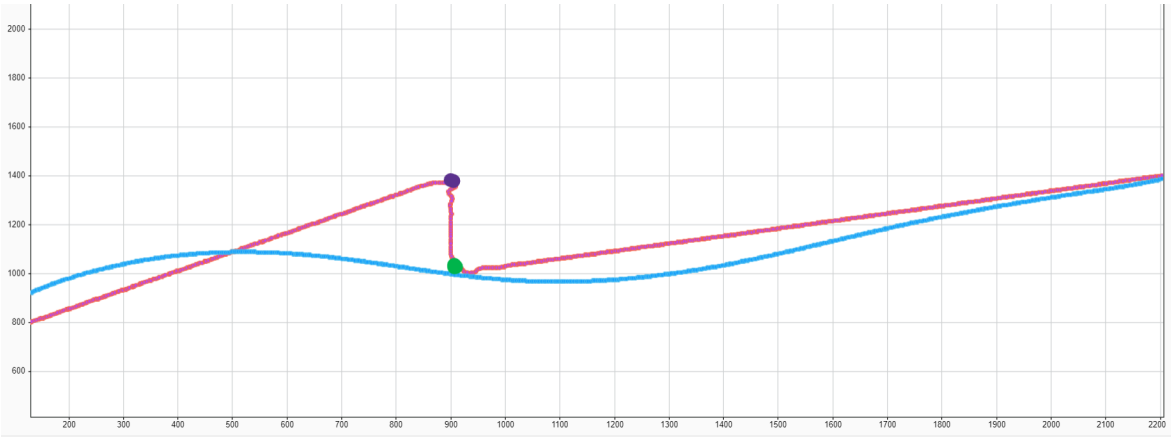


Figure 6.2: First route adaptation obtained from the integrated MATLAB/Simulink simulation. The red trajectory represents the UAV motion, the blue trajectory represents the ship path, the green marker identifies the verified safe point and the purple marker denotes the predefined unsafe waypoint.

The second route adaptation follows the same transition sequence but involves a different section of the route. When the UAV system enters the detection region associated with `WP5`, located at $[2600, 3200, -300]$, it is redirected towards `SAFE2`, located at $[3000, 2800, -300]$.

After the verification condition has been satisfied, the second safe point message activates the following stage of the `ShipWaypointManager`. The original `WP5` is replaced and the ship continues along the second adapted route segment, as shown in Figure 6.3.

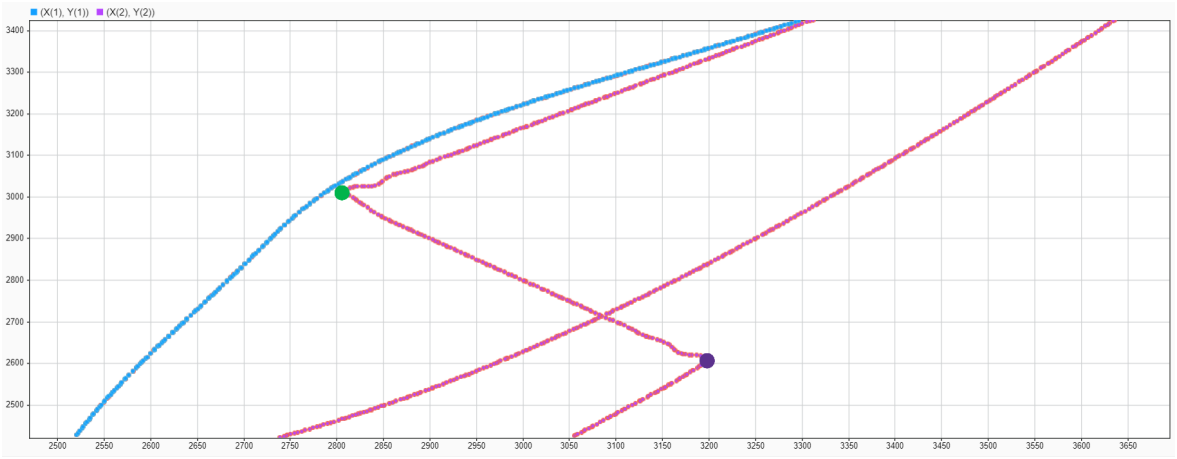


Figure 6.3: Second route adaptation obtained from the integrated MATLAB/Simulink simulation. The red trajectory represents the UAV motion, the blue trajectory represents the ship path, the green marker identifies the verified safe point and the purple marker denotes the predefined unsafe waypoint.

The two adaptations demonstrate that the implemented mission logic can manage multiple changes of the reference route. The first rerouting does not terminate the mission-management process, since the system subsequently returns to the nominal waypoint sequence and remains able to activate the second rerouting stage.

6.2.3 Return and Simplified Capture Phase

After completion of the inspection and communication tasks, the UAV system enters the return phase. Updated ship coordinates are received through the ship-to-UAV communication path and are used to compute the return-course reference of each aerial vehicle.

Since the maritime platform remains in motion, the return direction is continuously updated and is not calculated with respect to a fixed recovery point.

Figure 6.4 shows the resulting return trajectories.

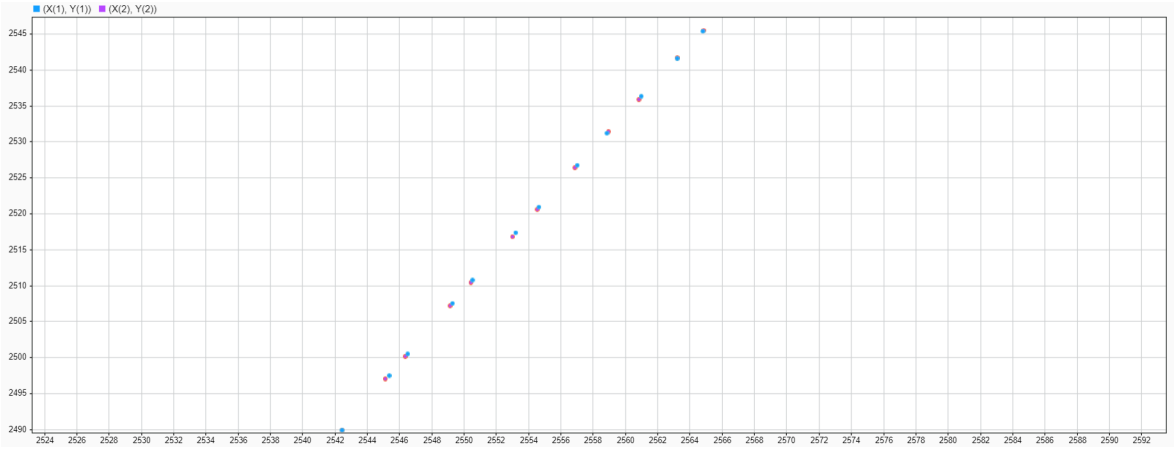


Figure 6.4: Final return and simplified capture phase obtained from the first integrated simulation in the MATLAB/Simulink environment. The red dotted trajectories represent the UAV return paths, whereas the blue trajectory represents the ship path.

