Adversarial Inverse Reinforcement Learning for Energy-Efficient Marine Vessel Operation

Master's thesis in Systems, Control and Mechatronics

Anish Janardhan
Muddur Ajay Nayak

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Adversarial Inverse Reinforcement Learning for Energy-Efficient Marine Vessel Operation

Anish Janardhan
Muddur Ajay Nayak

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2025

Adversarial Inverse Reinforcement Learning for Energy-Efficient Marine Vessel Operation
Anish Jarnardhan
Muddur Ajay Nayak

Supervisor: Simon Johansson, Cetasol AB
Examiner: Balázs AdamKulcsár, Chalmers University of Technology

Master's Thesis 2025
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Vessel navigation using AIRL algorithm.

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2025

Adversarial Inverse Reinforcement Learning for Energy-Efficient Marine Vessel Operation
Anish Jarnardhan
Muddur Ajay Nayak
Department of Electrical Engineering
Chalmers University of Technology

Abstract

The maritime industry serves as a backbone of global trade, transporting nearly 80% of goods worldwide, making it indispensable for the global economy (Statista,2024; UNCTAD, 2024). Yet, according to UNCTAD, this sector remains a notable contributor to greenhouse gas (GHG) emissions, accounting for 2-3% of total emissions- a figure projected to rise dramatically by 2050 under business-as-usual scenarios. Addressing this escalating environmental impact requires innovative strategies for optimizing vessel operations and minimizing fuel consumption, which are crucial for sustainable maritime practices. This thesis focuses exclusively on the application of adversarial inverse reinforcement learning (AIRL) to tackle these challenges. AIRL leverages expert demonstrations from seasoned captains who have successfully navigated complex and diverse maritime conditions. By analyzing these expert behaviors, AIRL derives comprehensive reward function that encapsulate optimal operational strategies, paving the way for the development of a control policy.

The strength of AIRL lies in its ability to capture the nuances of expert judgment, enabling marine vessels to replicate the energy-efficient decisions made by experienced captains. Unlike traditional control systems, which often struggle with the nonlinear dynamics and diverse challenges of maritime environments, AIRL offers a robust data-driven solution tailored to the specific operational needs of the vessels.

By integrating AIRL, this research aims to enhance energy efficiency, reduce fuel consumption, and lower carbon emissions. In doing so, it contributes not only to the operational goals of modern marine vessels but also to the broader environmental objectives of the global community. This work aligns with the maritime industry's transition toward greener practices and supports the adoption of sustainable solutions.

Keywords: AIRL, Actor-Critic, Generator, Discriminator, OpenAI Gym

Acknowledgements

First and foremost, we would like to express our heartfelt gratitude to Simon Johansson from Cetasol AB for providing us with this incredible opportunity to work on a project that aims to make meaningful contributions to the maritime industry. Your trust and support have been instrumental in driving this work forward.

We are deeply grateful to Balázs Kulcsár for his continuous supervision, invaluable insights, and unwavering support throughout this journey. His guidance and expertise have been a cornerstone in overcoming challenges and refining our research.

To our parents and families, thank you for your constant encouragement and understanding. Your unwavering support has been our strength and inspiration, enabling us to focus and persevere through this endeavor.

Anish Janardhan
Muddur Ajay Nayak
Gothenburg, June , 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

AIRL	Adversarial Inverse Reinforcement Learning
GAIL	Generative Adversarial Imitation Learning
GAN	Generative Adversarial Network
IMU	Inertial Measurement Unit
IRL	Inverse Reinforcement Learning
MC	Monte Carlo
MDP	Markov Decision Process
MPC	Model Predictive Control
PPO	Proximal Policy Optimization
RL	Reinforcement Learning
TD	Temporal Difference
WandB	Weights and Biases platform

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Variables

s	State
s'	Next state
a	Action

Sets

$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions

Parameters

$P(s'\|s,a)$	State transition probability
$R(s,a)$	Reward function
γ	Discount factor
$\pi(a\|s)$	Learned policy
$\pi_E(a\|s)$	Expert policy
$V_\pi(s)$	State-value function
$Q_\pi(s)$	Action-value function
D_θ	Discriminator parametrized by θ
$f_\theta(s,a)$	Learned function by discriminator
$g_\theta(s,a)$	Primary reward predictor
$h_\phi(s)$	Reward shaping term

$A(s, a)$	Advantage function
Δv	change in velocity
$\Delta\theta$	change in heading
t	time step
Δt	Time discretization step (time interval)
v_t	Speed at time t
θ_t	Heading at time t
ϕ	Course
δ	Angular displacement
α	Latitude
λ	Longitude

Contents

List of Figures

List of Tables

1
Introduction

1.1 Thesis Objective

The objective of this thesis is to address the critical need for reducing energy consumption in marine vessels while maintaining operational efficiency across diverse conditions. By leveraging advanced machine learning techniques, specifically AIRL, this thesis aims to develop a robust algorithm capable of optimizing control actions across varying operational states. The cornerstone of this approach is the implementation of AIRL to derive an optimal reward function based on data generated from expert demonstrations by experienced captains. This enables the formulation of a control policy representing the energy-efficient and expert-level decision-making that the captain observes in real-world maritime operations.

One of the primary challenges in this thesis is effectively training machine learning models to capture the nonlinear dynamics inherent to marine vessel operations. These dynamics are influenced by complex factors such as varying load conditions, environmental disturbances like wind and waves, and the intricate interplay between propulsion and maneuvering systems. Traditional control approaches often rely on static models that fail to adapt to these dynamic conditions, resulting in suboptimal control actions. This thesis investigates how a discriminator and generator model can be trained to accurately represent these optimal actions, thereby enabling expert like and robust control actions.

By providing an analysis of how the AIRL algorithm performs in real-world conditions, this thesis aims to highlight its potential as a solution for sustainable maritime vessel operations. Ultimately, this thesis contributes to ongoing efforts to mitigate the environmental impact of the maritime industry, supporting its transition to greener and more energy-efficient practices.

1.2 Research Question

The research questions that this thesis aims to answer are as follows:

- How can ML models be effectively trained to capture the non-linear dynamics of marine vessels across different operational states?
- How can AIRL be applied to learn an optimal reward function from expert demonstrations and thereby enabling to get the optimal policy?

1.3 Scope and Limitations

Scope: This thesis focuses on the application of AIRL to optimize energy-efficient routes for marine vessels. By utilizing data derived from expert demonstrations, specifically from experienced captains who have consistently navigated challenging operational conditions, the algorithm is designed to extract a reward function reflecting optimal energy-efficient behaviors. This extracted reward function forms the basis for developing an optimal policy that determines energy-efficient routes and control actions. The embedded knowledge of experienced captains within these data sets ensures that the proposed algorithm aligns closely with real-world operational strategies, making it both practical and effective.

The focus of this thesis is to learn expert decision-making by the captain as closely as possible, ensuring optimal actions are taken to navigate the vessel from the start to the destination point. Furthermore, the feasibility and performance of AIRL will be evaluated to ensure its ability to meet the desired optimization goals effectively. The thesis also aims to evaluate the robustness of the obtained policy. This ensures the practical applicability of the methodology and provides a realistic assessment of its impact on energy consumption and emission reduction.

Limitations: While this thesis presents an efficient and robust approach to optimizing marine vessel operations, it is important to understand its boundaries. The scope of optimization is specifically limited to suggesting optimal actions to be taken rather than addressing other aspects of vessel control, such as engine dynamics. This narrow focus ensures a deeper exploration of trajectory optimization, but may exclude other aspects of operational efficiency.

Additionally, the thesis is restricted to finding the optimal actions between two points, meaning scenarios involving complex multi-leg journeys or highly variable environmental conditions may not be fully addressed. The reliance on data derived from expert captains also assumes that these demonstrations comprehensively reflect optimal energy-efficient behavior, which may not account for all potential operational scenarios.

In summary, while this thesis takes a significant step towards energy efficiency in marine operations, its findings are framed within the specific scope and limitations outlined above, emphasizing the importance of future explorations to broaden its applicability.

1.4 Outline

The structure of this report is organized into the following key sections to provide a clear and comprehensive understanding of the research.

1. Theory: This section will provide a comprehensive overview of the theoretical foundations relevant to the thesis. It will introduce the key concepts of AIRL, reward function modeling, and optimal policy derivation. Additionally, it will explore the role of expert demonstrations in guiding the learning process and discuss the nonlinear dynamics of marine vessel operations.

2. Methodology: The methodology section will detail the step-by-step approach used to design, implement, and test the AIRL-based algorithm. It will describe how expert data was preprocessed, the structure of the AIRL pipeline, and the process of reward function extraction and policy generation. Furthermore, it will outline the evaluation setup on marine vessels, specifying the parameters used for evaluation and the criteria for assessing energy efficiency and emission reduction.

3. Results and Discussion: This section will present the findings of the research, focusing on the performance of the proposed AIRL-based system in optimizing marine vessel trajectories and reducing energy consumption. It will include an analysis of the results to evaluate the algorithm's robustness. The discussion will interpret the significance of the findings, address any limitations encountered during testing, and consider the implications for broader applications in the maritime industry.

4. Conclusion and Future Work: The report will conclude by summarizing the key achievements of the thesis, emphasizing its contributions to sustainable maritime operations. It will reflect on the success of the AIRL approach in achieving energy-efficient vessel trajectories and reducing emissions. Additionally, this section will outline potential future work, such as expanding the algorithm's scope to multi-leg journeys, incorporating other aspects of vessel control to address broader operational scenarios. These suggestions aim to provide directions for further advancements in AIRL-based maritime optimization.

2
Theory

2.1 Reinforcement Learning

This section explaining the concepts of RL is referred from the book *Reinforcement Learning: An Introduction* by Richard S. Sutton and Andrew G. Barto [14].

Reinforcement Learning is a subfield of machine learning where an agent interacts with an environment to maximize cumulative reward. It differs from supervised learning as it does not rely on labeled datasets but learns through trial and error interactions with the environment, receiving feedback in the form of rewards or penalties. To formalize this processes, RL relies on the framework of Markov Decision Processes (MDP), which is introduced in the next section.

2.1.1 Markov Decision Process

A MDP provides a mathematical framework for modeling decision-making in RL, where outcomes are partly random and partly controlled by a decision-maker. An MDP provides a structured framework for defining environments in reinforcement learning, consisting of states, actions, transition probabilities, and rewards. A fundamental assumption of MDPs is the Markov property, which states that the future state depends only on the current state and action, not on the sequence of past states. It is formally defined as a tuple [14] :

$$(S, A, P, R, \gamma) \tag{2.1}$$

where:

- S is the set of states.
- A is the set of actions.
- $P(s'|s, a)$ is the state transition probability.
- $R(s, a)$ is the reward function.
- $\gamma \in [0, 1]$ is the discount factor.

With the MDP framework, the agent develops a policy that optimizes cumulative rewards over time.

2.1.2 Policy and Value Functions

The goal of RL is to learn a policy π that maximizes the expected return. A policy is a mapping from states to probabilities of selecting actions, $\pi(a|s)$.To evaluate

the quality of the policy, the value functions are used, which estimate the expected reward an agent can get.
The state-value function $V_\pi(s)$ is given by [14]:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s \right] \tag{2.2}$$

Similarly, the action-value function $Q_\pi(s, a)$ is [14]:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) | s_0 = s, a_0 = a \right] \tag{2.3}$$

Value functions play a crucial role in training agents and improving their policies. The Bellman equation is used to compute them efficiently. It breaks down value functions recursively and allows the value of a state to be expressed in relation to the values of its successor states, making learning more structured and manageable.

$$V_\pi(s) = \sum_{a \in A} \pi(a|s) \sum_{s'} P(s'|s, a)[R(s, a) + \gamma V_\pi(s')] \tag{2.4}$$

For optimal policies, the Bellman optimality equation is:

$$V^*(s) = \max_{a \in A} \sum_{s'} P(s'|s, a)[R(s, a) + \gamma V^*(s')] \tag{2.5}$$

These equations are fundamental as they define how an optimal policy should be structured to maximize expected rewards.

2.1.3 Dynamic Programming and RL Algorithms

Dynamic programming methods, such as Value Iteration and Policy Iteration, solve the Bellman equations. However, in real-world RL problems where the environment is unknown or is too complex, model-free approaches such as Monte Carlo (MC) methods and Temporal Difference (TD) Learning are preferred.

2.1.3.1 Monte Carlo Methods

Monte Carlo methods estimate value functions by averaging returns sampled from complete episodes of experience. Unlike dynamic programming, MC methods do not require a model of the environment and learn directly from observed transitions. They update value estimates only after an episode ends, making them useful for episodic tasks.

2.1.3.2 Temporal Difference Learning

Temporal Difference learning is a method that combines ideas from Monte Carlo methods and dynamic programming. It updates value estimates using bootstrapping:

$$V(s) \leftarrow V(s) + \alpha[R + \gamma V(s') - V(s)] \tag{2.6}$$

where α is the learning rate. TD learning is fundamental in RL as it does not require the knowledge of transition probabilities and can learn directly from experience.

2.1.3.3 Q-Learning

Q-learning is an off-policy TD control algorithm. It finds the optimal action-selection policy irrespective of the policy that is currently being followed. It is defined by the update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2.7}$$

Q-learning is widely used as it allows learning of an optimal policy without requiring a model of the environment.

While RL aims to learn an optimal policy from a given reward function, IRL seeks to determine the reward function given observed behavior. This is particularly useful in cases where expert demonstrations are available and the underlying objectives remain unknown.