In the final portion of the plot, the UAV and ship markers overlap. This indicates that the aerial vehicles have entered the prescribed capture region and that the simplified recovery condition implemented by the `ReturnAndDockSupervisor` has been reached.

This representation verifies the supervisory state transition from return mode to capture mode. It does not reproduce the aerodynamic, mechanical or control details of an actual fixed-wing landing manoeuvre on a moving ship.

6.2.4 Outcome of the Reference Scenario

The first simulation verifies the complete sequence of the integrated architecture:

1. execution of the nominal UAV waypoint mission;
2. activation of the first unsafe-route state;
3. verification and transmission of `SAFE1`;
4. first update of the maritime waypoint sequence;
5. activation of the second unsafe-route state;
6. verification and transmission of `SAFE2`;
7. second update of the maritime waypoint sequence;
8. transmission of the moving ship position to the UAVs;

9. completion of the return and simplified capture phase.

The result therefore provides a functional verification of the interfaces among the UAV mission manager, the message generation logic, the ship waypoint manager and the return supervisor.

6.3 Alternative Route Geometry

The second simulation evaluates the same integrated architecture using a different waypoint arrangement. In this case, the nominal UAV route is defined by the following waypoint sequence:

$\texttt{WP1} = [0, 0, 0]$,

$\texttt{WP2} = [900, 300, -120]$,

$\texttt{WP3} = [1800, 900, -220]$,

$\texttt{WP4} = [2500, 1700, -160]$,

$\texttt{WP5} = [3400, 2500, -300]$,

$\texttt{WP6} = [4700, 3300, -140]$.

Unlike the reference scenario, the route presents a more uniformly oblique progression in the North-East plane, with modified waypoint spacing and safe points located on different sides of the corresponding nominal route segments. The two predefined unsafe locations remain associated with `WP3` and `WP5`.

The purpose of this simulation is not to repeat the complete explanation of the mission logic, but to verify that the implemented state transitions, communication interfaces and waypoint-replacement procedure remain valid after modifying the route geometry.

Figure 6.5 shows the complete trajectory evolution. The UAV system follows the modified nominal route and deviates from it to verify `SAFE1` at $[1500, 1300, -220]$ and `SAFE2` at $[3800, 2200, -300]$.

After their verification, the two sets of coordinates are transmitted to the maritime subsystem. The ship route is consequently updated by replacing `WP3` with `SAFE1` and `WP5` with `SAFE2`.

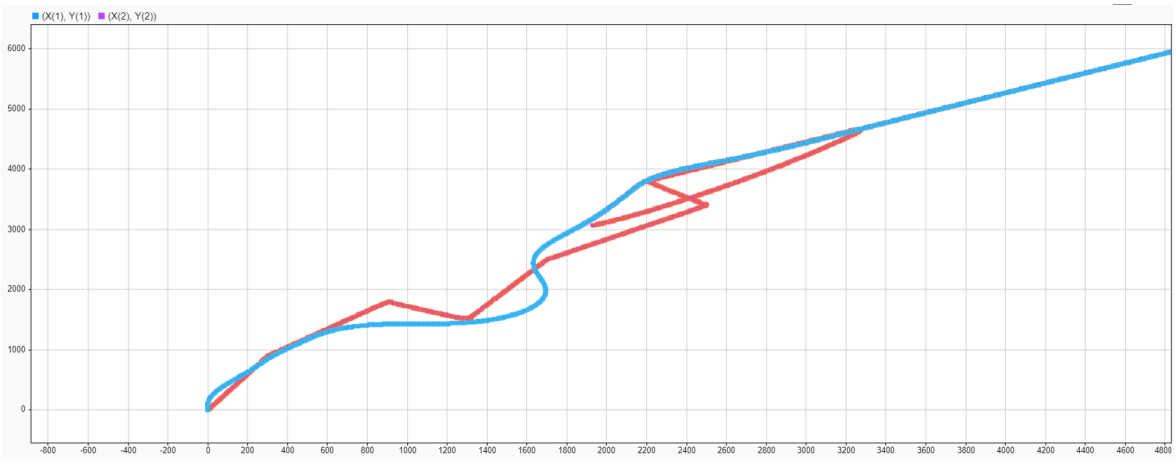


Figure 6.5: Overall trajectory evolution obtained from the second integrated simulation in the MATLAB/Simulink environment. The red trajectory represents the motion of the five UAVs, whereas the blue trajectory represents the ship path.

The two route adaptations are shown in Figures 6.6 and 6.7. In both cases, the ship changes its reference route only after the associated safe point verification condition has been satisfied.

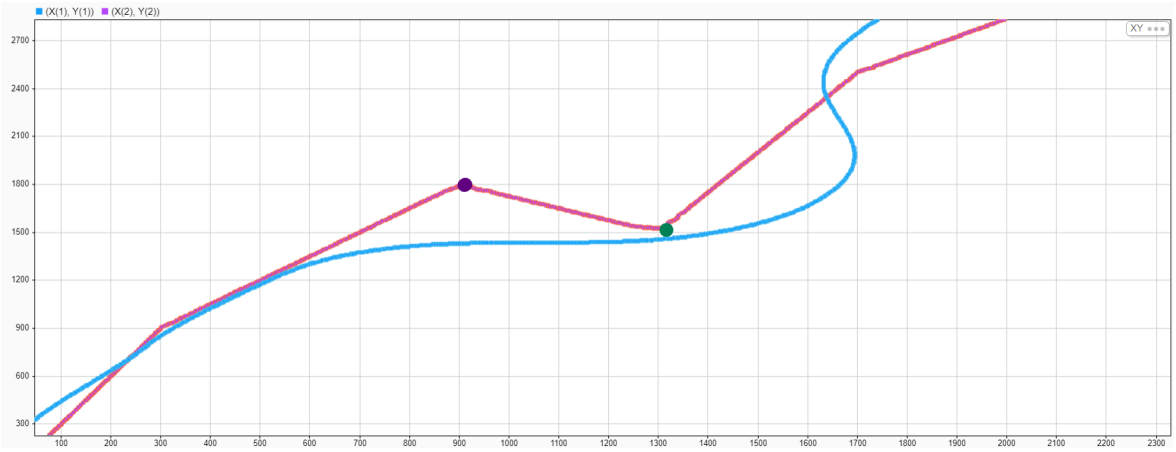


Figure 6.6: First route adaptation obtained from the second integrated MATLAB/Simulink simulation. The green marker identifies `SAFE1`, whereas the purple marker denotes the corresponding predefined unsafe waypoint.