Category	**Policy**	**Algorithms**
Policy Gradient [Continuous]	On-Policy	PPO
	Off-Policy	DDPG
Value-Function Approximation [Discrete]	Off-Policy	DQN
	DP	DP Methods

Table 2.1: Reinforcement Learning Methods

2.2 Inverse Reinforcement Learning

IRL is a powerful methodology due to its ability to use data from task demonstrations to create agents capable of modeling behavior without directly interfering with the task itself. IRL makes the assumption that the expert follows a policy, π_E, which is unknown to it. The learning agent observes sequences of the expert's state action pairs, referred to as trajectories, and attempts to arrive at a reward function. While the reward function is unknown, it is often structured using linear combinations of features, probabilistic distributions, or neural networks to facilitate learning. [3]

Classical IRL takes a finite set of expert task demonstrations, along with knowledge of the environment and the expert's dynamics, to infer the potential reward function guiding the expert's behavior. Several methodologies for IRL exist, categorized by their core approaches, such as margin-based optimization, entropy-based optimization, Bayesian inference, classification, and regression. Within these categories, methods can further be distinguished by the specific objective functions they employ. IRL serves as a foundational method for learning from demonstration and is critical for developing systems that infer reward structures from observed behaviors. [2]

One prominent IRL method utilizes the principle of maximum entropy (Jaynes, 1957). to resolve ambiguities in observed behaviors by generating a globally normalized distribution over decision sequences. This approach ensures performance parity with existing methods while providing clear probabilistic foundations for modeling behaviors. [15]

The main objective of IRL is to figure out the reward function behind the expert's actions, based on the observations given to it. A common approach involves first recovering the expert's cost function through IRL and then generating a policy from this cost function using RL. However, this two-step process can be indirect and computationally slow. To address this, frameworks like GAIL propose directly extracting policies from data, bypassing the need for an intermediate reward function. GAIL leverages analogies between imitation learning and generative adversarial networks, resulting in a model-free imitation learning algorithm that demonstrates significant performance gains in replicating complex behaviors within high-dimensional environments. [8] [1]

2.3 Adversarial Inverse Reinforcement Learning

The concepts related to AIRL were taken from the papers [5] and [9].

Building upon the foundations of RL and IRL, AIRL provides a powerful framework designed to infer the underlying reward function from expert demonstrations. Unlike standard IRL, which explicitly models reward functions, AIRL leverages adversarial training techniques inspired by GANs to learn robust reward functions. The method introduces a discriminator network to distinguish expert demonstrations from generated policies, forming a min-max optimization problem.

AIRL operates within the MDP framework. The reward function is modeled implicitly by a discriminator D_θ, and the policy is optimized using RL. The AIRL framework is structured as a two-player game between:

- A generator (policy network) that learns to imitate expert behavior.
- A discriminator that differentiates between expert demonstrations and generated trajectories.

The objective is to optimize the generator such that its generated trajectories become indistinguishable from expert demonstrations, thus learning an implicit reward function.

2.3.1 Discriminator and Reward Function

In adversarial learning, the discriminator $D_\theta(s, a)$ plays a crucial role in distinguishing between expert demonstrations and trajectories generated by the learned policy. It operates as a binary classifier, trained to assign high probabilities (close to 1)

to expert state-action pairs $(s, a) \sim \pi_E$ and low probabilities (close to 0) to those generated by the agent $(s, a) \sim \pi$. [7] [5]
The discriminator is formally defined as:

$$D_\theta(s, a) = \frac{\exp(f_\theta(s, a))}{\exp(f_\theta(s, a)) + \pi(a|s)} \tag{2.8}$$

Here, $f_\theta(s, a)$ is a learned function parameterized by θ, which implicitly captures the underlying reward structure of the expert's behavior. The denominator ensures normalization, balancing the influence of the learned policy $\pi(a|s)$ against the exponentiated reward estimate.

Recovered Reward Function: The reward function $R(s, a)$ is derived through a combination of discriminative learning and potential-based shaping. The core components are [5]:

$$f_{\theta,\phi}(s, s') = g_\theta(s, a) + \gamma h_\phi(s') - h_\phi(s) \tag{2.9}$$

where:

- $g_\theta(s, a) : \mathcal{S} \to \mathbb{R}$ is the primary reward predictor learned from expert demonstrations
- $h_\phi(s) : \mathcal{S} \to \mathbb{R}$ is a learned potential function that provides shaping rewards
- γ is the discount factor matching the RL problem formulation

The complete reward function combines the f function with a policy entropy term:

$$R(s, a) = f_\theta(s, a) - \log \pi(a|s) \tag{2.10}$$

This formulation has several important theoretical and practical properties:

- **Expert Alignment**:
  - The $g_\theta(s)$ component rewards states similar to expert demonstrations
  - The difference $\gamma h_\phi(s') - h_\phi(s)$ provides consistency in shaping the reward
- **Exploration Management**:
  - The $-\log \pi(a|s)$ term serves as an entropy regularizer
  - Prevents premature convergence to suboptimal policies
  - Maintains adequate exploration throughout training
- **Dynamic Adaptation**:
  - The reward automatically adjusts as π improves
  - Ensures stable learning across different policy stages
- **Policy Invariance**:
  - Potential-based shaping preserves optimal policies
  - Guaranteed by [11] framework

Training the Discriminator: The discriminator is optimized via cross-entropy loss, a standard objective for binary classification [4] [9]:

$$L_D = -\mathbb{E}_{(s,a)\sim\pi_E}[\log D_\theta(s,a)] - \mathbb{E}_{(s,a)\sim\pi}[\log(1 - D_\theta(s,a))] \tag{2.11}$$

where:

- The first term maximizes the likelihood of expert samples being classified correctly.
- The second term does the same for generated samples, pushing the discriminator to reject non-expert behavior.

As training progresses, the discriminator's feedback refines the reward function, which in turn steers the policy π toward expert-like performance. This adversarial interplay is central to frameworks like GAIL, where the discriminator and policy are optimized alternately until convergence.

2.3.2 Actor-Critic Training

The actor-critic framework represents a fundamental approach in RL that combines the strengths of both policy-based (actor) and value-based (critic) methods. This architecture enables more stable and efficient learning compared to pure policy gradient methods, particularly in complex environments with high-dimensional state and action spaces.

The actor-critic system consists of two key components:

- **Actor**: The policy $\pi_\theta(a|s)$ that selects actions based on the current state
- **Critic**: The value function $V_\phi(s)$ that evaluates the quality of states

These components work together, where the actor proposes actions while the critic provides feedback about their quality, creating a continuous improvement loop.

Advantage Estimation: The advantage function $A(s,a)$ measures how much better an action is compared to the average expected return from state s [14] [9]:

$$\hat{A}(s_t, a_t) = r_t + \gamma Q(s_{t+1}) - Q(s_t) \tag{2.12}$$

Generalized Advantage Estimation (GAE): To reduce variance while maintaining reasonable bias, GAE is used, which combines multi-step advantage estimates [12] [9]:

$$\hat{A}_t = R_t(s) + \gamma Q(s_{t+1}) - Q(s_t) + \gamma\lambda\hat{A}_{t+1} \tag{2.13}$$

where $\lambda \in [0,1]$ controls the bias-variance tradeoff:

- $\lambda = 0$: High bias, low variance (single-step TD)
- $\lambda = 1$: Low bias, high variance (Monte Carlo)

Critic Loss Function: The critic is trained to minimize the discrepancy between the estimated action-value function and the target advantage values. The loss function is defined as:

$$\mathcal{L}_Q = \frac{1}{n} \sum_{i=0}^{n} \left(Q(s_i, a_i) - \hat{A}_i \right)^2 \tag{2.14}$$

Policy Optimization: The actor updates its policy using the advantage estimates from the critic.To ensure stable updates,a clipped surrogate loss is used [13]:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon \right) \hat{A}(s,a) \right) \right] \tag{2.15}$$

where ϵ determines the clipping range.

Surrogate losses constrain policy updates to ensure stability and prevent drastic changes in the agent's behavior. This guides the improvement of the policy by maximizing the expected advantage of actions, ensuring that the agent favors high-reward strategies while staying within stable update boundaries.

2.3.2.1 Training Dynamics

The complete training procedure alternates between:

- Collecting trajectories using the current policy
- Computing advantages and value targets
- Updating the critic to minimize value estimation error
- Updating the actor to maximize the clipped objective

This approach combines the sample efficiency of value-based methods with the flexibility of policy-based methods, while maintaining training stability through careful control of policy updates.

2.3.3 Optimization and Training Process

The training follows the below framework:

Algorithm 1 Training Adversarial Inverse Reinforcement Learning (SAIRL)

Input: Expert trajectories $\mathcal{D}_E$ (states, actions, dones)
Initialize: Generator G, Discriminator D, Environment $\mathcal{E}$, Replay buffer $\mathcal{R}$
for each epoch $e = 1, 2, \ldots,$ num_epochs **do**
    Sample leg trajectory T and corresponding expert data $\mathcal{D}_E^T$
    Reset environment state $s \leftarrow \mathcal{E}.\text{reset}(T)$
    Initialize empty rollout buffers: $\mathcal{B}_{\text{policy}}$
    while not done **do**
        Sample action $a \sim G(s)$ and log probability $\log \pi_\theta(a|s)$
        Apply action a to environment: $(s', r, \text{done}, \text{info}) \leftarrow \mathcal{E}.\text{step}(a)$
        Store $(s, a, s', r, \log \pi_\theta(a|s), \text{done})$ in $\mathcal{B}_{\text{policy}}$
        $s \leftarrow s'$
    end while
    Add rollout buffer $\mathcal{B}_{\text{policy}}$ to replay buffer $\mathcal{R}$
    for discriminator update step $d = 1, 2, \ldots,$ d_steps **do**
        Sample policy batch $\mathcal{B}_{\text{policy}}$ from replay buffer $\mathcal{R}$
        Sample expert batch $\mathcal{B}_{\text{expert}}$ of same size from $\mathcal{D}_E$
        Update discriminator D using $\mathcal{B}_{\text{policy}}$ and $\mathcal{B}_{\text{expert}}$
    end for
    Compute discriminator rewards R_D
    for generator update step $g = 1, 2, \ldots,$ g_steps **do**
        Update generator G using $\mathcal{B}_{\text{policy}}$ and rewards R_D
    end for
end for

3
Methods

3.1 Problem Formulation

Identifying ways to create optimal strategies by analyzing expert behavior without having to know the underlying reward structure before implementation is one of the primary challenges in learning-based control approaches such as IRL and imitation learning. In order to address this issue, AIRL uses adversarial training approaches to simultaneously learn a reward function and a control policy. It improves upon conventional IRL by adding a discriminator that differentiates expert behaviours from those generated by the learnt policy. As a result, it is more adept at optimizing policies and recovering accurate rewards [5] and [9].

AIRL works through two connected processes: figuring out the reward and generating a good optimal policy. The discriminator network is trained to tell apart state-action pairs from expert demonstrations and those generated by the policy being learned. In the meantime, the policy network tries to fool the discriminator into believing that its outputs mimic expert behavior while optimizing itself to maximize the reward. The system gradually learns an appropriate reward function and an efficient control policy that can even work well in scenarios outside of the demonstration data, thanks to the continuous push and pull between the two.

However, the formulation comes with its own set of challenges. Assigning appropriate parameters to learn the reward function can be challenging, and maintaining training stability in an adversarial context is not an easy task. The discriminator must possess sufficient precision to detect significant variations among trajectories while avoiding pitfalls such as detecting redundant patterns. However, even in high-dimensional, complicated action spaces with intricate dynamics, the policy must efficiently maximize the rewards. Overcoming these obstacles is crucial for creating a control policy that, with the aid of the optimum reward structure, not only imitates expert behavior but also flexibly adjusts to new scenarios.

3.2 Data Processing

The purpose of data processing was to transform unprocessed, noisy vessel data into a clear, standardized structure suitable for the AIRL application. Every stage of the procedure was designed to address certain issues with the preparation and quality of the data.

Filtering and Selecting Relevant Data: Only the most relevant navigation parameters were extracted from the dataset. Five essential features—latitude, longitude, velocity, true course, and true heading were chosen from the approximately 60 data points in the raw files. Additionally, only instances where the vessel was in 'cruising' state were included, as this operational mode represents the primary focus of the study.

$$States = \{Latitude, Longitude, Velocity, Course, Heading\} = \{\alpha, \lambda, v, \phi, \theta\} \quad (3.1)$$

Due to the unavailability of vessel parameters like mass, hydrodynamic damping, and other physical and powertrain data, a kinematic model was used therefore, the choice of features are as above.