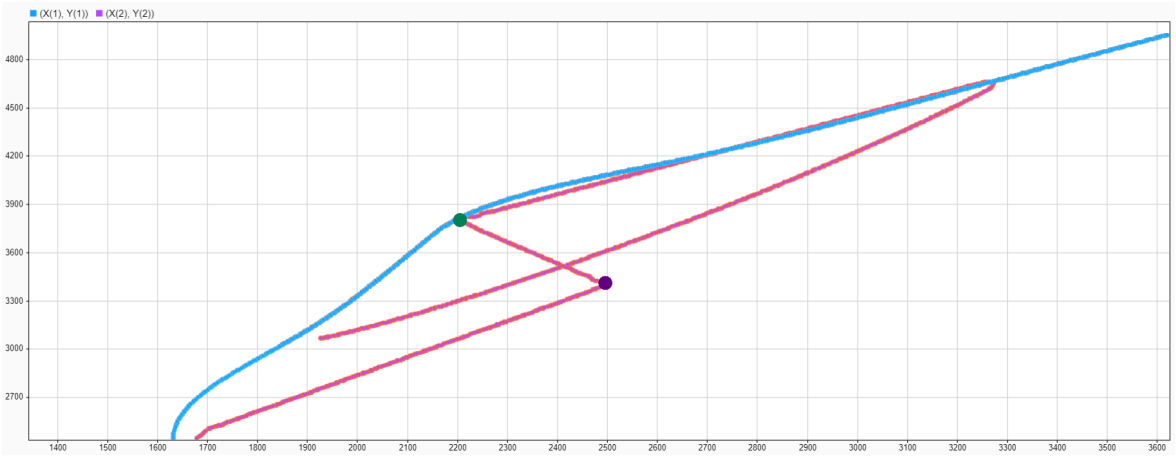


Figure 6.7: Second route adaptation obtained from the second integrated MATLAB/Simulink simulation. The green marker identifies `SAFE2`, whereas the purple marker denotes the corresponding predefined unsafe waypoint.

The return phase is reported in Figure 6.8. Despite the modified ship trajectory, the UAV return-course references remain continuously related to the current maritime-platform position and the final capture condition is reached.

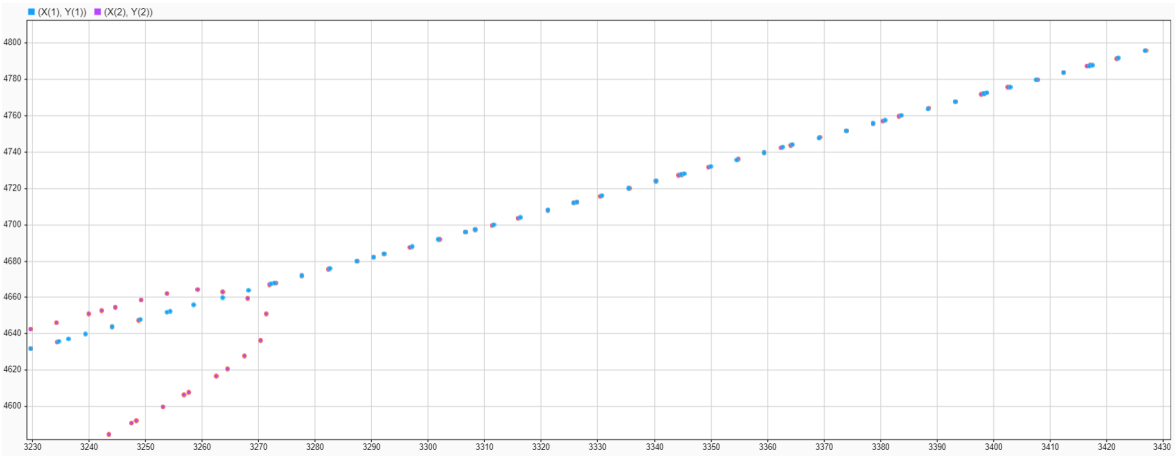


Figure 6.8: Return and simplified capture phase obtained from the second integrated simulation in the MATLAB/Simulink environment. The red dotted trajectories represent the UAV return paths, whereas the blue trajectory represents the ship path.

The second simulation confirms that the implemented architecture is not restricted to the waypoint arrangement used in the reference case. The same functional sequence is completed after modifying the waypoint positions, segment orientations and safe point directions.

6.4 Demanding Route Geometry

The third simulation introduces a more pronounced variation of the route geometry. In this case, the nominal UAV route is defined by the following waypoint sequence:

$\texttt{WP1} = [0, 0, 0]$,

$\texttt{WP2} = [900, 700, -120]$,

$\texttt{WP3} = [1800, 1500, -220]$,

$\texttt{WP4} = [3000, 2200, -160]$,

$\texttt{WP5} = [3900, 1200, -300]$,

$\texttt{WP6} = [6500, -500, -140]$.

Compared with the previous scenarios, the final section of the nominal trajectory presents a more significant change in direction. Moreover, the alternative safe locations require wider deviations from the original ship path. As in the previous simulations, the two predefined unsafe locations correspond to `WP3` and `WP5`.

This scenario is intended to evaluate whether the waypoint-replacement logic remains compatible with the limited manoeuvrability of the Mariner class vessel when the route modifications become more pronounced.

Figure 6.9 shows the overall mission evolution. The UAV system follows the modified nominal route and deviates from it to verify `SAFE1` at $[2000, 2300, -220]$ and `SAFE2` at $[4300, 1900, -300]$.

After the verification process, the safe point coordinates are transmitted to the maritime subsystem. The ship waypoint sequence is consequently updated by replacing `WP3` with `SAFE1` and `WP5` with `SAFE2`.