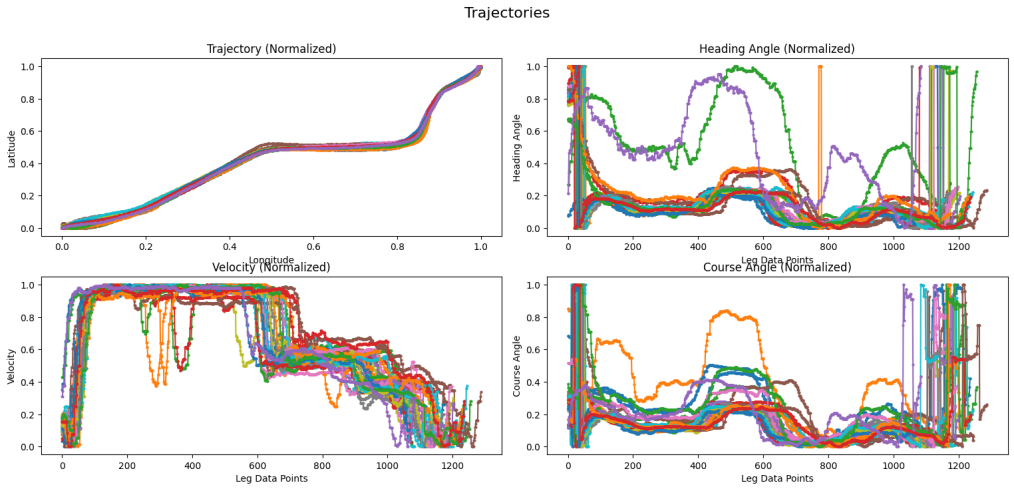


Figure 3.1: Raw Vessel data - Training

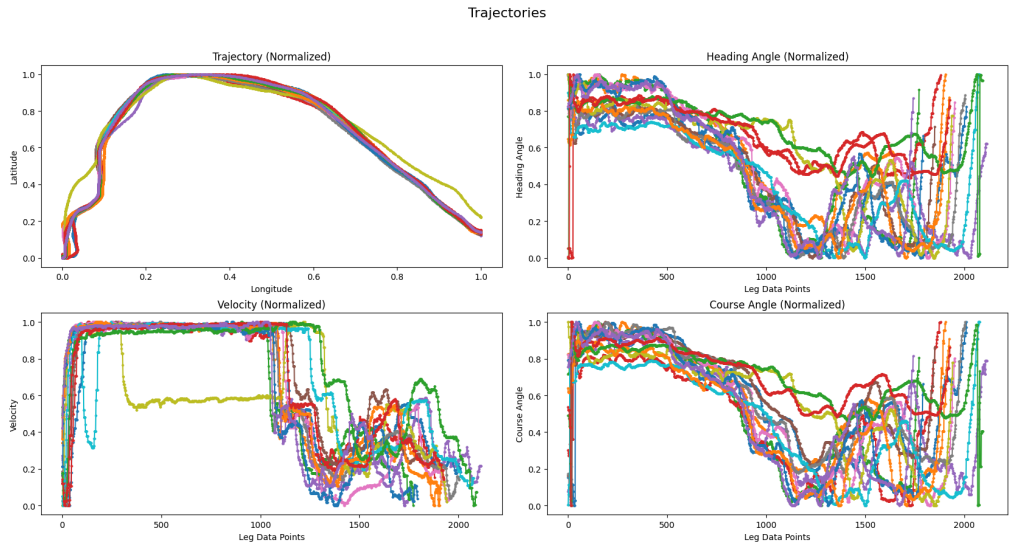


Figure 3.2: Raw Vessel data - Testing

Segmenting and Trimming Trajectories: Each continuous journey is uniquely identified by its `leg_id`. The initial and last 100 data points of each trajectory were eliminated to reduce the effect of anomalies that might show up at the beginning or end of each trip of the vessel. Since stable data segments are the most useful for comprehending control policies, this trim made sure that enough focus was given to them.

Normalizing Angular Data: Given the circular nature of the course and heading measurements, these values were transformed into modulo 360° space. This prevented artificial jumps across the 0°/360° boundary. For instance, a transition from 359° to 1° was interpreted correctly as a 2° shift, rather than an abrupt change of 358° in the opposite direction.

Noise Reduction through Signal Processing: Signal processing techniques were used to remove noise from the latitude and longitude coordinates while maintaining realistic vessel movements. A second-order, zero-phase, forward-backward Butterworth low-pass filter with a cutoff frequency of 0.03 Hz was used. The design of this filter was based on the FFT analysis performed on the data. This strategy provided several benefits:

- Preserved the true trajectory of the vessel by enabling the Butterworth filter's flat passband characteristic.
- Reduced GPS noise and high-frequency oscillations without compromising vessel motion data.

Creating Derived Features: To reflect the vessel's control actions, two key features were calculated:

- **Delta velocity**: The difference in velocity between consecutive data points, which corresponds to acceleration or deceleration commands. The data was sampled at 1Hz, so delta velocity and acceleration are the same numerically.
- **Delta heading**: The angular change between consecutive heading values, representing steering adjustments.

$$Actions = \{\text{Change in velocity}, \text{Change in heading}\} = \{\Delta v, \Delta\theta\} \tag{3.2}$$

These derived features transformed the raw state data into meaningful inputs that represent control actions tied to decision-making processes.

Data Normalization: Predefined maximum and minimum values of each feature were normalized to a range of $[-1, 1]$. This normalisation was crucial to improve model robustness, speed up convergence, improve numerical stability during model training, and ensure uniform feature contribution across a range of scales.

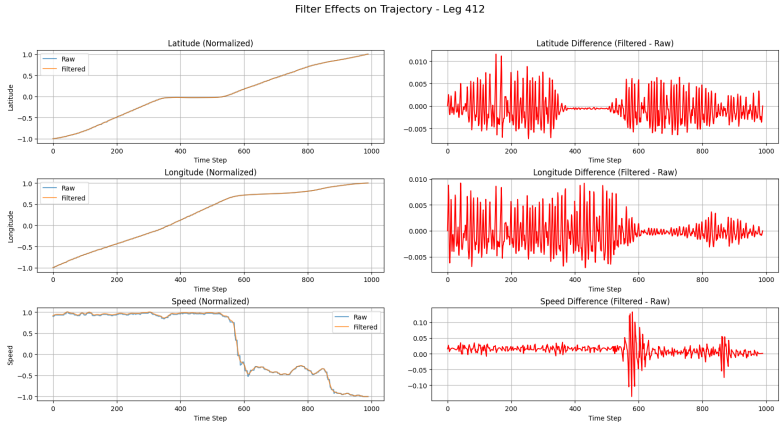


Figure 3.3: Buterworth Filter for GPS data

Marking Terminal States: To accurately handle episode termination during reinforcement learning, a 'done' flag was added to indicate the last timestep of each trajectory. This step was particularly important for ensuring correct value function estimation and temporal difference updates. Whenever the vessel went out of a predefined boundary, reached the destination, or exceeded the time steps, the value of done would be set to 1.

Visualization: To examine the processed data at each step, thorough visualisations were made, confirming that all variables were consistently normalised, trajectories were smooth, angular measurements were continuous, and derived features were calculated. These assessments were essential in detecting and resolving any irregularities hindering learning objectives.

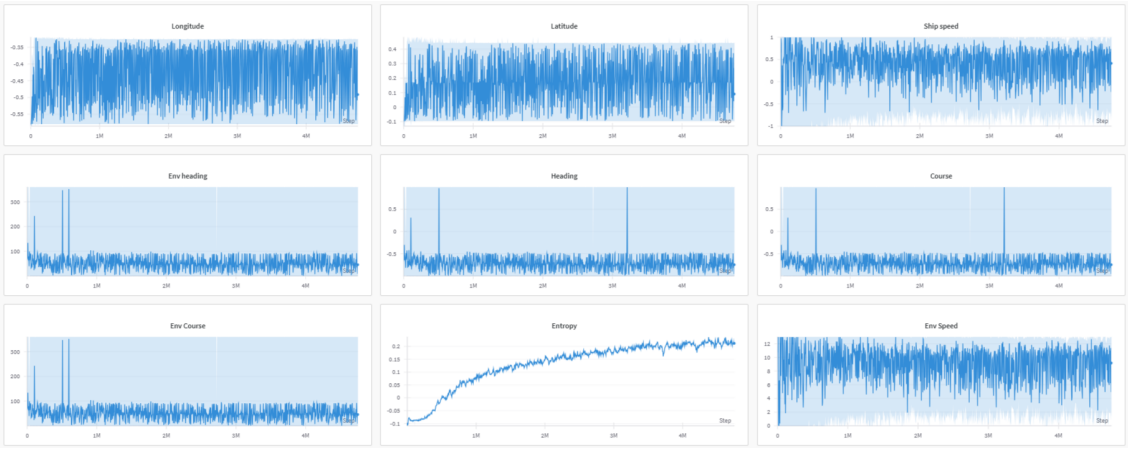


Figure 3.4: Logging data to Wandb (Light blue represents the range of minimum to maximum values for states and actions)

This pipeline transformed raw, noisy sensor data into good-quality, well-structured trajectory data. Each processing step was carefully designed to preserve the integrity of the original data while maximizing its utility to develop a control policy.

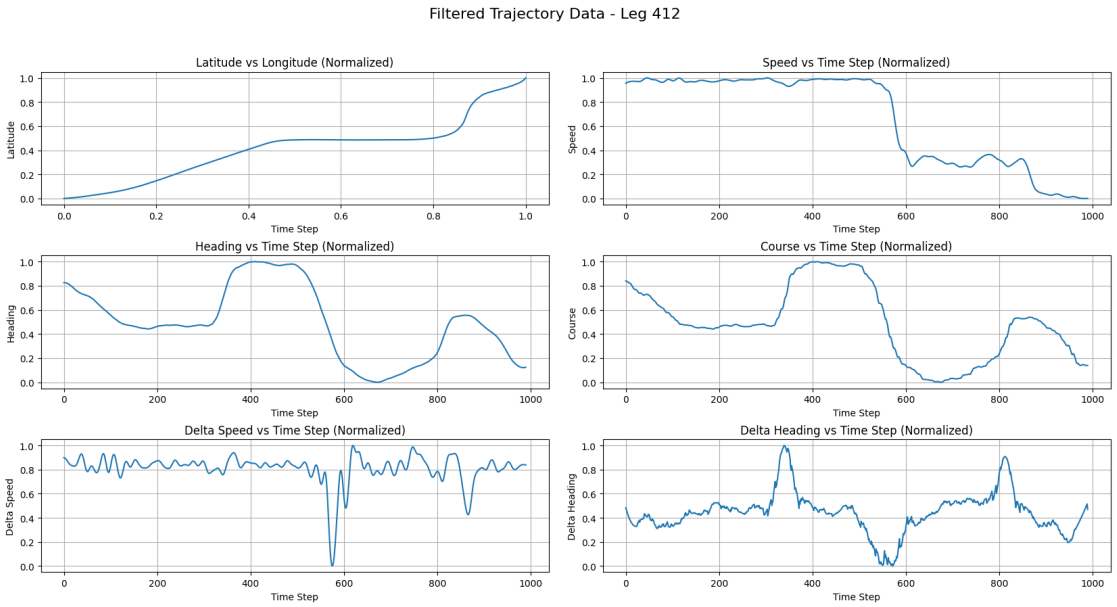


Figure 3.5: Expert Demonstration used for training

3.3 Formulation of AIRL

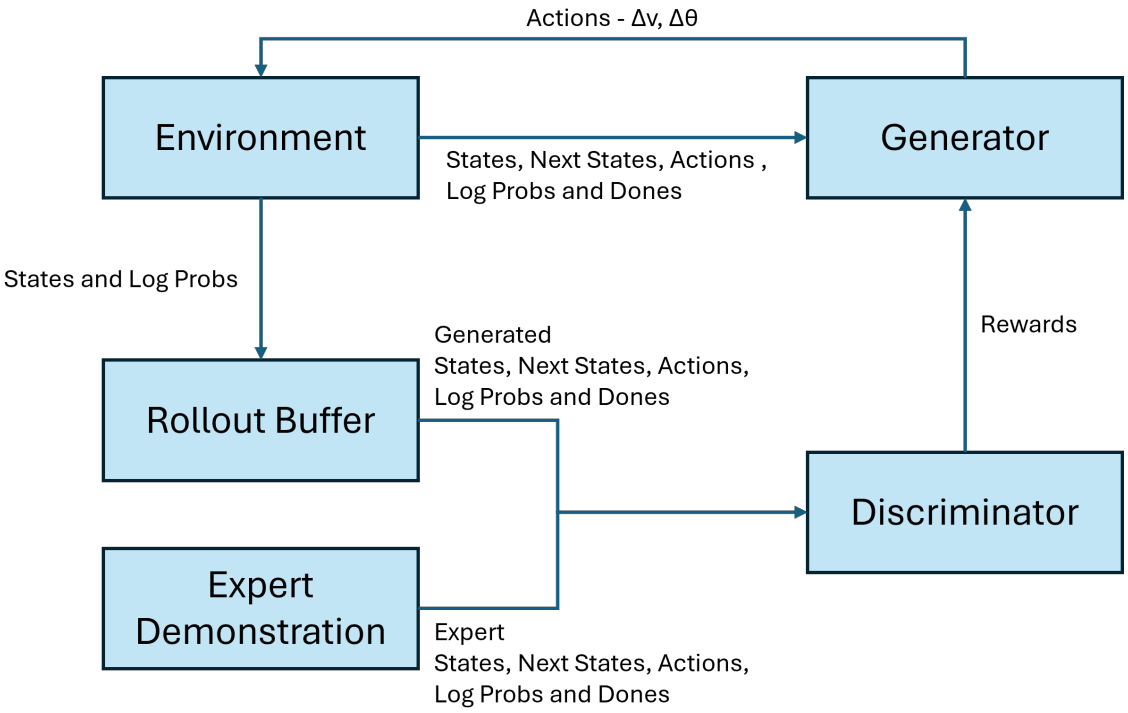


Figure 3.6: AIRL flow-chart

3.3.1 Environment

Using the OpenAI Gym package, a custom environment was developed. This particular environment, which includes the simulation of the vessel, successfully depicts the actual vessel navigation using the kinematic equations given below.

This environment's state representation is intended to give a thorough picture of the vessel's operational condition. Latitude and longitude for geographic location, vessel velocity, ground course, and vessel heading are the five key navigational elements. In order to restrict the exploration space for the vessel in the environment, each episode creates a buffered boundary around the expert trajectory with a margin, set by default to 300 meters. This buffer is dynamically computed by constructing a polygon around the expert path. 36 simulated radar distance measurements are also included in the environment, creating a 180-degree forward-facing detection field with a 5-degree resolution. By projecting geodesic rays from the vessel's current position and detecting intersections with the navigable boundary polygon, these radar sensors make realistic boundary detection possible. This enables precise geometric calculations, making it possible to detect when boundaries are crossed and measure the minimum distances accurately. This feature is vital for developing collision-avoidance and maintaining safe navigation corridors.

The continuous action space reflects real-world vessel control mechanisms: delta velocity and delta heading. Both actions are normalized to the range of [-1,1], allowing for flexibility across different vessel types and operational conditions.

$$v_{t+1} = \text{clip}(v_t + \Delta v, v_{\min}, v_{\max}) \tag{3.3}$$

$$\theta_{t+1} = (\theta_t + \Delta\theta) \tag{3.4}$$

$$\phi_t = \theta_t + \epsilon \tag{3.5}$$

$$d_t = v_t \cdot \Delta t \tag{3.6}$$

$$\delta_t = \frac{d_t}{R} \tag{3.7}$$

$$\alpha_{t+1} = \arcsin\left(\sin(\alpha_t)\cos(\delta_t) + \cos(\alpha_t)\sin(\delta_t)\cos(\theta_t)\right) \tag{3.8}$$

$$\lambda_{t+1} = \lambda_t + \arctan 2\left(\sin(\theta_t)\sin(\delta_t)\cos(\alpha_t), \cos(\delta_t) - \sin(\alpha_t)\sin(\alpha_{t+1})\right) \tag{3.9}$$

Category	Variable	Description	Unit
State	$v_t,\ v_{t+1}$	Velocity at time $t,\ t+1$	m/s
	$\theta_t,\ \theta_{t+1},$	Heading angle (azimuth)	degrees (°)
	ϕ	Course angle (azimuth)	degrees (°)
	$\alpha_t,\ \alpha_{t+1}$	Latitude at time $t,\ t+1$	degrees (°)
	$\lambda_t,\ \lambda_{t+1}$	Longitude at time $t,\ t+1$	degrees (°)
Action	Δv	Change in velocity	m/s
	$\Delta\theta$	Change in heading angle	degrees (°)
Other	$v_{\min},\ v_{\max}$	Minimum and maximum velocity	m/s
	Δt	Time step	seconds (s)
	d	Distance traveled	meters (m)
	R	Radius of the sphere (e.g., Earth)	meters (m)
	δ	Angular distance on the sphere	radians (rad)
	ϵ	Drift	degrees (°)

Table 3.1: States Actions description and their units.

The state transition model was built with geodesic calculations to update the vessel's position based on control inputs. The Geopy library was used to calculate the next position of the vessel, taking into account the Earth's ellipsoidal shape. This ensures more accurate position updates, especially in long-distance navigation scenarios. During training, the drift value was set to 0. This decision was based on observations showing that when drift was eliminated, the algorithm demonstrated greater robustness to changes during evaluation, a point that will be further discussed in the results section.

Episodes terminate under three main conditions: when the vessel successfully arrives within 50 meters of its destination, this is to accommodate the uncertainites in GPS data being collected which is assumed to have uncertainty of 10m. When it collides with the buffered boundary, or when the maximum episode duration is exceeded determined as the length of the expert trajectory plus 200 steps, the additional 200 steps are given so that the algorithm can explore more state action pairs by interacting with the environment. These conditions ensure that training progresses in a meaningful way while allowing the agent enough time to develop effective navigation policies. To handle termination cases correctly, the environment incorporates masking logic that differentiates between termination types. This prevents the agent from learning inaccurate patterns from artificial time-limit terminations and ensures that successful goal achievements are properly rewarded.

To track training progress and analyze policy behavior, the environment consists of plots that compare expert and learned trajectories based on the traced trajectories, velocity profiles, heading changes, course variations, and control inputs.

The training and performance tracking are achieved by seamless integration with the wandb platform. This setup logs the following states: velocity, heading, course,

control actions(delta velocity and delta action), and success flags at every timestep. Additionally, trajectory logs are saved, allowing for detailed post-training analysis. This logging and monitoring process has been essential for fine-tuning the learning process, identifying areas for optimization, and effectively debugging issues.

3.3.2 Discriminator

The discriminator has an important dual function in AIRL. It differentiates between expert and generated states while simultaneously learning a reward function that reflects the policy behind the expert's actions. This design addresses a major challenge in IRL: the uncertainty involved in estimating the reward function, which can often be influenced by how the environment's dynamics shape the learning process. The discriminator builds on the maximum causal entropy framework, ensuring the reward function aligns with the MDP and remains unaffected by changes in system dynamics [5].

Discriminator Architecture: The architecture of the discriminator breaks the reward estimation process into two complementary parts:

- **Reward Approximator**: This component estimates the immediate reward for state-action pairs. It processes concatenated state and action information, along with the radar measurements.

- **Shaping Term**: This part focuses on learning a function that ensures the reward function is invariant under transformations. It specifically depends only on the state, excluding actions, which helps maintain policy consistency.

Mathematical Formulation: As explained earlier in the theory section, the discriminator integrates these components into a structure resembling advantage functions in RL. Its output includes a discount factor applied to the shaping term at the next state, minus the shaping term at the current state. This formulation allows the discriminator to recover the true reward function while ensuring the shaping term converges toward the optimal value function. The training process uses a binary classification objective, teaching the discriminator to distinguish between states from expert demonstrations and those generated by the learned policy.

Training Stability Strategies: To ensure stable and effective training, the following techniques are used:

- **Spectral Normalization**: Maintains Lipschitz continuity of the neural networks, ensuring controlled gradients [10].

- **Gradient Clipping**: Prevents unstable updates during optimization [6].

- **Handling Terminal States**: Avoids assigning misleading rewards at the end

of episodes.

- **Dropout Regularization**:Prevents overfitting

With this carefully designed architecture and training approach, AIRL's discriminator fulfills two critical objectives: it provides a meaningful signal to guide policy optimization during training, and it develops a robust reward function that captures the expert behavior. This dual functionality addresses persistent challenges in reward ambiguity and transfer learning, marking a significant leap forward in the field of IRL.

3.3.3 Generator

The generator component of the AIRL framework is built on the PPO algorithm. It consists of two neural networks: an actor network, which defines the control policy, and a critic network, which estimates state values. These networks are designed to handle the continuous action space required for vessel maneuvering.

The actor network produces parameters for a Gaussian distribution, specifically, the mean and log standard deviation for each control action (delta velocity and delta heading). This stochastic policy exploration is crucial in dynamic maritime environments. Reparameterization is used to enable differentiable sampling, ensuring smooth gradient flow during training. The sampled actions are passed through a tanh activation function, restricting them to the range of $[-1, 1]$ needed for the control system.

Network Architecture: The generator has three hidden layers with LeakyReLU activations. This architecture was selected after extensive empirical testing to strike a balance between representational capacity and training efficiency for navigation tasks. The critic network mirrors this structure but focuses on estimating state values instead of generating distribution parameters for the actions. Weight decay regularization is applied at rates of 3×10^{-4} for the actor and 7×10^{-4} for the critic to prevent overfitting. Additionally, gradient clipping with a maximum norm of 0.5 is employed to ensure stable training, which is particularly important given the adversarial setting of AIRL. The critic's greater vulnerability to overfitting during value estimation is reflected in the stronger regularization that is given to it.

Policy Updates: Policy updates leverage GAE with parameters $\gamma = 0.99$ and $\lambda = 0.95$, chosen to balance bias and variance in the advantage estimates. To maintain consistent scaling during training, advantages are normalized within each batch. The PPO clipped surrogate objective is designed to keep updates in check, avoiding overly large policy changes that might disrupt training while ensuring sample efficiency.

The training process alternates between policy improvement through actor updates

and value function refinement via critic updates. This synchronized approach ensures stable training when competing against the discriminator.

By incorporating these elements, the generator in AIRL achieves robust policy optimization for marine navigation. The combination of stochastic exploration, domain-specific adjustments, and careful synchronization with the discriminator enables the system to develop a control policy that can handle the complex dynamics of vessel navigation. It effectively responds to reward signals learned from the discriminator, resulting in a policy that is both flexible and capable of adapting to new scenarios.

3.3.4 Training

The training process for the AIRL system is organized as a structured pipeline that integrates the generator, discriminator, and the environment. The process begins by initializing key components, including the environment, the generator's policy network, and the discriminator. Expert demonstration data, consisting of vessel trajectories and their corresponding state-action pairs, is then loaded. A replay buffer is established to store and sample policy-generated transitions, using a first-in-first-out strategy with a configurable maximum capacity to ensure the storage of recent and relevant experiences. Each training loop iterates over a predefined number of epochs, starting with the selection of a vessel trajectory (`leg_id`) from the expert demonstrations. This approach ensures exposure to a variety of navigation scenarios during training.

Episode Rollouts: During each epoch, the agent interacts with the marine environment by completing a full episode rollout. In this phase, the current policy generates actions for each state, with the resulting transitions being stored. The environment processes these actions through its vessel dynamics model, which returns the next states. Comprehensive logging during this rollout captures navigation metrics such as position coordinates, velocity changes, and heading adjustments. These logs provide a detailed view of the policy's behavior throughout training. The collected trajectories are stored in the replay buffer along with the computed log probabilities of the actions taken, forming a dataset used for training the discriminator.

Discriminator Updates: The discriminator training phase involves balanced updates using both expert demonstrations and policy-generated trajectories. Batches are sampled from the replay buffer and paired with corresponding expert samples to ensure balanced training. The discriminator learns to differentiate between expert and policy trajectories, while simultaneously recovering a reward function through its dual-network architecture. Training utilizes a binary logistic regression loss with gradient clipping and spectral normalization, which help maintain stability during learning.

Generator Updates: After the discriminator updates, the generator's policy is refined using PPO. The rewards used for policy improvement are derived from the

discriminator's evaluations. GAE is applied to handle reward assignment effectively.

Marine-Specific Features: A radar scanning system provides environmental awareness by simulating distance measurements to navigational boundaries, which are integrated into the state representation. Logging tools capture key training metrics, such as discriminator loss, policy performance, and policy entropy, to provide insights into learning progress.

Monitoring and Debugging: Periodic model checkpoints and trajectory visualizations allow for detailed progress tracking and debugging. The visualizations compare the expert trajectories with the policy-generated ones, showcasing the navigated path, velocity, and angular profiles, and action profiles.

Experiment Tracking: The entire training pipeline is instrumented using the Weights & Biases (wandb) platform. This setup captures hyperparameters, learning curves, navigation performance metrics, and other key indicators throughout the training duration. This comprehensive monitoring framework provides an in-depth understanding of how the system evolves during training and ensures reproducibility of results.

The structured training pipeline effectively coordinates the generator, discriminator, and marine environment, enabling the AIRL system to learn a robust control policy for vessel navigation. By integrating domain-specific enhancements, detailed monitoring, and advanced training techniques, the system is well-suited for addressing the complexities of real-world maritime scenarios.

Table 3.2: Hyperparameters used in the AIRL setup

Parameter	Value
Actor learning rate	3×10^{-5}
Critic learning rate	3×10^{-5}
Discriminator learning rate	5×10^{-5}
GAE gamma	0.99
GAE lambda	0.98
Generator loss epsilon	0.15
Generator discount factor	0.999
Entropy coefficient	0.075
Discriminator discount factor	0.995
Buffer size	10000
Seed value	3
Batch size	128
Number of epochs	6000
D steps in training	2
G steps in training	10
Boundary buffer	300

3.4 Software Library

Pandas: The pandas package is a Python library for data manipulation and analysis. It provides flexible data structures like DataFrames and Series, which make it easy to handle and process structured data. Pandas is widely used for tasks such as data cleaning, transformation, and exploration.

PyTorch: The PyTorch package is a deep learning framework that provides tools for building and training neural networks. It offers dynamic computation graphs, which allow for flexible model building and debugging. PyTorch also integrates well with other libraries and is commonly used in reinforcement learning projects.

Shapely: The Shapely package is a Python library for creating and analyzing planar geometric objects. It allows users to work with shapes like points, lines, and polygons to perform spatial operations. In the thesis, Shapely is used to model vessel trajectories and create buffer zones. It checks if the vessel stays within these zones and calculates intersections for distance sensing. This helps enforce navigation boundaries and detect proximity to obstacles.

WandB:
The Weights & Biases (wandb) package is a tool for tracking machine learning algorithms and visualizing results. It helps log metrics, model parameters, and outputs in real time, making it easier to monitor training progress and compare runs. Wandb is also used to save performance data, generate interactive plots, and share results seamlessly. This makes algorithm management more organized.

4

Results and Discussion

Once the policy converged, the generator and discriminator models were saved and later tested on various trajectories and dynamic conditions. The main focus of the evaluation was to analyze how well the policy performed when faced with disturbances and to assess its ability to adapt to new, unfamiliar trajectories and dynamic scenarios. The training was specifically conducted using leg ID 412.