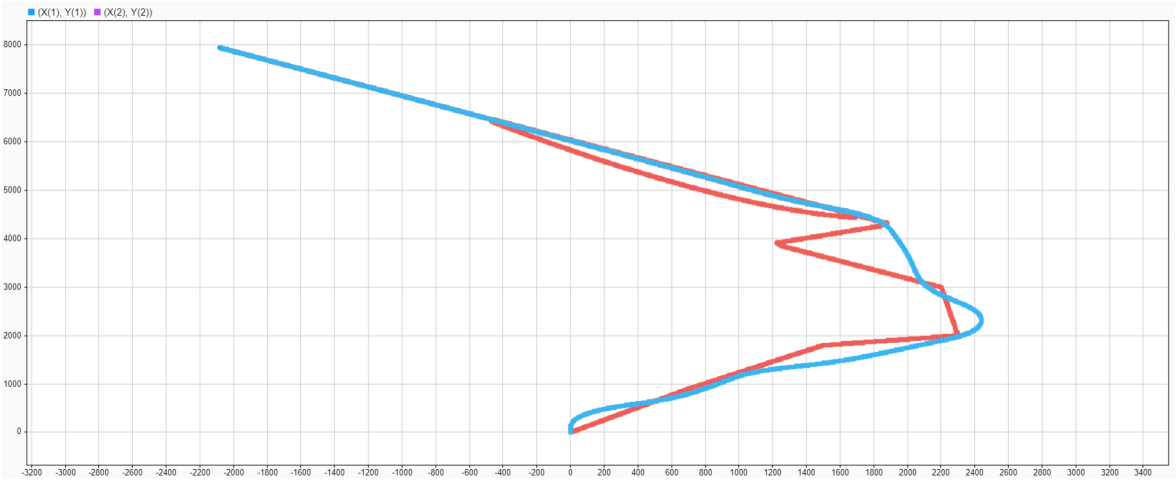


Figure 6.9: Overall trajectory evolution obtained from the third integrated simulation in the MATLAB/Simulink environment. The red trajectory represents the motion of the five UAVs, whereas the blue trajectory represents the ship path.

The first route adaptation is reported in Figure 6.10. In this case, `SAFE1` produces a wider displacement from the original segment than in the reference scenario. Nevertheless, the ship remains capable of converging towards the updated waypoint and subsequently returning to the remaining route sequence.

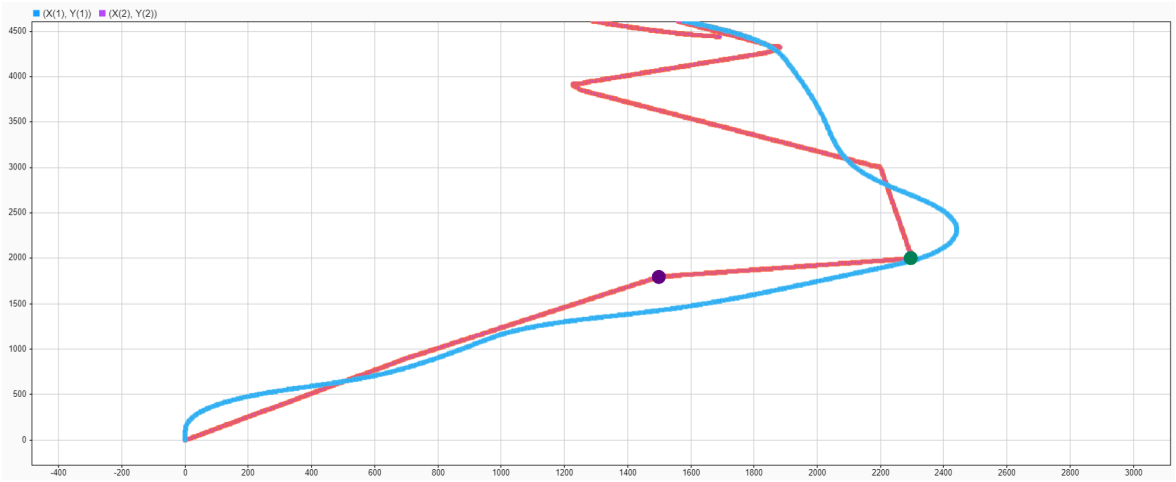


Figure 6.10: First route adaptation obtained from the third integrated MATLAB/Simulink simulation. The green marker identifies `SAFE1`, whereas the purple marker denotes the corresponding predefined unsafe waypoint.

The second adaptation, shown in Figure 6.11, causes a further modification of the maritime trajectory. Compared with the first two simulations, the path requires a more visible change of course before the ship proceeds towards the final waypoint.

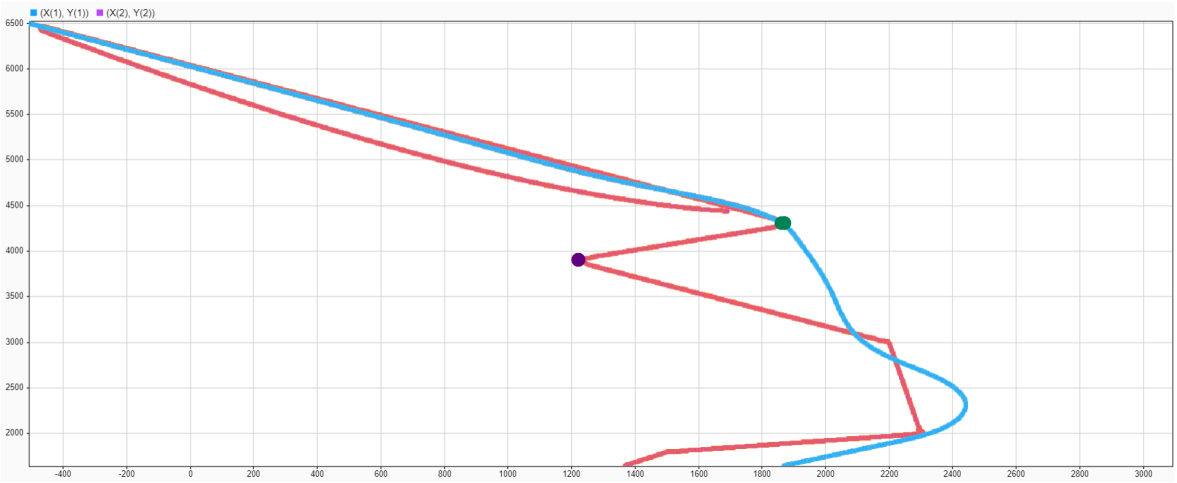


Figure 6.11: Second route adaptation obtained from the third integrated MATLAB/Simulink simulation. The green marker identifies `SAFE2`, whereas the purple marker denotes the corresponding predefined unsafe waypoint.

The UAV return and simplified capture phase is shown in Figure 6.12. The final overlap between the UAV and ship trajectories indicates that the return supervisor reaches the capture condition despite the substantially modified maritime path.

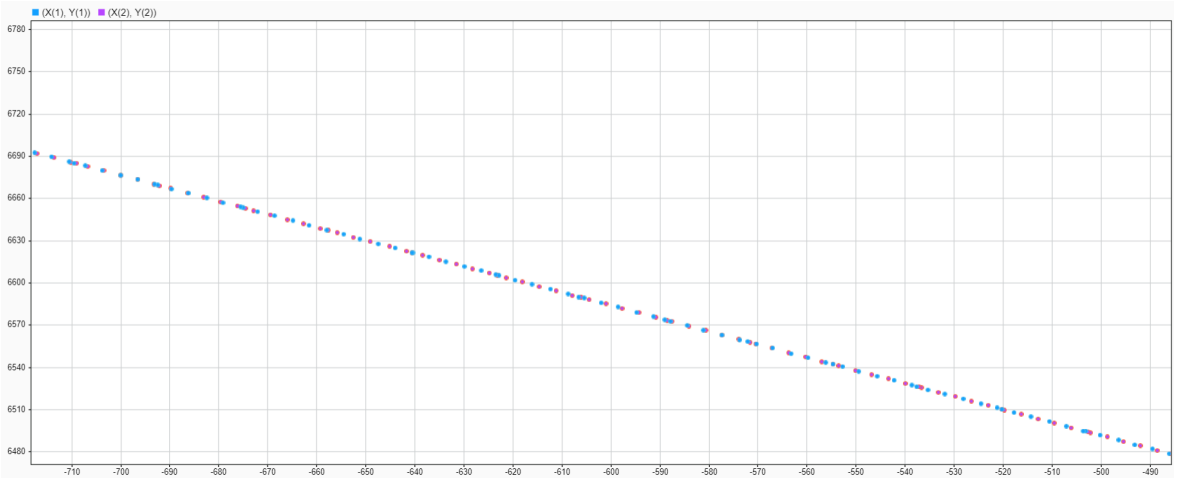


Figure 6.12: Return and simplified capture phase obtained from the third integrated simulation in the MATLAB/Simulink environment. The red dotted trajectories represent the UAV return paths, whereas the blue trajectory represents the ship path.

The third scenario therefore provides the most demanding geometric test among the three analysed cases. The successful completion of both route adaptations shows that the waypoint-replacement mechanism can operate with wider spatial deviations, provided that the alternative safe locations remain compatible with the manoeuvring capability of the ship.

6.5 Comparative Discussion

Table 6.2 provides a direct comparison of the three simulation scenarios.

Case	**Route characteristic**	**Rerouting outcome**	**Recovery outcome**	**Main observation**
1	Reference geometry with two separated deviations	Both unsafe waypoints replaced by verified safe points	Capture condition reached	Complete functional sequence verified in detail
2	Modified oblique route and different safe point directions	Both rerouting stages completed	Capture condition reached	Mission logic remains valid after changing the waypoint arrangement
3	Wider route changes and more pronounced ship manoeuvres	Both rerouting stages completed	Capture condition reached	Waypoint replacement remains feasible under a more demanding geometry

Table 6.2: Qualitative comparison of the three integrated simulation scenarios.

The comparison shows that all three scenarios produce the expected sequence of mission-management events. In each case, the UAV system reaches and verifies both candidate safe points, the corresponding coordinates are transferred to the maritime subsystem and the two unsafe waypoints are replaced before the ship reaches the affected route sections. The final return supervisor also reaches the simplified capture condition in all three configurations.

The main difference among the simulations is therefore not the logical sequence executed by the system, but the geometry under which that sequence is tested. The first scenario provides the reference behaviour, the second demonstrates that the architecture can be applied to a different waypoint arrangement and the third evaluates wider route modifications requiring more pronounced ship manoeuvres.

These results support the functional correctness of the integrated Multidomain architecture. In particular, they verify that the developed supervisory functions remain connected correctly when the waypoint geometry is modified and that bidirectional information exchange influences both the ship rerouting process and the final UAV return phase.

However, the three scenarios differ only in their geometric configuration. Consequently, the results should not be interpreted as a complete robustness assessment. The sim-

ulations do not consider packet loss, communication delays, sensor uncertainty, wind disturbances or energy constraints. These aspects are discussed in Chapter 7 as future extensions of the proposed framework.

Chapter 7

Conclusions and Outlook

7.1 Conclusions

The developed simulation framework addresses the behavioural modelling and simulation of a cooperative multidomain scenario involving a Mariner cargo ship and a fixed-wing UAV swarm. The main objective was to integrate aerial and maritime platforms within a common MATLAB/Simulink environment and to verify how information exchange can support route safety assessment, ship trajectory adaptation and final UAV recovery.

Starting from reference models for the two platforms, the original UAV and ship architectures were extended with additional mission management and communication functionalities. On the aerial side, the UAV swarm was modified to detect unsafe waypoints, redirect the vehicles towards alternative safe points and validate these locations before transmitting them to the maritime platform. This logic allows the UAVs to operate as an active route inspection layer rather than as independent waypoint-following vehicles.

On the maritime side, the Mariner cargo ship model was extended through a rerouting mechanism based on the information received from the UAV leader. The ship does not autonomously generate new safe locations, but updates its waypoint sequence only after a safe point has been verified by the UAV swarm and communicated through the drone-to-ship communication channel. As a result, the maritime trajectory remains explicitly connected to the cooperative mission logic.

The final multidomain model also includes the opposite communication direction. The ship position is transmitted back to the UAV subsystem and used during the return and docking phase. This enables the UAVs to compute a return direction towards the

moving maritime platform, rather than towards a fixed predefined point. Therefore, the implemented communication architecture supports both route adaptation and final recovery.

The integrated model was validated through three simulation scenarios with different waypoint geometries. In each case, the UAV swarm was able to detect unsafe route segments, verify the corresponding safe points and transmit the validated information to the ship. The maritime platform then updated its reference trajectory and avoided the unsafe locations by following the verified safe route. The final phase of the simulations also confirmed that the UAVs could use the updated ship position to return towards the vessel.

Overall, the results confirm the correct interaction among the main components of the proposed architecture: the UAV mission management logic, the ship rerouting logic, the drone-to-ship communication channel, the ship-to-drone communication channel and the return and capture supervision. The simulations provide an initial behavioural validation of the cooperative aerial-maritime framework and show that the implemented logic can be applied to different route geometries while preserving the same information flow and decision sequence.