4.1 Training Results

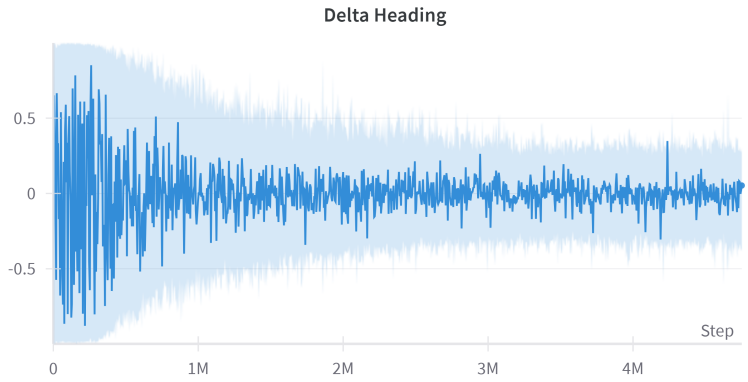


Figure 4.1: Action: Delta Heading

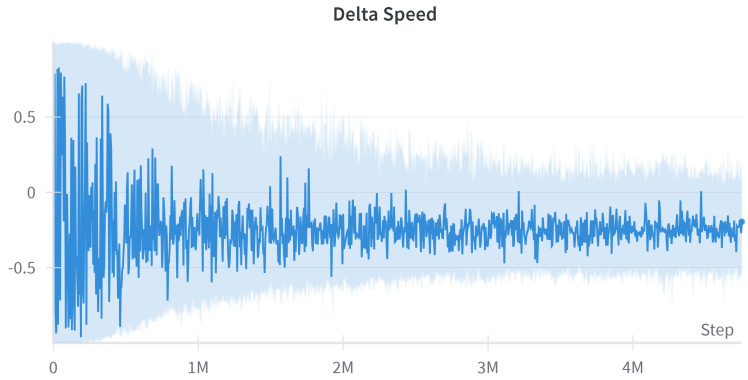


Figure 4.2: Action: Delta Velocity

Action Convergence: Figures 4.1 and 4.2 show that as the training progress the actions generated from the policy which maximizes the reward approximated from the discriminator are converging to values similar to what the expert had taken. This tells us that the learning process was converging and was stable. Additionally, the actor and critic loss also tells the same about the stability of the learning process, which is addressed later.

Total reward: The total reward starts high, peaking above 800, but drops and stabilizes around 400 as the training progresses. This initial spike reflects early exploration before the policy settles into more consistent, learned behavior. After the sharp decline, the reward remains fairly stable over the period of the training, with small fluctuations. This suggests the policy stabilizes where it reliably performs at a moderate level. The stable total reward indicates that the agent has converged to a consistent policy and is not diverging or collapsing.

Figure 4.3: Training reward

Actor loss: PPO stabilizes training by using a clipped surrogate loss that limits how much the policy can change at each update. This helps ensure improvements in performance without letting the policy drift too far from the current one. To keep the policy exploring, an entropy bonus is also added, which rewards policies that stay stochastic rather than becoming overly deterministic too quickly. Early in training, the policy is still exploring a lot, so the entropy is high and the advantages are large, which makes the policy updates strong and the actor loss relatively low. As training progresses, the policy becomes more confident, and the advantages flatten out because there's less room left to improve. As a result, the learning weakens and the actor loss increases, reflecting the fact that updates are getting smaller and the policy is stabilizing.

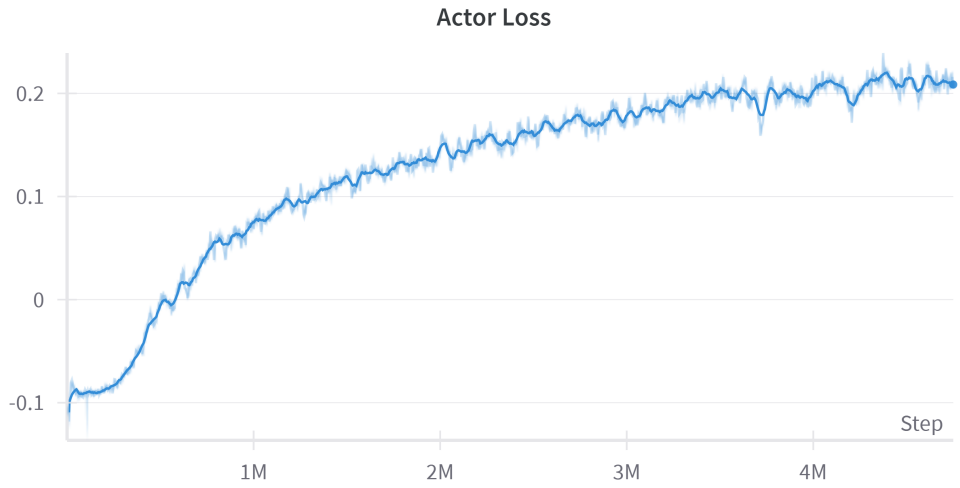


Figure 4.4: Actor loss during the training

Discriminator loss: The discriminator loss starts high (around 1.7), then drops and stabilizes in the range of 0.8 to 0.9. This suggests that at the beginning, the discriminator had difficulty distinguishing between expert and generated actions, but it learned to improve its classification. The fact that it settles around 0.8 indicates it's maintaining a good level of uncertainty, which is desirable in adversarial training. It prevents the model from overfitting to either the expert's or the policy's behavior.

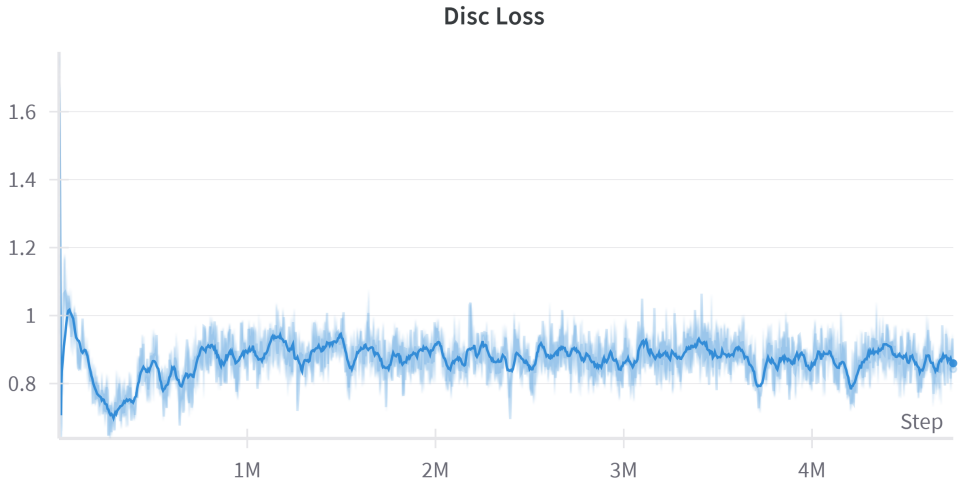
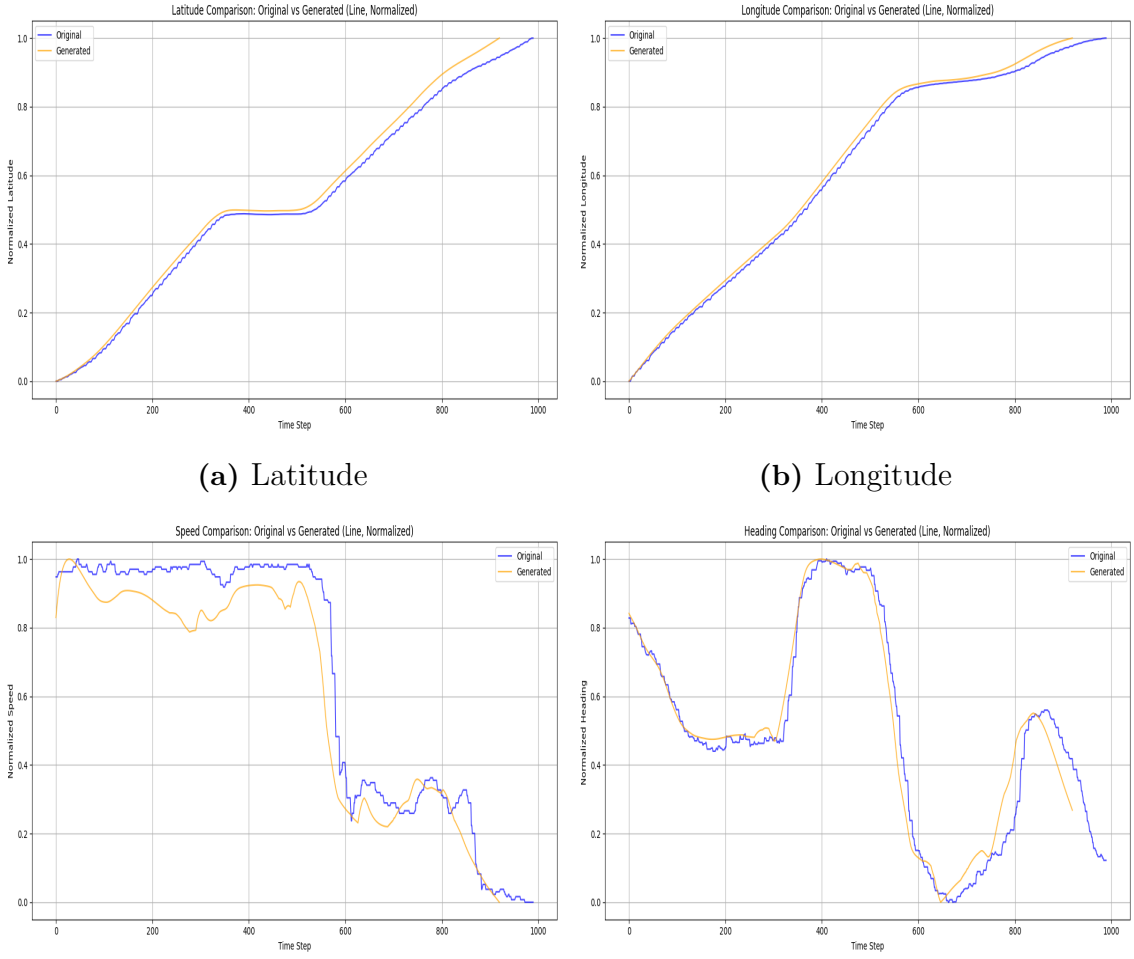


Figure 4.5: Discriminator loss during the training

4.2 Evaluation - Ideal conditions

The training and evaluation environments had identical dynamics, with no disturbances affecting the vessel's movement. As shown in the figure, the trajectory produced by the optimal policy closely follows the expert demonstration provided by the

captain. The key states—latitude, longitude, velocity, course, and heading—align well with the captain's policy, as seen in figure 4.6. However, the velocity shows a relatively higher error compared to the other states, and the reason for this will be discussed in later sections.



(a) Latitude **(b)** Longitude

(c) Velocity **(d)** Heading

Figure 4.6: Comparison of predicted and expert states under ideal conditions

4.3 Evaluation - With disturbances

With the policy that demonstrates strong performance under ideal conditions, disturbances were introduced to evaluate its robustness. To simulate the effects of environmental factors such as wind, current, and waves, two additional terms were incorporated into the environment model. The first was a drift term, representing the angular difference between the vessel's heading and its actual course over ground. The second was a velocity disturbance, modeled by introducing a random perturbation within a specified range to the vessel's velocity. Using these disturbance models, the optimal policy was tested to assess its stability and adaptability under non-ideal conditions. The evaluation results are presented as follows.