7.2 Limitations

Although the developed model provides a useful validation of the proposed cooperative architecture, some limitations must be considered.

First, the communication channels are represented in a simplified way. The current implementation assumes that the exchanged messages are correctly transmitted and received, without explicitly modelling packet losses, communication delays, limited bandwidth, interference or range-dependent degradation. This choice is acceptable for an initial behavioural validation, but it does not reproduce all the effects that would appear in a real maritime communication scenario.

Second, the unsafe waypoints and the alternative safe points are predefined in the simulation. The UAV swarm does not autonomously detect environmental hazards or generate alternative safe areas from sensor measurements. Instead, the route-safety logic is based on predefined mission points used to verify the behaviour of the communication and rerouting architecture.

A further limitation concerns the communication strategy based on the UAV leader. In the current model, the ship receives the relevant safe point information from the UAV

leader. This approach makes the rerouting process deterministic and easier to validate, but it does not yet represent a fully distributed swarm communication architecture in which several UAVs independently contribute to the ship decision process.

The return and docking phase is also represented in a simplified way. The UAVs are redirected towards the updated ship position and the final recovery is modelled through a capture condition around the maritime platform. However, detailed fixed-wing landing dynamics, deck constraints, approach corridors, ship motion effects and recovery mechanisms are not included.

Finally, the ship rerouting logic is based on waypoint replacement rather than on a complete online trajectory planner. The ship can avoid unsafe areas by replacing selected waypoints with verified safe points, but it does not continuously optimize its trajectory according to dynamic constraints, collision risk, environmental disturbances or navigation rules.

7.3 Outlook

Several future developments can be considered to extend and improve the proposed simulation framework.

A first extension concerns the communication model. Future work could introduce realistic communication effects, such as transmission delays, packet losses, limited communication range, variable update rates and possible interruptions of the ship-to-UAV and UAV-to-ship links. In the current implementation, a first step in this direction could be obtained by acting on the message validity flag. For example, the first row of the `MesDrones` matrix is currently set equal to 1 to indicate that the transmitted message is valid and correctly received. By modifying this value according to predefined conditions, it would be possible to emulate temporary communication losses or unavailable messages within the existing architecture. This would make it possible to evaluate the robustness of the cooperative logic under more realistic operating conditions.

Another important development would be the introduction of perception-based route assessment. Instead of using predefined unsafe waypoints and safe points, the UAVs could be equipped with simulated sensors or detection algorithms capable of identifying obstacles, restricted areas or hazardous regions along the route. Consequently, the safe point generation process could become more autonomous and closer to a real mission scenario.

The swarm coordination strategy could also be extended. Instead of relying on a

single UAV leader, future implementations could allow multiple UAVs to transmit complementary information, compare observations or reach a shared decision before communicating with the ship. This would increase the robustness of the system and reduce the dependence on a single aerial agent.

Further work could also improve the return and docking phase. A more detailed recovery model could include relative positioning between the UAVs and the ship, approach direction constraints, deck geometry, ship motion and fixed-wing landing requirements. This would make the final recovery phase more representative of a real maritime UAV operation.

Finally, environmental effects could be further exploited in future simulations. The developed UAV model already includes an environmental subsystem, but in the present implementation the wind contribution is not configured to represent a disturbed flight condition. Future work could therefore assign non-zero wind components to the existing environmental inputs in order to evaluate the behaviour of the UAV swarm under more realistic operating conditions. Additional disturbances could also be considered on the maritime side, such as sea currents, wave-induced ship motion and sensor uncertainty. These additions would support a more complete evaluation of the proposed multidomain architecture and could represent a useful step towards real-time simulation, hardware-in-the-loop testing or experimental validation.

In conclusion, the developed MATLAB/Simulink framework represents a first step towards the simulation of coordinated aerial-maritime operations. The results show that communication between a fixed-wing UAV swarm and a cargo ship can support route-safety evaluation, trajectory adaptation and return-to-ship behaviour. While several modelling simplifications remain, the proposed architecture provides a structured basis for future extensions towards more realistic, autonomous and robust multidomain cooperative systems.

Appendix A

Source Code Listings

A.1 Dynamic Waypoint Manager Function

```
function [wayPoints_active, msgType, modeUnsafe, safeNorth,
   safeEast] = DynamicWaypointManager(droneNorth, droneEast)

persistent mode
persistent sentSafe1
persistent sentSafe2

if isempty(mode)
    mode = zeros(5,1);
    sentSafe1 = false(5,1);
    sentSafe2 = false(5,1);
end

wayPoints_active = zeros(8,3,5);
msgType    = ones(5,1);   % 1 = normal message, 3 = verified
   safe point to be communicated
modeUnsafe = zeros(5,1);
safeNorth  = zeros(5,1);
safeEast   = zeros(5,1);

% Starting waypoints
WP1 = [0    0      0];
WP2 = [700  0    -120];
WP3 = [1400 900  -220];    % first unsafe waypoint
```

```
23 WP4 = [1400 2200 -160];
24 WP5 = [2600 3200 -300];    % second unsafe waypoint
25 WP6 = [4000 4000 -140];
26 WP7 = WP6;
27 WP8 = WP6;
28
29 % Safe points
30 SAFE1 = [1000 900  -220];
31 SAFE2 = [3000 2800 -300];
32
33 R_detect1 = 60;
34 R_safe1   = 40;
35
36 R_detect2 = 60;
37 R_safe2   = 40;
38
39 % Nominal waypoint sequence
40 WP_nom = [WP1;
41           WP2;
42           WP3;
43           WP4;
44           WP5;
45           WP6;
46           WP7;
47           WP8];
48
49 % Alternative waypoint sequence including the safe points
50 WP_safe_complete = [WP1;
51                     WP2;
52                     WP3;
53                     SAFE1;
54                     WP4;
55                     WP5;
56                     SAFE2;
57                     WP6];
58
59 for i = 1:5
60
61     % Distances from the unsafe waypoints
62     d_wp3 = sqrt((droneNorth(i) - WP3(1))^2 + ...
```

```
63                  (droneEast(i)  - WP3(2))^2);
64
65      d_wp5 = sqrt((droneNorth(i) - WP5(1))^2 + ...
66                  (droneEast(i)  - WP5(2))^2);
67
68      % Distances from the safe points
69      d_safe1 = sqrt((droneNorth(i) - SAFE1(1))^2 + ...
70                    (droneEast(i)  - SAFE1(2))^2);
71
72      d_safe2 = sqrt((droneNorth(i) - SAFE2(1))^2 + ...
73                    (droneEast(i)  - SAFE2(2))^2);
74
75      % Operating modes:
76      % 0 = nominal navigation
77      % 1 = unsafe waypoint WP3 detected, UAV redirected toward
           SAFE1
78      % 2 = SAFE1 verified, normal navigation resumed toward WP4/
           WP5
79      % 3 = unsafe waypoint WP5 detected, UAV redirected toward
           SAFE2
80      % 4 = SAFE2 verified
81
82      if mode(i) == 0 && d_wp3 <= R_detect1
83          mode(i) = 1;
84      end
85
86      if mode(i) == 1 && d_safe1 <= R_safe1
87          mode(i) = 2;
88      end
89
90      if mode(i) == 2 && d_wp5 <= R_detect2
91          mode(i) = 3;
92      end
93
94      if mode(i) == 3 && d_safe2 <= R_safe2
95          mode(i) = 4;
96      end
97
98      % Selection of the active trajectory for each UAV
99      if mode(i) == 0
```

```
100         wayPoints_active(:,:,i) = WP_nom;
101     else
102         wayPoints_active(:,:,i) = WP_safe_complete;
103     end
104
105     % Default condition: normal message
106     msgType(i) = 1;
107
108     % SAFE1 verified: communicate the safe point to the ship
            only once
109     if mode(i) == 2 && ~sentSafe1(i)
110         msgType(i) = 3;
111         safeNorth(i) = SAFE1(1);
112         safeEast(i)  = SAFE1(2);
113         sentSafe1(i) = true;
114
115     % SAFE2 verified: communicate the safe point to the ship
            only once
116     elseif mode(i) == 4 && ~sentSafe2(i)
117         msgType(i) = 3;
118         safeNorth(i) = SAFE2(1);
119         safeEast(i)  = SAFE2(2);
120         sentSafe2(i) = true;
121
122     else
123         % No new communication to the ship
124         if mode(i) < 3
125             safeNorth(i) = SAFE1(1);
126             safeEast(i)  = SAFE1(2);
127         else
128             safeNorth(i) = SAFE2(1);
129             safeEast(i)  = SAFE2(2);
130         end
131     end
132
133     modeUnsafe(i) = mode(i);
134
135 end
```