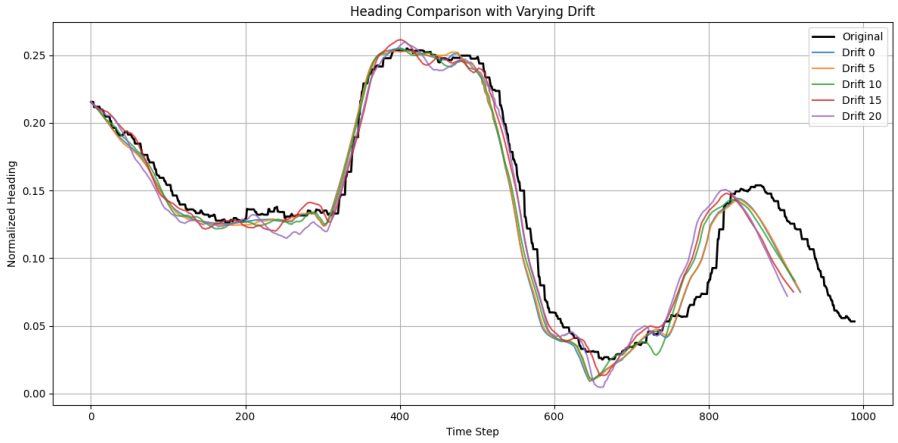


Figure 4.7: Comparison: Heading for varying drift

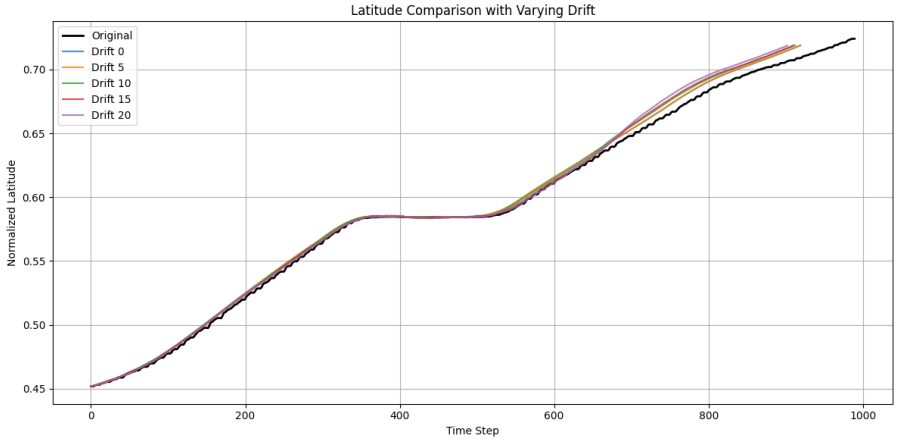


Figure 4.8: Comparison: Latitude for varying drift

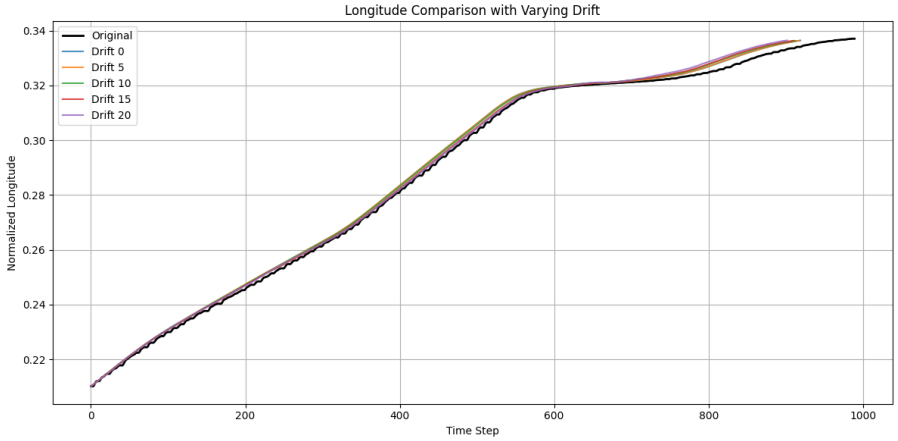


Figure 4.9: Comparison: Longitude for varying drift

The policy was tested under a range of artificially induced disturbances, where velocity and drift perturbations were introduced using uniformly sampled random values. Specifically, the velocity disturbance ranged from [-0.5,0.5]m/s, while the drift disturbance ranged from [-10,10] degrees. The results indicate that the policy is more robust to drift disturbances compared to velocity disturbances.

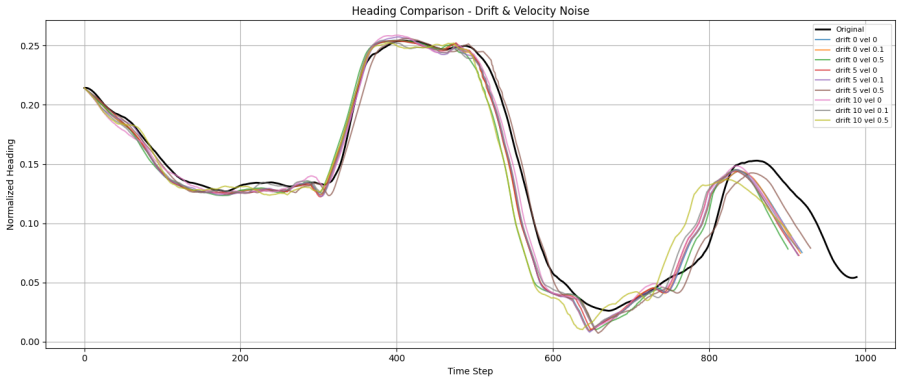


Figure 4.10: Comparison: Heading for varying drift and velocity

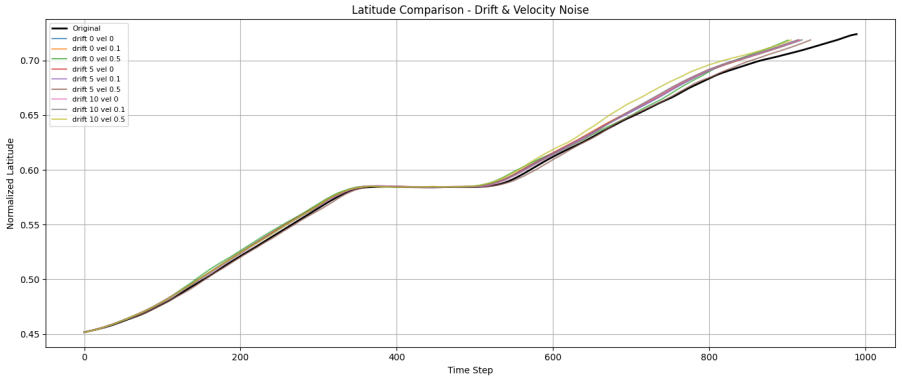


Figure 4.11: Comparison: Latitude for varying drift and velocity

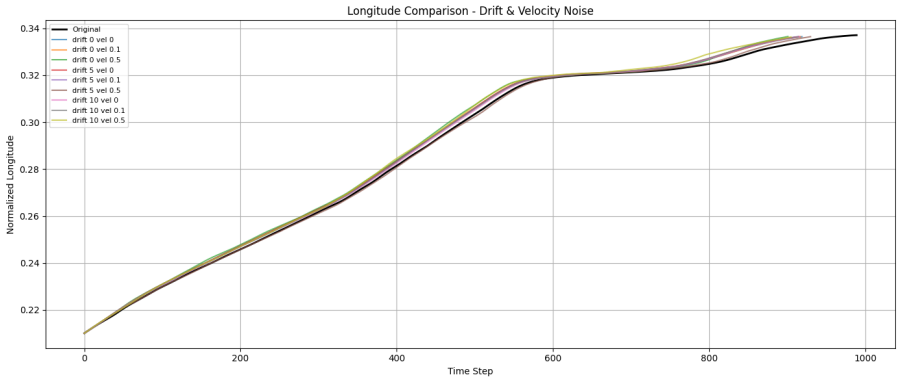


Figure 4.12: Comparison: Longitude for varying drift and velocity

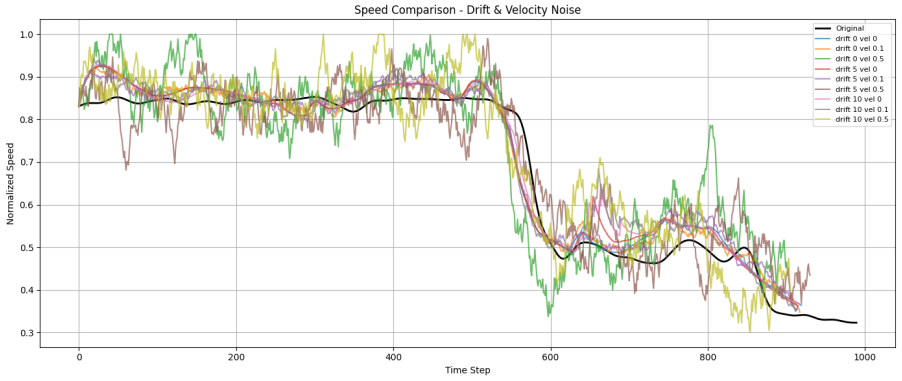
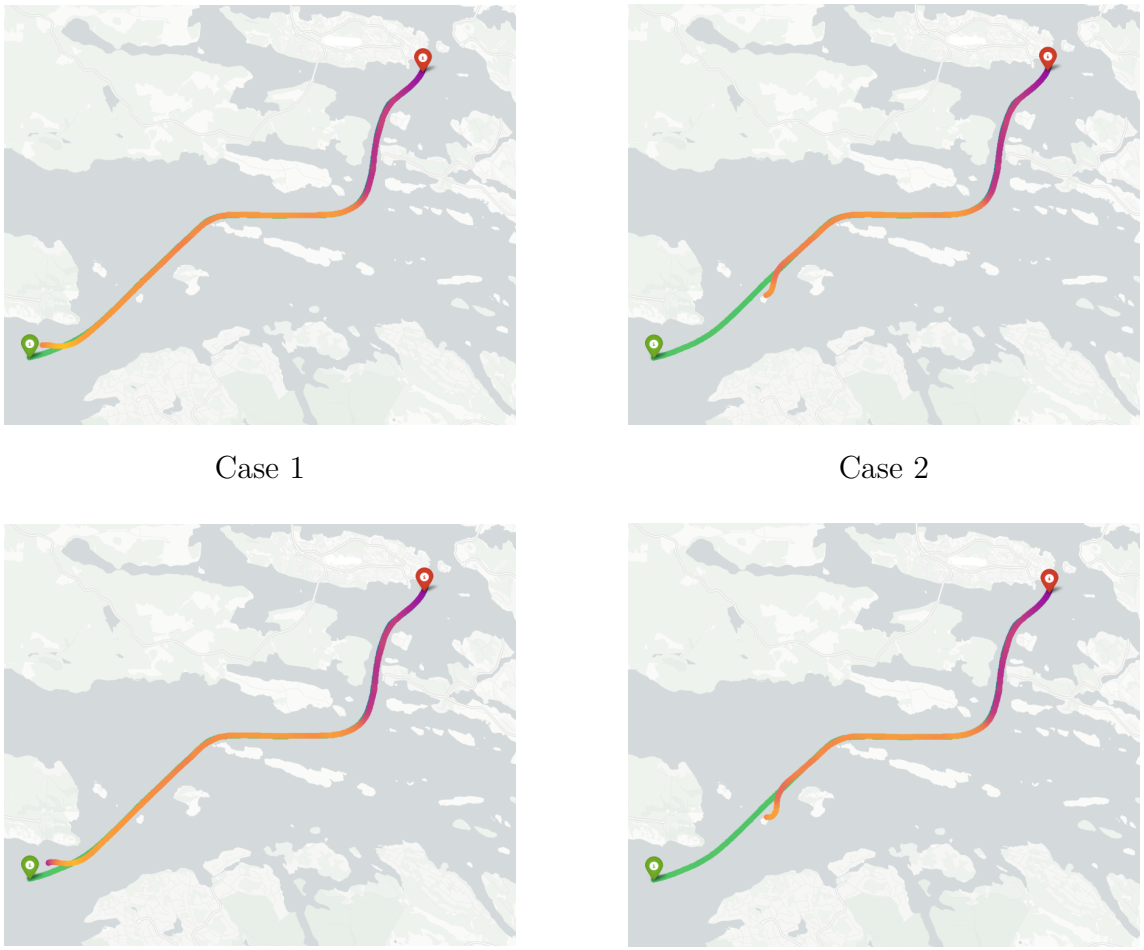


Figure 4.13: Comparison: velocity for varying drift and velocity

4.4 Evaluation - With offset initial conditions

The policy was evaluated under different initial conditions to assess whether it could still follow expert behavior, and the following observations were made.

- **Case 1**, the initial latitude and longitude were offset by 200–300 meters, and the policy successfully guided the vessel to the goal. The trajectory shows a reduction in velocity near the initial point (observed as a change in gradient), after which the vessel aligns with the expert path and follows the expected velocity profile.
- **Case 2**, the starting point was placed significantly farther from the actual initial position, yet the policy managed to reach the goal.
- **Case 3**, the initial velocity was offset by 10 m/s, and the policy accelerated the vessel appropriately to match the expert demonstration.
- **Case 4**, the initial orientation of the vessel was changed by 45 degrees, and the policy still aligned the vessel with the expert trajectory, demonstrating consistent behavior across all tested conditions.

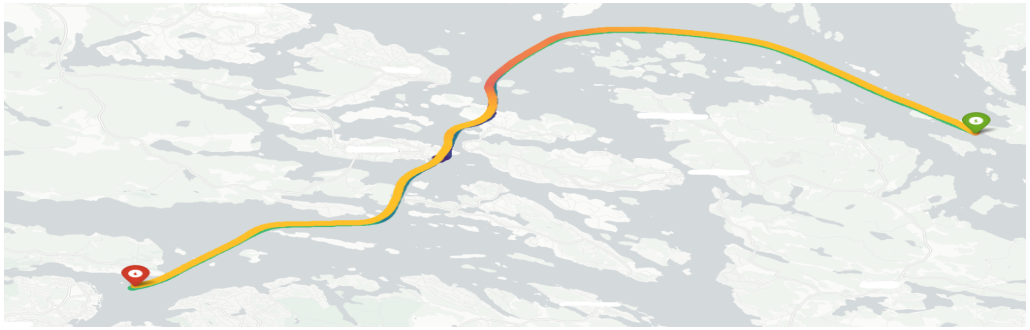


Case 1 Case 2

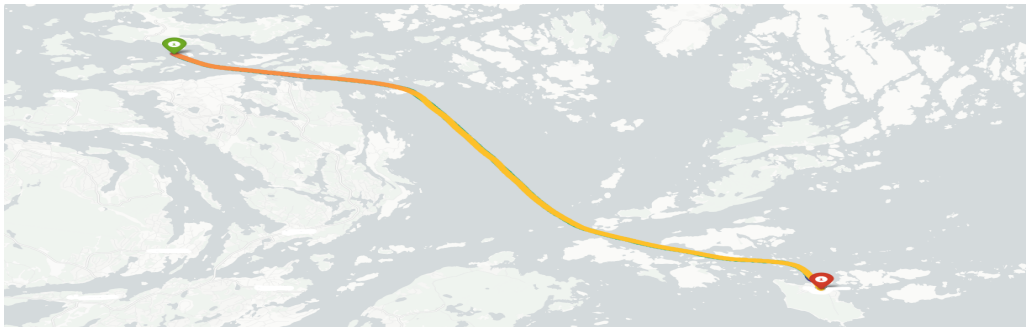
Case 3 Case 4

Figure 4.14: Different latitude, longitude, velocity and course as initial condition

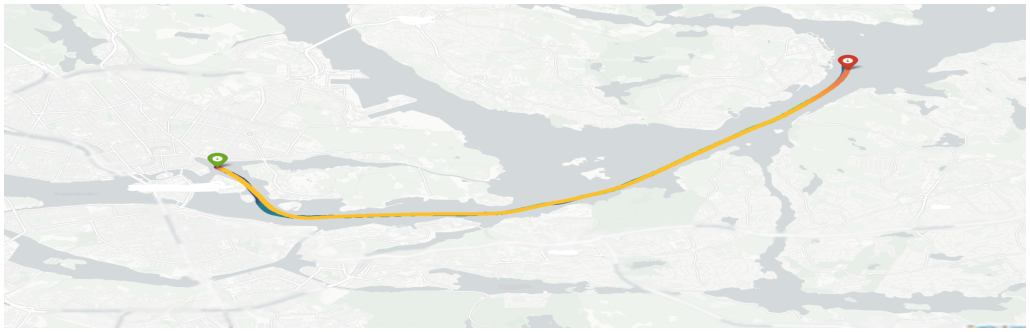
4.5 Evaluation - Unknown trajectories



(a) Test Trajectory A



(b) Test Trajectory B



(c) Test Trajectory C

(d) Test Trajectory D

Figure 4.15: Comparison of different test trajectories

The policy was evaluated on several test trajectories that were not part of the training data. In all cases, the vessel successfully reached its intended destination. The policy closely tracked the positional data (latitude and longitude), although it did not accurately reproduce the velocity profiles. This mismatch is likely due to the inconsistency in expert demonstrations, as the captain employed varying control strategies across different trajectories. The improved positional accuracy can be attributed to the inclusion of radar data in the reward function, which helps avoid obstacles, such as navigating around islands.