Code A.1: MATLAB Implementation of the `DynamicWaypointManager` function.

A.2 Return and Dock Supervisor Function

```
function [desiredCourse_out, northOut, eastOut, heightOut,
   headingOut, modeOut] = ...
    ReturnAndDockSupervisor(desiredCourse_wp, droneNorth,
       droneEast, ...
    droneHeight, droneHeading, shipNorth, shipEast)

numUAVs = 5;
persistent mode

if isempty(mode)
    mode = zeros(numUAVs,1);
end

desiredCourse_out = zeros(numUAVs,1);

northOut   = droneNorth;
eastOut    = droneEast;
heightOut  = droneHeight;
headingOut = droneHeading;
modeOut    = zeros(numUAVs,1);

North_last = 4000;
East_last  = 4000;

R_end = 60;
R_capture = 85;

deckHeight = 0;

for i = 1:numUAVs

    shipNorth_i = shipNorth(i);
    shipEast_i  = shipEast(i);

    d_end = sqrt((droneNorth(i)-North_last)^2 + ...
                 (droneEast(i)-East_last)^2);

    d_ship = sqrt((shipNorth_i-droneNorth(i))^2 + ...
```

```
37                  (shipEast_i-droneEast(i))^2);
38
39      if mode(i) == 0 && d_end <= R_end
40          mode(i) = 1;
41      end
42
43      if mode(i) == 1 && d_ship <= R_capture
44          mode(i) = 2;
45      end
46
47      if mode(i) == 0
48          desiredCourse_out(i) = desiredCourse_wp(i);
49
50      elseif mode(i) == 1
51          desiredCourse_out(i) = atan2(shipEast_i - droneEast(i),
                ...
52                                       shipNorth_i - droneNorth(i)
                                            );
53
54      else
55          desiredCourse_out(i) = desiredCourse_wp(i);
56          northOut(i)   = shipNorth_i;
57          eastOut(i)    = shipEast_i;
58          heightOut(i)  = deckHeight;
59          headingOut(i) = droneHeading(i);
60      end
61
62      modeOut(i) = mode(i);
63  end
```

Code A.2: MATLAB implementation of the `ReturnAndDockSupervisor` function.

A.3 Ship Waypoint Manager Function

```
1  function [xdata_out, ydata_out, rerouteFlag] =
       ShipWaypointManager(modeUnsafe_leader, xdata_out, ydata_out,
       safeNorth_leader, safeEast_leader)
2
3  persistent rerouteStage
```

```
persistent safe1_North safe1_East
persistent safe2_North safe2_East

if isempty(rerouteStage)
    rerouteStage = 0;

    safe1_North = 1000;
    safe1_East = 900;

    safe2_North = 3000;
    safe2_East = 2800;
end

% rerouteStage:
% 0 = no ship rerouting
% 1 = SAFE1 verified by the UAV, ship route is updated through SAFE1
% 2 = SAFE1 and SAFE2 verified by the UAV, ship route is also updated through SAFE2

% First safe point verified: the UAV has reached SAFE1
if modeUnsafe_leader >= 2 && rerouteStage < 1
    rerouteStage = 1;

    safe1_North = safeNorth_leader;
    safe1_East = safeEast_leader;
end

% Second safe point verified: the UAV has reached SAFE2
if modeUnsafe_leader >= 4 && rerouteStage < 2
    rerouteStage = 2;

    safe2_North = safeNorth_leader;
    safe2_East = safeEast_leader;
end

% Apply the first rerouting only after SAFE1 verification
if rerouteStage >= 1
    xdata_out(3) = safe1_North;
    ydata_out(3) = safe1_East;
```

```
42 end
43
44 % Apply the second rerouting only after SAFE2 verification
45 if rerouteStage >= 2
46     xdata_out(5) = safe2_North;
47     ydata_out(5) = safe2_East;
48 end
49
50 rerouteFlag = rerouteStage;
```

Code A.3: MATLAB implementation of the `ShipWaypointManager` function.

A.4 Build Drone Messages Function

```
1  function MesDrones = BuildDroneMessages(Uavinfo, modeUnsafe,
       safeNorth, safeEast)
2  MesDrones = zeros(7,5);
3
4  for i = 1:5
5
6      north_i = Uavinfo(i,1);
7      east_i  = Uavinfo(i,2);
8
9      MesDrones(1,i) = 1;              % valid
10     MesDrones(2,i) = i;              % senderID
11     MesDrones(3,i) = north_i;        % north drone i
12     MesDrones(4,i) = east_i;         % east drone i
13     MesDrones(5,i) = modeUnsafe(i);  % stato unsafe/safe
14     MesDrones(6,i) = safeNorth(i);   % safe north
15     MesDrones(7,i) = safeEast(i);    % safe east
16
17 end
```

Code A.4: MATLAB implementation of the `BuildDroneMessages` function.

References

[1] P. Dong, J. Liu, H. Tao, Y. Zhao, Z. Feng, and H. Luo, "Long-endurance collaborative search and rescue based on maritime unmanned systems and deep-reinforcement learning," *Sensors*, vol. 25, no. 13, p. 4025, 2025, doi: 10.3390/s25134025.

[2] Y. Wang, W. Liu, J. Liu, and C. Sun, "Cooperative USV–UAV marine search and rescue with visual navigation and reinforcement learning-based control," *ISA Transactions*, 2023, doi: 10.1016/j.isatra.2023.01.007.

[3] L. Checker, H. Xie, S. Khaksar, and I. Murray, "Systematic review of multi-objective UAV swarm mission planning systems from regulatory perspective," *Drones*, vol. 9, no. 7, Art. no. 509, 2025, doi: 10.3390/drones9070509.

[4] X. Xiao, J. Dufek, T. Woodbury, and R. Murphy, "UAV assisted USV visual navigation for marine mass casualty incident response," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, doi: 10.1109/IROS.2017.8206510.

[5] M. W. Akhtar and N. Saeed, "UAVs-enabled maritime communications: Opportunities and challenges," *arXiv preprint arXiv:2206.03118*, 2022, doi: 10.48550/arXiv.2206.03118.

[6] F. Li, Y. Shui, J. Liang, K. Yang, and J. Yu, "Ship-to-ship maritime wireless channel modeling under various sea state conditions based on REL model," *Frontiers in Marine Science*, vol. 10, Art. no. 1134286, 2023, doi: 10.3389/fmars.2023.1134286.

[7] G. Yu, X. Ding, and S. Liu, "Joint resource management and trajectory optimization for UAV-enabled maritime network," *Sensors*, vol. 22, no. 24, p. 9763, 2022, doi: 10.3390/s22249763.

[8] N. Nomikos, P. K. Gkonis, P. S. Bithas, and P. Trakadas, "A survey on UAV-aided maritime communications: Deployment considerations, applications, and

future challenges," *IEEE Open Journal of the Communications Society*, vol. 4, pp. 56–84, 2023, doi: 10.1109/OJCOMS.2022.3225590.

[9] J. P. Queralta, J. Taipalmaa, B. C. Pullinen, V. K. Sarker, T. N. Gia, H. Tenhunen, M. Gabbouj, J. Raitoharju, and T. Westerlund, "Collaborative multi-robot search and rescue: Planning, coordination, perception, and active vision," *IEEE Access*, vol. 8, pp. 191617–191643, 2020, doi: 10.1109/ACCESS.2020.3030190.

[10] N. Nomikos, A. Giannopoulos, A. Kalafatelis, V. Ozduran, and P. Trakadas, "Improving connectivity in 6G maritime communication networks with UAV swarms," *IEEE Access*, vol. 12, pp. 18739–18751, 2024, doi: 10.1109/ACCESS.2024.3360133.

[11] I. Bae and J. Hong, "Survey on the developments of unmanned marine vehicles: Intelligence and cooperation," *Sensors*, vol. 23, no. 10, p. 4643, 2023, doi: 10.3390/s23104643.

[12] J. Li, G. Zhang, C. Jiang, and W. Zhang, "A survey of maritime unmanned search system: Theory, applications and future directions," *Ocean Engineering*, vol. 285, p. 115359, 2023, doi: 10.1016/j.oceaneng.2023.115359.

[13] M. C. Santos, B. Bartlett, V. E. Schneider, et al., "Cooperative unmanned aerial and surface vehicles for extended coverage in maritime environments," *IEEE Access*, vol. 12, pp. 9206–9219, 2024, doi: 10.1109/ACCESS.2024.3353046.

[14] R. Costanzi, D. Fenucci, V. Manzari, M. Micheli, L. Morlando, D. Terracciano, A. Caiti, M. Stifani, and A. Tesei, "Interoperability among unmanned maritime vehicles: Review and first in-field experimentation," *Frontiers in Robotics and AI*, vol. 7, p. 91, 2020, doi: 10.3389/frobt.2020.00091.

[15] W. Müller et al., "Interoperability open architecture of unmanned systems," in *Proceedings of the 2023 Congress in Computer Science, Computer Engineering, and Applied Computing (CSCE)*, Las Vegas, NV, USA, 2023, pp. 2621–2629, doi: 10.1109/CSCE60160.2023.00419.

[16] T. I. Fossen, "Line-of-sight path-following control utilizing an extended Kalman filter for estimation of speed and course over ground from GNSS positions," *Journal of Marine Science and Technology*, vol. 27, no. 3, pp. 806–813, 2022, doi: 10.1007/s00773-022-00872-y.

[17] C.-H. Chang, G. Kontovas, N. Ventikos, and L. E. E. Svenson, "Risk assessment of the operations of maritime autonomous surface ships," *Reliability Engineering and System Safety*, vol. 207, p. 107324, 2021, doi: 10.1016/j.ress.2020.107324.

[18] The MathWorks, Inc., "Tuning Waypoint Following Controller for Fixed-Wing UAV," *MathWorks Documentation*, 2026. [Online]. Available: https://www.mathworks.com/help/uav/ug/tuning-waypoint-follower-for-fixed-wing.html. Accessed: Jun. 2, 2026.

[19] The MathWorks, Inc., "Simulate Multiple Fixed-Wing UAVs in Simulink Using System Objects," *MathWorks Documentation*, 2026. [Online]. Available: https://www.mathworks.com/help/uav/ug/simulating-multiple-UAV-simulink-system-object-blocks.html. Accessed: Jun. 4, 2026.

[20] T. I. Fossen and T. Perez, "Marine Systems Simulator (MSS)," 2004. [Online]. Available: https://github.com/cybergalactic/MSS. Accessed: Jun. 4, 2026.

[21] T. Perez, Ø. N. Smogeli, T. I. Fossen, and A. J. Sørensen, "An overview of the Marine Systems Simulator (MSS): A Simulink toolbox for marine control systems," *Modeling, Identification and Control*, vol. 27, no. 4, pp. 259–275, 2006, doi: 10.4173/mic.2006.4.4.