4.6 Plots of State Errors

The difference between the expert states and the generated states is plotted to see how well the policy follows the expert demonstration. The difference is taken at each time step, and as seen in 4.16 and 4.17, the error is around 0.001, and this corresponds to an approximate error of 100m which is equivalent to the uncertainty in the GPS sensor, so the error is acceptable and reasonable. 4.18 shows that the velocity error is relatively high when compared to other states. This is because the policy tracks the trajectory better than the velocity, and the reason for this is that the expert does not follow a fixed policy when it comes to velocity.

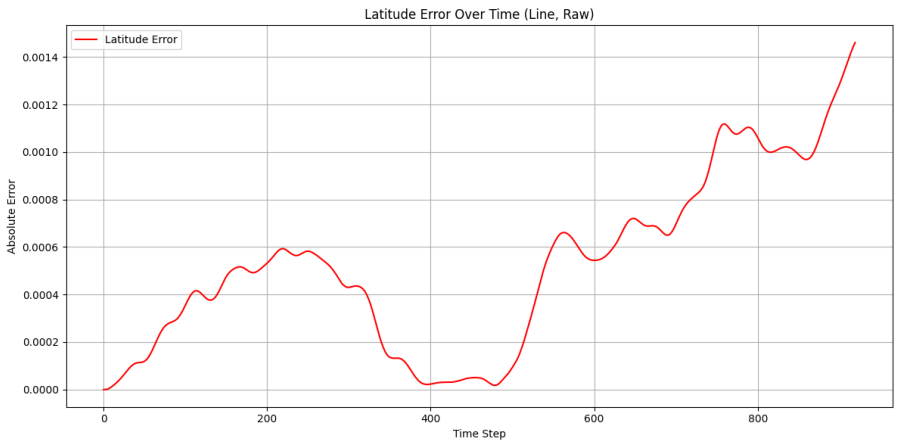


Figure 4.16: Absolute Error in Latitude

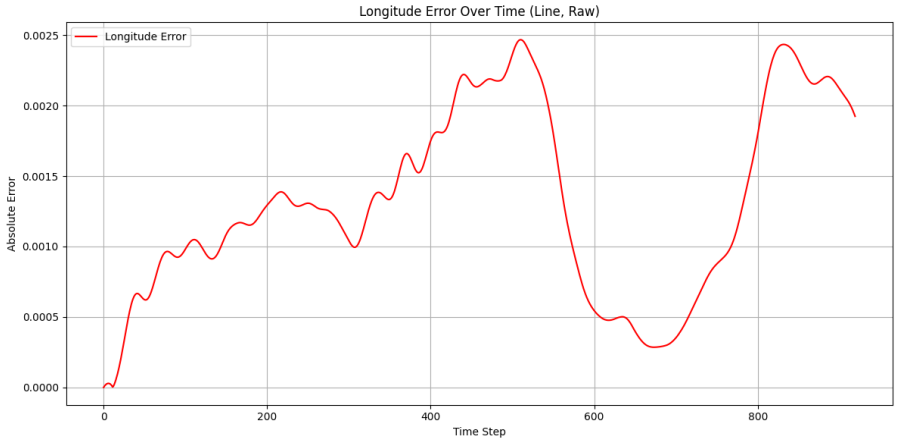


Figure 4.17: Absolute Error in Longitude

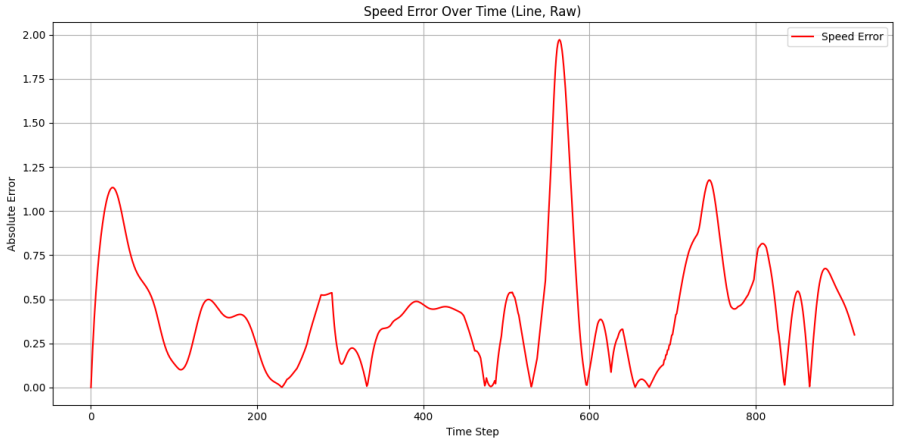


Figure 4.18: Absolute Error in velocity

4.7 Discussion

Impact of Radar Data Integration The incorporation of radar measurements as state inputs to both discriminator and generator networks significantly improved the navigation performance of the system. This enhancement provided the learning algorithm with more environmental information, leading to improvements in trajectory following and goal convergence.

The radar system provides 36 distance measurements that span a 180° forward-looking field with 5° angular resolution, creating a detailed spatial representation of the vessel's surroundings. When these measurements were incorporated into the state representation the following results were found:

- **Learning Dynamics**: The algorithm demonstrated distinct phased learning behavior:

1. *Phase 1 - Avoid going out of boundary*: Initially focused on interpreting radar signals to maintain safe distances from boundaries

2. *Phase 2 - Path Convergence*: Rapidly reduced cross-track error by aligning with expert trajectory

3. *Phase 3 - Action Optimization*: Refined control actions while maintaining the same state values as the expert demonstration.

Robustness When the model was trained in a disturbance-free environment but evaluated under conditions with significant drift and varying velocities, it was observed that the learned policy was still able to reach the destination by closely following the expert trajectories. This demonstrates robustness in the learned behavior. Furthermore, with a sufficiently large dataset and a consistent expert demonstration, it is possible to develop a more generalized and fuel-efficient policy. When disturbances were introduced during training, the model was able to learn the underlying patterns of those disturbances and adapt its behavior accordingly, indicating its potential to generalize to more dynamic and realistic navigation scenarios.

5
Conclusion and Future Work

5.1 Conclusion

In this master's thesis, an AIRL model was developed to perform vessel navigation. The AIRL model successfully navigated the vessel from its origin to its destination. It closely followed the expert's trajectory but differed in the velocity profile. This discrepancy can be attributed to the fact that the captain does not follow a fixed policy throughout; for example, there are instances where the captain slows down near certain points while cruising, which is not ideal or consistent.

When tested under disturbances at certain states, the model demonstrated robustness. It did not require the same initial conditions as the expert to follow the trajectory and was capable of driving to the destination from any starting point within the environment. However, the model failed when the vessel's orientation was reversed by 180 degrees. This was observed when the vessel started from a point between the origin and the destination, but was facing the origin instead of the destination. In this case, the model drove the vessel toward the origin, indicating that the policy is sensitive to initial orientation.

The model showed good generalization with radar data, and the learned policy transferred well to various previously unseen trajectories the vessel could still reach the destination. Although the velocity profiles in these unknown trajectories did not match the expert demonstrations, the spatial paths remained similar. This indicates that the model effectively solves the navigation problem, potentially outperforming approaches focused solely on energy optimization.

The above explanation addresses the two research questions that were initially raised at the beginning of the report.

5.2 Future Work

- **Incorporating Fuel Consumption into the Reward Structure:** Currently, fuel consumption is not included in the state or reward formulation. If expert trajectories exhibit consistent patterns in fuel usage, fuel consumption data can be integrated into the reward function. This would allow the model to learn not only optimal navigation trajectories but also energy-efficient behavior. Additionally, auxiliary reward terms related to fuel usage could be introduced during training to guide the policy toward fuel-efficient decision-making.

- **Using Actual Island Boundaries:** At present, the environment uses generalized boundary representations. Replacing these with actual geospatial island boundaries would allow the model to explore more realistic and efficient paths, especially in complex navigation scenarios involving narrow passages or irregular coastlines.

- **Increasing Data Sampling Frequency:** The current data sampling rate is 1 Hz, which is insufficient to fully capture the vessel's dynamics. A higher sampling frequency would provide better temporal resolution, allowing the model to learn fine-grained control behavior and react more effectively to disturbances.

- **Transitioning from Kinematic to Dynamic Modeling:** Due to the lack of IMU data, a simplified kinematic model was used in this study. Incorporating vessel-specific physical parameters along with IMU measurements would enable the use of a dynamic model, which can better represent the vessel's acceleration and response to external forces such as wind and currents. This would significantly enhance the model's ability to generalize and handle real world disturbances.

- **Leveraging AIRL Reward with Model Predictive Control:** The AIRL framework outputs both generator and discriminator weights, effectively learning a reward structure that solves the navigation task. This learned reward function can be combined with auxiliary objectives (e.g., fuel consumption) and used within an MPC framework. This could result in better results in terms of fuel optimization.

- **Enhancing Environmental Realism:** To better capture the expert's behavior, the simulation environment can be improved by incorporating realistic environmental disturbances such as waves, currents, and wind. These additions would force the model to learn more robust navigation strategies that align closely with expert behavior.

DECLARATION:
We have used artificial intelligence tools to help us with rephrasing sentences and to improve readability. These tools were used only to improve the clarity and presentation of the content and not to generate original research ideas or analytical results.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, pages 1–8. ACM, 2004.

[2] B. D. Argall, S. Chernova, M. Veloso, and B. Browning. A survey of robot learning from demonstration. *Robotics and Autonomous Systems*, 57(5):469–483, 2009.

[3] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods, and progress. *Journal of Artificial Intelligence Research*, 67:1–38, 2024.

[4] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

[5] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[6] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[7] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[8] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4565–4573, 2016.

[9] Lingyu Li, Yong Ma, and Defeng Wu. Underactuated msv path following control via stable adversarial inverse reinforcement learning. *Ocean Engineering*, 299:117368, 2024.

[10] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[11] Andrew Y Ng, Daishi Harada, and Stuart J Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, pages 278–287, 1999.

[12] John Schulman, Philipp Moritz, Sergey Levine, Michael I Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[14] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition, 2018.

[15] Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 1433–1438, 2008.

A

Appendix 1

The stages of training: As the number of epochs increases, we can see that the generator generates actions very similar to those of the expert, and the states are similar to the expert states.

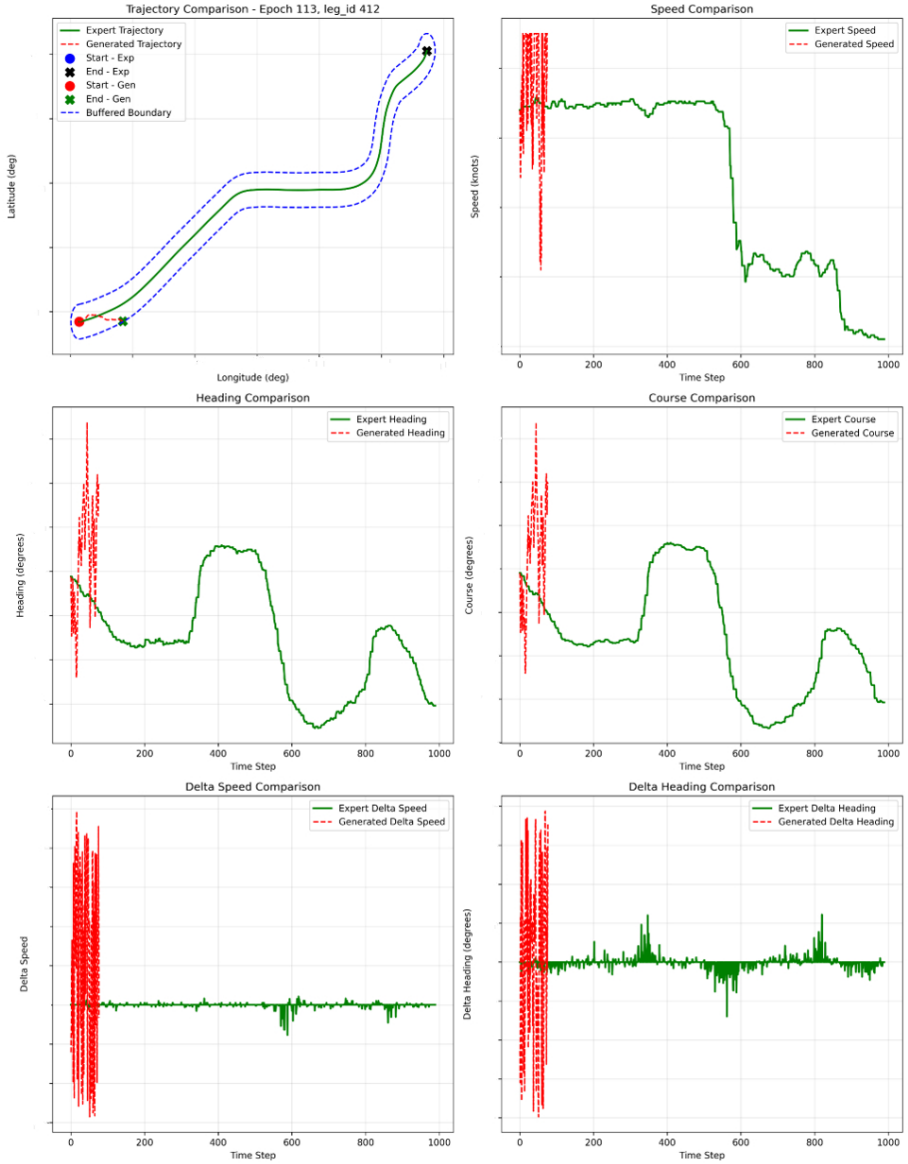


Figure A.1: Training results - 1st stage

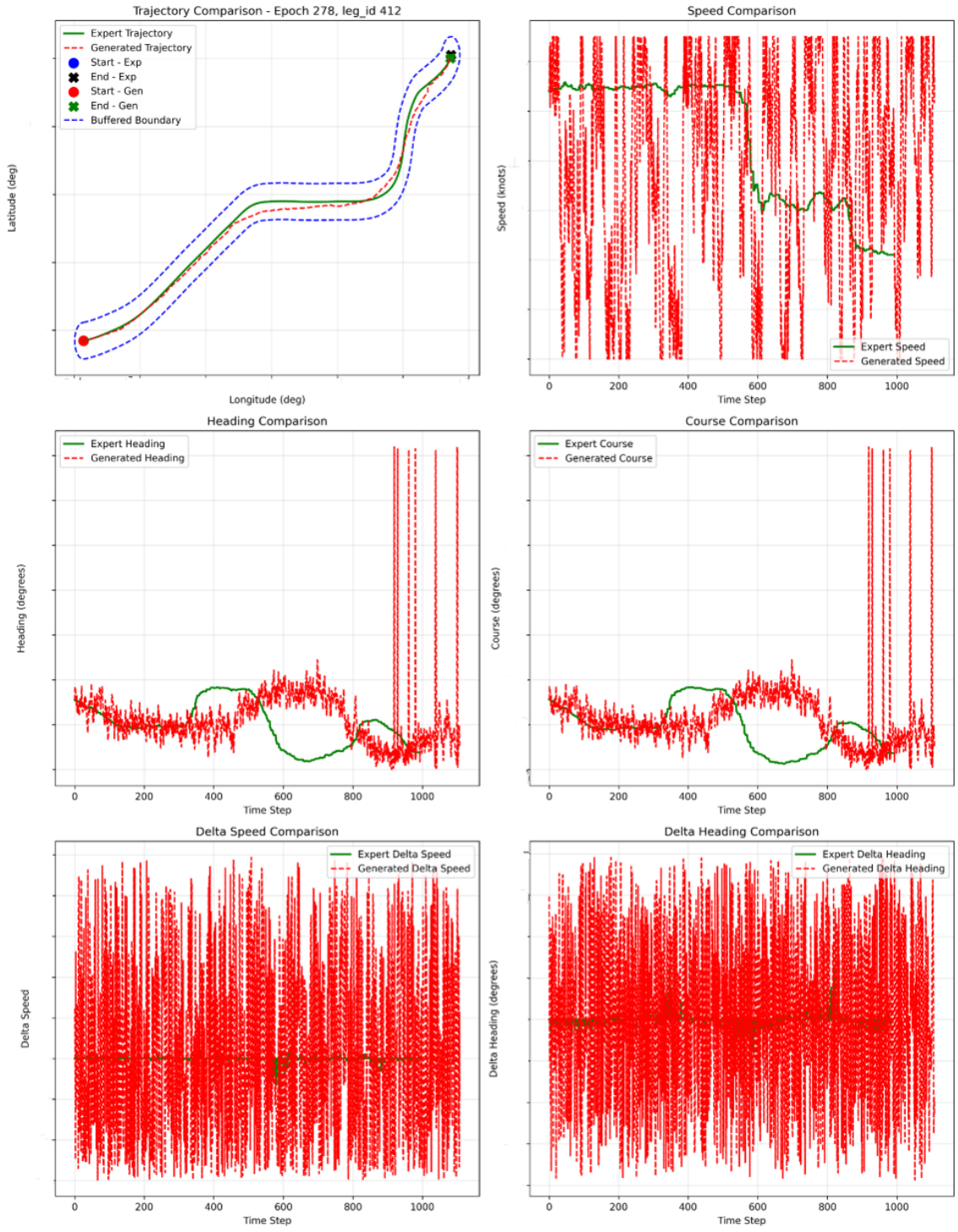


Figure A.2: Training results - 2nd stage

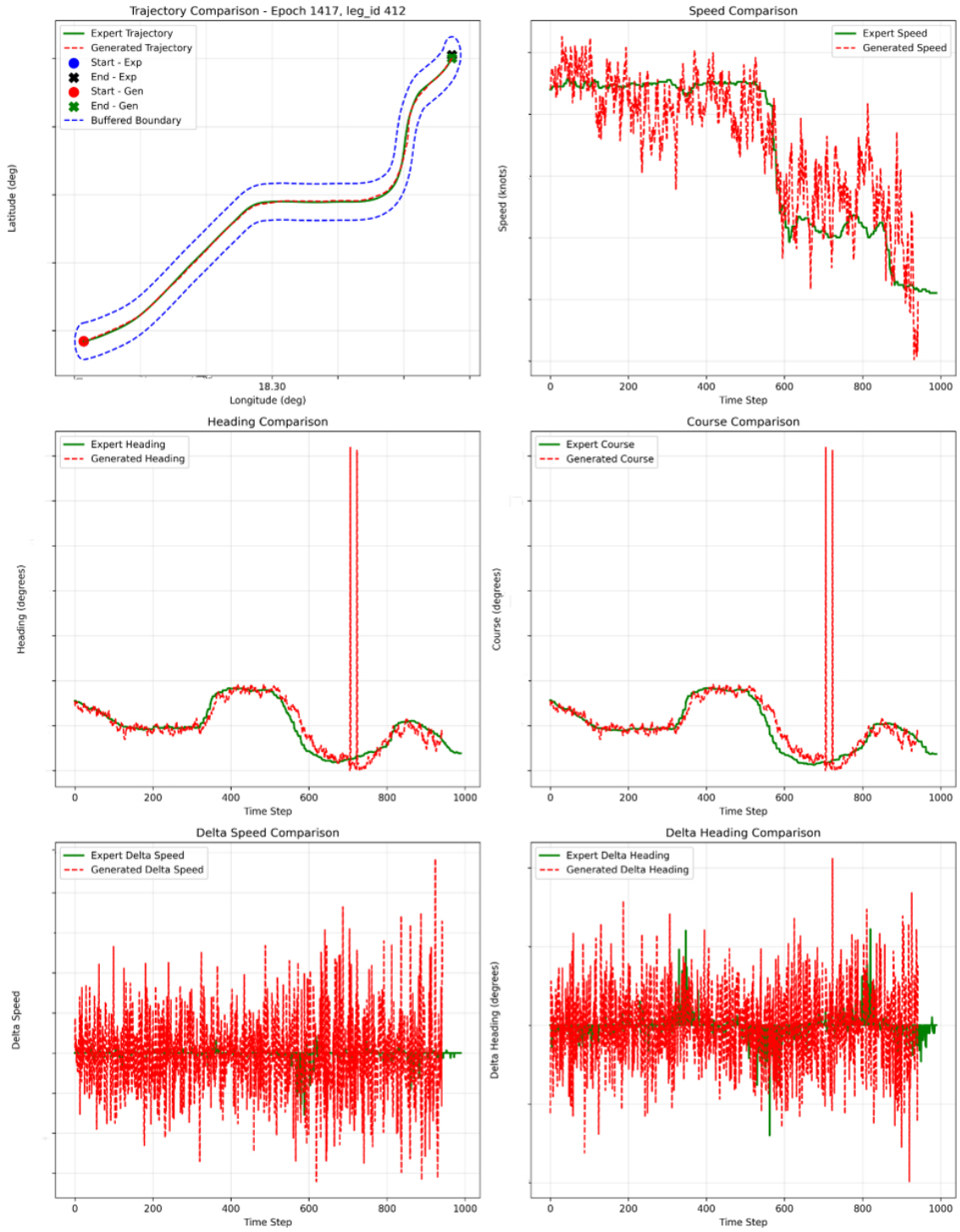


Figure A.3: Training results - 3rd stage

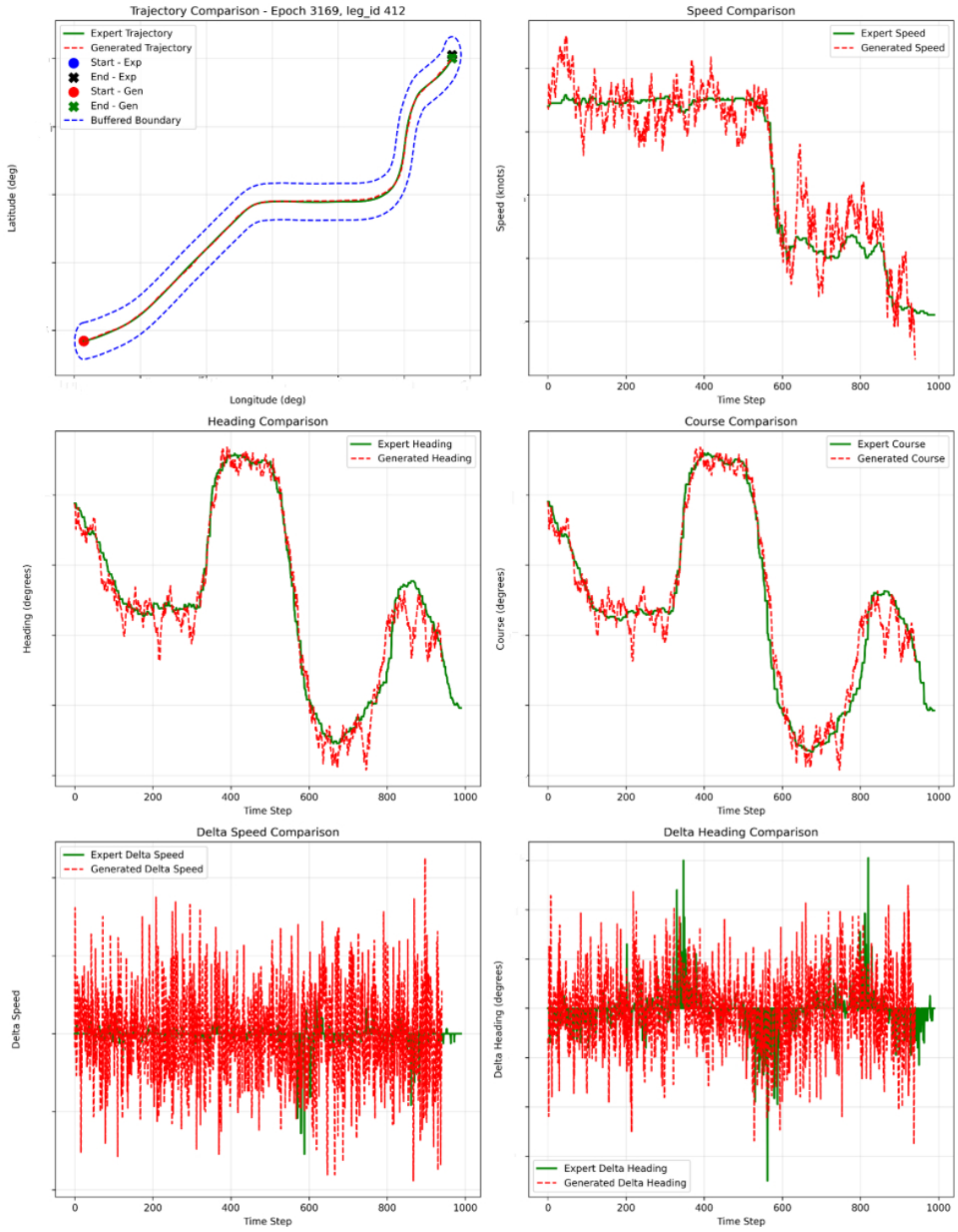


Figure A.4: Training results - 4th stage

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY