Department of Computer Science

This is a report on a project submitted for the degree of MEng in Artificial Intelligence in the Department of Computer Science at the University of York.

Autonomous Navigation System for Vessels in Channels Using Neural Networks Evolved with Genetic Algorithms

Thomas Haslam

28th April 2025

Supervisor: Dr Poonam Yadav

Acknowledgements

I would like to thank my supervisor for all the advice they gave me during this project.

Contents

List of Figures

List of Tables

Executive Summary

This project presents the development of an autonomous navigation system designed for nautical vessels operating in narrow channel environments such as ports, harbours, and passages. Neptune's system utilises a convolutional neural network evolved through genetic algorithms to create a safe, effective, and generalised navigational model.

The motivation for this project stems from real-world issues caused by navigational failures and errors, notably the Suez Canal blockage in 2021, which resulted in an estimated USD $54 billion in trade losses. Incidents like these demonstrate the need for autonomous systems to aid in navigating these complex passages safely, reducing the likelihood of collisions and reliance on human pilots. The systems designed for use in larger vessels to aid in complex navigation can also be utilised in smaller vessels, providing recreational sailors with the added peace of mind of an autonomous system to navigate difficult channels, which otherwise would require constant attention and careful planning.

The primary objectives of this project were to develop a system capable of navigating a channel without any collisions occurring, as the safety of the crew and vessel is of utmost importance, real-time responses to changing environments, and generalising across various environmental features to allow for navigation across all environments all while providing performance similar to a trained human pilot. To achieve these objectives, a physics-based simulated environment was initialised in Gazebo, and a TurtleBot3 Waffle Pi robot, equipped with LiDAR, odometry, and camera sensors, was used to train and test the various models developed, all utilising ROS2 to control and manage these systems.

Once the required environment was set up, the focus shifted to designing the neural network responsible for controlling the vessel's movements. The initial design used a deep deterministic policy gradient (DDPG) model; however, the performance demonstrated did not meet the basic success criteria of navigating any channel. A simple feed-forward neural network provided greater navigational performance, but again, this model could not navigate a channel safely and ultimately, a one-dimensional convolutional neural network was implemented. This model could accurately capture spatial information from the presented LiDAR data, allowing for safe and

effective navigation of channels. This network evolved with genetic algorithms, utilising population-based training, elitism, tournament selection, crossover and mutation to gradually improve the model's performance over generations.

The training process was greatly improved by pre-training the network before running the genetic algorithm. This pre-training stage involved collecting LiDAR captures and the corresponding linear and angular velocities; then, the model was trained on this data, and the resulting performance allowed the genetic algorithms to maximise the performance across the whole generation rather than focusing on channel exploration. This change resulted in a system which could successfully navigate multiple channels in a significantly reduced time, allowing for further testing and evaluations to be made. The performance of each individual in a population was evaluated against various metrics, including time taken, distance travelled, waypoint progression and navigational smoothness, providing a resulting score that could compare all individuals.

Across various channels and different environments, the system displayed performance comparable to manual channel navigation, and could also generalise to various environments, including unseen channels through accurate detection of various channel features, indicating its potential use for real-world applications. The Neptune system met the majority of laid-out objectives and provided a baseline to expand further on this system. This demonstrated performance resulted in a system that could be used to aid sailors when navigating complex channels, ports and harbours, and with further testing in complex environments, the Neptune system could potentially replace a human navigator, allowing for a full autonomous navigation system.

The future work should include various tests using nautical environments with a variety of weather conditions, including strong wind, currents, and waves, as this would further evaluate the system's performance. Alongside these tests, a focus on integrating COLREGs (International Regulations for Preventing Collisions at Sea) into the system would aid in the decision-making process and result in a safer system, which is essential for a real-world implementation. Once these tests have been implemented, the first real-world land-based tests could take place, providing the first real insight into how this system truly performs, as comparisons between current systems can be made. Once the land-based environments have been tested, the progression onto nautical environments would complete the evaluation process for this system as a whole, allowing for the advanced testing required to enhance the safety and efficiency of vessels navigating in narrow channels.

Statement of Ethics

This project, titled *Autonomous Navigation System for Vessels in Channels Using Neural Networks Evolved with Genetic Algorithms*, follows the ethical principles and considerations outlined by the Department of Computer Science at the University of York.

The data utilised in this project was generated from a simulated Gazebo environment and was controlled manually with no participation of any human subjects; therefore, no personal or sensitive data was collected, stored or processed. The LiDAR and sensor data captured for training and evaluation purposes were artificial numeric data only produced for this project. If the system were implemented in the real world, further considerations would have to be taken about how the environmental data, such as LiDAR captures and velocity recordings, is captured, as this would introduce new ethical and legal concerns. This would include ensuring that the collected data does not capture any personal data by accident, securing the consent from the channel when collecting the data, and complying with any relevant local laws. Following on from the data collection process, the data storage process would require strict protocols to prevent any unauthorised access and misuse.

The autonomous navigation system was trained and tested within a physics-based simulation environment, meaning that no physical risk was posed to any individuals, robots or the environment. However, considerations have been made concerning the real-world impacts of this system. Further real-world testing would require stricter safety regulations and further testing to ensure no damage due to collisions. The developed system is designed for practical maritime applications, including improving safety and accessibility, and the ethical implications of dual-use applications such as military or surveillance have been considered. The system's design prioritises safety, but any future developments would have to ensure that maritime laws and ethical artificial intelligence principles are followed.

Whilst the system developed shows promise in simulations, limitations such as performance in varied weather conditions and real-world deployment challenges have been considered, and further ethical reviews would be necessary if real-world testing and evaluation were to take place.

1 Introduction

1.1 Motivation

1.1.1 Disruptions due to navigational issues

Nautical transportation is vital to global trade, contributing to 80% of all trade by volume and two-thirds of trade by value [1]. Approximately 12 billion tons of goods were transported via the sea in 2022. The Suez Canal is one of the most critical maritime transport routes. The canal connects the Red Sea and the Mediterranean, providing a direct route from Asia to Europe. This route avoids the dangerous passage of the Cape of Good Hope and reduces travel by 5,500 nautical miles and multiple days [2]. Approximately 50 ships utilise the channel daily, handling an estimated 12% of all global cargo ship transportation and 30% of container trade [3].

The Suez Canal Blockage occurred in March 2021, the M/V Ever Given was navigating the channel on a day when the winds reached 40 knots (74 km/h, 46 mph), and the resulting force from these strong winds caused the bow of the vessel to drift, ending up stuck upon the southern canal bank. The disruption caused major trade losses of approximately USD $54 billion and a further $31 billion due to the resulting compensation necessary for victims [4]. It is estimated that Egypt alone had losses of $14 million each day the canal was closed.

Affected ships were required to deviate to the alternative route past the Cape of Good Hope, resulting in additional fuel costs and two to three extra weeks of travel. Those vessels which could not deviate to the alternative route were left waiting in the canal, resulting in a congestion of 370 vessels [4]. Once the Ever Given was freed, traffic could resume, and 450 ships passed through the canal successfully. However, a further 400 ships were still waiting in other areas. The backlog of ships was only cleared 11 days after the blockage began and 5 days after the blockage had been successfully removed.

The impact on global trade while the Ever Given was grounded was massive. However, there are still residual effects today, as it demonstrated how vul-

nerable maritime transportation is to a single disruption, forcing industries and companies to create solutions for all potential disruptions regardless of their likelihood of occurrence.

1.1.2 Aiding Recreational Sailors

Navigating narrow passages, harbours, and unfamiliar ports can be challenging for any sailor, regardless of skill, especially for recreational sailors or those sailing alone. These environments require precise movements and constant awareness, and the complexities involved with real-time decision-making, navigational chart comprehension, and appropriate responses result in human errors due to fatigue, lack of knowledge, or stress. For many sailors, integrating an autonomous system would alleviate some of the pressure caused in these scenarios and help with the demands of passage planning.

The combination of enhanced situational awareness, a guided decision-making process and improved safety would provide those sailing recreationally with an added peace of mind, enabling more people to participate in sailing and improve overall enjoyment for those involved. The system could also aid sailors by acting as a warning system to alert them to any incoming collisions or by managing the typical tasks undergone during navigation, contributing to the overall fatigue, greatly improving the vessel's safety.

This system could make recreational sailing much more accessible, allowing a wider demographic of people to get involved who otherwise would be unable to participate. Additionally, the system would enable newer sailors to learn in a safer environment and understand complex navigational skills by observing the system's decisions. Ultimately, integrating an autonomous system would improve the safety and enjoyment of recreational sailing while also increasing overall participation.

1.2 Success Criteria and Objectives

To evaluate the performance of the Neptune system, a set of pre-defined success criteria and outcomes will be used to guide the development and aid in the decision-making process. These will be used to evaluate the project, aiming to keep the development clear and straightforward. These objectives will be tested and evaluated in a simulated environment, allowing maximum experimentation and a streamlined development process.

The most basic success outcome is a system that can navigate any channel without collisions. Ensuring no collisions is the minimum goal for this project as this prioritises safety, which is of utmost importance as any collision within a channel could have a massive local impact, resulting in the closure of that channel and may require a potentially monumental response effort to rectify this collision. A collision may also have a global consequence, devastating social and economic impacts. For these reasons, any system unable to navigate a channel without causing such collisions will be a failure.

Once the system can successfully navigate a channel without causing a collision, the next success criterion is to ensure the navigation is completely efficiently and effectively, as any disruptions due to slow moving or stationary vessels could affect other vessels within the channel, causing congestion and resulting in both global and local economic impacts depending on the channels importance in the international trade routes. To meet this objective, the system must be able to navigate in real-time, respond to the dynamic environment and safely avoid all collisions. This project aims to improve the system's overall performance over multiple generations and provide an effective navigation system.

The next success criterion, improving the overall system performance, would be ensuring the system can successfully navigate various channels with distinct features. The system would be generalised to varied channels and navigational features such as turns, outcrops and bends. Ultimately, effectively navigating any channel it is presented with and any further training using genetic algorithms would result in greater performance and specialisation for the trained channel.

Once a generalised system is developed, the next success criterion for testing performance would be evaluating it under various weather conditions. The conditions would have increasing difficulty further assessing the performance in many different scenarios, the initial scenarios would consist of a gentle wind and current speeds with favourable directions. The conditions would gradually deteriorate, resulting in more complicated scenarios with extreme wind and challenging sea states, such as extreme currents or large waves.

The final success criteria to evaluate the system's performance would be testing the system in a real-world scenario, providing excellent data about the performance. This would allow for accurate evaluation and comparison against other real navigation systems. As any collisions occurring in these scenarios have tangible effects, these tests would be run in a carefully controlled environment to eliminate damage caused by errors. While this does not completely mimic real-world scenarios, it would provide insights into how the system could perform if placed in a nautical environment.

2 Background and Literature Review

The primary focus of this background and literature review is to analyse historical navigational systems, how these systems have developed with new technology, current autonomous navigation systems and their implementations, and research into similar systems applying to this project.

2.1 Background Information

2.1.1 Navigational Systems

Historical Navigation

The earliest nautical navigation solely relied on landmarks, following coastlines and maintaining a constant line of sight with land, which provided sailors with the first forms of nautical transportation. The use of landmarks developed into following current and wind patterns alongside bird migrations to help guide sailors [5]. Polynesian sailors could use the direction and types of waves to collect information about their position at sea. They also tracked weather patterns, allowing for effective open-ocean navigation. By around 400 AD, they had travelled from the Marquesas Islands to Hawaii, a distance of 2,300 miles [5][6].

As navigation progressed, sailors began using celestial objects to guide them; the sun's position as it travelled from east to west was used to guide their route, and sailors could use the shadows cast at noon to determine the directions of north and south [6]. When the stars rose, sailors watched their movements, measuring their height in the sky and determining their progress. As the knowledge of the night sky increased, constellations were used. Sailors could use the position of each constellation to determine the direction they were heading [5]. As time progressed, sailors used meteorological and astronomical means to navigate the seas; those most knowledgeable could distinguish cold north winds from the warm south

winds. Therefore, names were given to the eight principal winds, culminating in the wind rose [7]. Upon discovering a lodestone's magnetism, the magnetic compass confirmed the wind direction when otherwise obscured. The combination of the wind rose card with a magnetic needle allowed a navigator to read the bearing from the card to determine their heading, this developed into the compass, initially having basic headings corresponding to the winds and gradually increasing in number until it arrived at the 32 points in holds today [6].

Pilot books were created to aid in coastal navigation, detailing the headings to be steered, landmarks, currents and port entrances. Together, this information allowed sailors safe passage between ports. These coastal pilot books gradually accumulated and evolved into the Portolan charts [8]. These charts were used by Mediterranean sailors during the Middle Ages, allowing for the successful navigation of the whole Mediterranean Sea [6]. Portuguese navigators attempted to expand these charts and navigate along the west coast of Africa; however, this was not possible as these charts did not relate to the new methods of dead reckoning they had devised. These methods required charts which utilised latitude and longitude. As methods to accurately determine latitude using quadrants and astrolabes became more common [5] [9], the issue of calculating longitude became ever more apparent. The problem of longitude calculation at sea lasted for centuries, as it relied upon an accurate clock. The clock could be used to calculate the difference between absolute time and local time, thus calculating the longitude as one minute corresponds to four degrees, upon the creation of John Harrison's H4 watch [10] accurate timekeeping was now possible, as the watch only lost five seconds during a two-month duration at sea [9], both your latitude and longitude could be calculated allowing sailors to locate their position accurately at sea. The ability to accurately plot your position allowed for the creation of accurate sea charts and safe navigation for mariners.

Modern Navigation

At the end of the 19th century, nautical navigation involved the use of accurate sea charts and pilot books to plan passages across the sea and the use of instruments such as the sextant and chronometer backed up by the use of dead reckoning [11] allowed for an accurate depiction of the ship's position. This created a strong foundation upon which the technological developments of the 20th century would build. The introduction of radio transmitters, radar, and gyroscopic compasses [12] gave navigation a massive leap for position calculation and safety, as the detection of nearby vessels or landmasses was possible with radar [13]. Another technological revolution came in the form of the Global Positioning System (GPS); the

use of satellites allowed for the precise positioning of vessels. This system grew to include access to all major satellite constellations, allowing accurate determination of positions up to one metre, called the Global Navigation Satellite System (GNSS) [14]. The paper charts used in the past were adapted into vector data sets as Electronic Navigational Charts (ENC) [15]. The combination of GNSS and ENCs provides a fully electronic navigation system (Electronic Chart Display and Information Systems, ECDIS), allowing sailors to view their location on an accurate chart. The invention of the Automatic Identification System (AIS) allowed for the automatic transfer of ship data between vessels. Information such as the vessel name, IMO number, position, course, and speed is all transmitted between vessels; this information appears on the ship's ECDIS, helping navigators avoid collisions by highlighting hazards.

As technology progressed, the combination of ECDIS, AIS, ARPA and various sensors [16] enabled navigators to make accurate decisions even in poor conditions. A myriad of tools can assist an expert navigator in all situations, only requiring a small amount of intervention from the sailor. Ships also have the possibility of following planned routes using autopilot systems however, these systems do not replace the need for an expert human navigator but only assist in controlling the ship. These systems have evolved from basic course-holding computers to adaptive systems which can reduce fuel consumption, rudder movement and journey times [17]. Despite all this technology, pilots are still used in ports and narrow passages. A nautical pilot is a sailor with expert knowledge about a specific area or channel [18]. Navigating a port is the most difficult part of a journey due to every channel's varied nature, the constant traffic flow and the requirement to dock safely. This is why vessels rely on a pilot to navigate safely. Canals such as the Suez Canal require a pilot for each ship to reduce the chance of collisions occurring, as without the knowledge each pilot possesses, major disruptions would occur regularly, as the usual navigators would struggle to navigate the channel safely.

Sensors are required to collect the massive amounts of data needed for navigation. Most boats are equipped with radar, magnetic compasses, a speed and distance log device, an echo sounder, an anemometer, and an ECDIS, embedded with AIS and GNSS [16]. All these sensors result in a large amount of data being collected. To successfully navigate a channel, the vessel gathers information on the channel chart, the boat's distances between all sides of the channel, the wind speed and direction, the channel current speed and direction and the locations of any vessels that could collide. The anemometer, log, depth sensor, and ECDIS collect the data to determine the distance to the sides of the channel. An accurate position fix within the channel would allow the ECDIS to calculate the distance using chart data.

Autonomous Navigation

Recently, we have seen advancements in autonomous navigation for boats through machine learning and deep learning models. These techniques provide boats with automated systems and reduce the need for human involvement. The International Maritime Organisation use Maritime Autonomous Surface Ships (MASS) to describe vessels that operate with little to no human interaction and perform all necessary tasks. There are four levels of autonomy, from basic automated processes with a crew on board ready to take over at all times to a fully autonomous vessel capable of making its own decisions [19]. The introduction of MASS would reduce operational costs, eliminating the need for crew wages and accommodation, improve safety as errors made due to human error would no longer occur, and improve overall efficiency by reducing fuel consumption and optimising routes. The four main categories of electronic systems required for autonomous navigation are sensing, communication, decision-making, and actuation.

Sensing requires the sensors on board, such as radar, lidar, GNSS and cameras, to view the environment to provide information about the boat's speed, position, direction, and other vessels and objects nearby [20]. Communication systems are used to transmit data between ships and shore stations. This is essential for navigation as it provides data which can be used to avoid collisions. The decision-making systems process the data from the sensors using computer vision, and together with data communicated from other boats, artificial intelligence responds to the environment to navigate through it. The actuation systems execute the commands by physically controlling the navigational instruments to adjust speed, course, and ballast [21].

The NeuBoat technology from Avikus currently provides navigational assistance to captains aiming to reduce collisions during navigation and docking through cameras and sensors gathering information and relaying this through augmented reality [22]. Avikus has an additional product, an AI-controlled route-planning system, navigation and collision avoidance requiring human intervention. An advanced system adds an AI-controlled docking system providing safe entry to a dock, one of the most difficult parts of a passage. Uncrewed Surface Vessels (USV) from Maritime Robotics have a variety of uses, including data collection, surveying, remote-controlled operations and endurance passages. These USVs have multiple applications, but the underlying structure remains the same, demonstrating that once a framework has been developed and implemented. The core systems of the USVs include a vehicle control system or remote control centre where the vehicle is controlled and monitored remotely, providing detailed information for safe navigation; the onboard controller which manages all operations ensuring they occur accurately and precisely; and information

collection this is provided by a 360-degree view of the vehicle surroundings and analysis of this in real-time allows for assessment of hazards and other objects [23].

2.1.2 Neural Networks Evolved with Genetic Algorithms

The current systems use a variety of artificial neural network (ANN) architectures including convolutional neural networks (CNN) for image classification and computer vision, recurrent neural networks (RNN) as time-series data processing is necessary and specifically long short-term memory (LSTM) networks are used as predicting future states from previous observations is essential for continuous navigation, and deep reinforcement algorithms are used as through continuous interaction the quality and performance of the algorithm greatly improves [24] [25]. The machine learning algorithm being used includes regression algorithms for localisation and actuation specifically Bayesian regression and decision forest regression models, pattern recognition (DRL) algorithms are used alongside CNN for object detection and filtering sensor output, and decision matrix algorithms are used when making decisions as it analyses the relationships between values and information to select the correct response [26] [27] [28].

The challenges involved with autonomous navigation include ensuring that all algorithms can operate within real-time constraints and that the algorithms are generalised to provide good performance across a variety of different environments, as safety is ultimately the most important issue, ensuring that the algorithms are robust enough to prevent failures in both time and safety-critical situations. Genetic algorithms could provide a cost-effective solution for autonomous navigation, as they would reduce the number of expensive sensors required. To overcome these challenges. An approach utilising evolutionary algorithms (EA) could solve these issues. Combining the numerous algorithms currently used with genetic algorithms, optimising parameters and building on previous iterations, reducing error and becoming more robust and effective. When utilising evolutionary algorithms each neural network is an individual of the total population for that generation containing different parameters, after each population of solutions, the performance is quantified using a fitness algorithm to measure the success of each solution then those deemed to be the most successful create new solutions with a combination of parameters [29].

An environment is required to train these algorithms, and simulating it virtually allows for easy modification. The simulation would provide real-world scenarios, allowing the algorithm to develop in a real-world context. By altering the environment we can simulate multiple specific scenarios with a variety of factors improving the robustness of the algorithm and the

quality of navigation, as safety is the main priority, ensuring the algorithm correctly follows the International Regulations for Preventing Collisions at Sea (COLREGs) in all scenarios would provide confidence in the algorithm.

2.2 Literature Review

This paper, titled *An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning*, tries to solve the challenge of autonomous path planning in marine environments, reflecting on the difficulty due to the dynamic and complex nature of the sea. The current method involves prior knowledge and lots of computational resources, which is unsuitable for real-time decision-making. The authors propose a deep reinforcement learning (DRL) model using the deterministic policy gradient (DDPG) algorithm. Alongside the DDPG, the authors used an artificial potential field (APF) to improve decision-making accuracy. They integrated navigational rules (COLREGS), which ensured safety, training this in a simulated environment using real-world AIS data for validation and testing, ultimately producing a system that effectively avoids obstacles and other vessels in numerous scenarios. Using real-world AIS data would provide an effective blueprint for training and testing the neural networks [25].

This paper, titled *Online modeling and prediction of maritime autonomous surface ship maneuvering motion under ocean waves*, addresses the challenge of predicting the movement of Maritime Autonomous Surface Ships (MASS) in real time. Currently, models struggle to adapt to the dynamic conditions of the sea in real time, resulting in poor performance. The proposed solution relied upon a modelling method based on the Least Squares Support Vector Machine (LS-SVM); this helped enhance adaptability and accuracy by using a black box model that could constantly relate all the input variables collected. They also utilised a sliding window technique to select the most relevant data and combined it with error-based updates; this handled long-term predictions in various environments. Although this study looks at environments with ocean waves and not narrow channels, the algorithms used provide evidence for using adaptive components within my project [24].

This paper, titled *Nonlinear Model Predictive Control for Enhanced Navigation of Autonomous Surface Vessels*, specifically focuses on collision avoidance, anti-grounding, and environmental adaptability. These authors utilised a non-linear model predictive control (NMPC) combined with Artificial Potential Fields (APFs) and a nonlinear disturbance observer. Electronic Navigational Charts (ENCs) were incorporated to define grounding

hazards, combined with APFs guiding the vessel to ensure safe navigation. This paper used a distance observer, which improved NMPC performance to compensate for environmental factors. This project directly compares with mine as it involves all the same aspects and provides insight into useful, robust frameworks and validation methodologies [30].

This paper, titled *Nonlinear Model Predictive Control with Obstacle Avoidance Constraints for Autonomous Navigation in a Canal Environment*, addresses the problem of autonomous navigation for small boats in canal environments, which directly relates to this project. This paper again uses non-linear model predictive control (NMPC) combined with LiDAR-based obstacle detection and then validates the experimental data in the Pohang Canal. Using a single NMPC layer reduced the computational resources and improved real-world adaptability. This paper directly relates to my project as both consider navigation in constrained environments. The reliance on LiDAR has limitations in harsher weather conditions, which are more prevalent at sea [31].

The paper, titled *Towards A COLREGs Compliant Autonomous Surface Vessel in a Constrained Channel*, addresses the challenge of autonomous navigation for surface vessels while complying with the International Regulations for Preventing Collisions at Sea (COLREGs) in constrained waterways. The proposed solution uses a system of sensors and a Kalman Filter for object tracking combined with a Visibility Graph-Inspired Path Planning (VGIPP) algorithm to avoid collisions by dynamically adjusting waypoints. The algorithm was validated on the Charles River, demonstrating effectiveness in real-world scenarios. This paper is highly relevant to my project as object avoidance and navigation in constrained environments directly parallel this project. However, the algorithm is focused on predetermined paths, which limits its adaptability [32].

This paper, titled *Neural Network-based Genetic Algorithm for Autonomous Boat Pathfinding*, focuses on a pathfinding method for autonomous boats navigating through different environments. The solution uses a hybrid approach using neural networks optimised by genetic algorithms. The neural networks are trained on sensors detecting distances and altering the boat's acceleration and heading to control the boat's movement. Genetic algorithms are then used to improve the performance of the algorithm, solutions are tested in virtual environments to allow for varied test conditions. This paper relates to mine, as both use neural networks evolved with genetic algorithms, although the limited scenarios used in the reliability in dynamic environments are untested [33].

This paper, titled *NeuroTrajectory: A Neuroevolutionary Approach to Local State Trajectory Learning for Autonomous Vehicles*, focuses on the trajectory planning of autonomous vehicles by focusing on the limitations of

current approaches. The solution proposed uses genetic algorithms to train deep neural networks, these networks combine CNNs and an LSTM for predictions. This paper uses a Pareto front to balance and optimise the three main objectives. The neuroevolutionary training evolves the network, optimising multiple objectives simultaneously. Using LSTM branches stabilises predictions, and the integration of real-world training improved robustness and reliability. Although this project parallels my own, its focus is on land vehicles and not nautical vessels, which could result in the methodologies used not being as effective [34].

This paper, titled *Neuroevolutionary Multi-objective approaches to Trajectory Prediction in Autonomous Vehicles*, tackles the challenge of optimising trajectory prediction in autonomous vehicles using deep neural networks (DNNs). The authors propose combining neuroevolution with Evolutionary Multi-Objective Optimisation (EMO). This paper again combines CNNs and LSTM for predictions; however, a non-dominated Sorting Genetic Algorithm-II (NSGA-II) optimises the hyperparameters, training and validating using GridSim. The NSGA-II effectively identifies the optimal solutions to balance all objectives. The paper is similar as it uses genetic algorithms and a simulated environment; however, as the focus is on land vehicles, the results demonstrated may not cross over to nautical vessels [35].

This paper, titled *TinyLidarNet: 2D Lidar-based End-to-End Deep Learning Model for F1TENTH Autonomous Racing*, utilises end-to-end deep learning to convert raw LiDAR data into movement commands. This solution employs a 1D convolution neural network to process data in real time for autonomous navigation around a track; they found that a CNN performs better at high speeds and generalises better than other MLP-based models [36].

The papers that are the most relevant directly integrate neural networks with genetic algorithms and focus on optimisation techniques applicable to vessel navigation[33] [34] [35]. Using Pareto front approaches balances out all objectives to ensure efficiency across all areas. Those papers focusing on collision avoidance and path planning by utilising COLREGs ensure that safety is at the forefront of the project [25] [30] [32]. COLREGs enhance the real-world applicability of these systems and the methodologies utilised. Meanwhile, papers using NMPC provide an approach for real-time obstacle avoidance and anti-grounding, complementing the project's aim on adaptability [30] [31]. Those models trained in virtual environments but utilise real-world data gathered by AIS and ENCs provide an excellent blueprint for training, validation and testing this project [25] [30]. Models using CNNs and LSTM networks are highly relevant to my implementation as they provide similar architecture methods for this project [34] [35] [36].

In my research, I discovered a gap in the current state-of-the-art sys-

tems. While extensive research on autonomous navigation using various deep learning techniques has been conducted, few studies utilise neural networks evolved by genetic algorithms, specifically focusing on nautical navigation for constrained channels. Although there is research using a neuroevolutionary approach, these papers focus on land-based vehicles and are not tested in realistic nautical environments to evaluate the performance of a vessel in a narrow channel. There is a paper investigating neural networks evolved by genetic algorithms; however, this paper evaluates performance in simple environments, lacking dynamic environments and complex sensors.

My project directly addresses this gap in research as it explores various areas. This project focuses on channel navigation for nautical vessels, adapting similar neuroevolutionary methods into real-time nautical navigation, where the dynamic environment and challenging manoeuvrability greatly impact the navigation. While the use of CNNs evolved by genetic algorithms has shown great success in other papers, the performance of these networks is largely untested in nautical environments, and this project would help further research in this area and provide data to evaluate the architecture. Genetic algorithms provide a powerful optimisation technique and supply advantages for real-time performance by enabling the model to learn through practice and exploration, a common problem encountered when utilising deep reinforcement or supervised learning. This project explores a navigation system without needing massive amounts of pre-defined training data while remaining computationally inexpensive, providing research into small-scale systems while still providing excellent navigational performance.

Paper Title	Methodology Similarities	Methodology Differences
An Autonomous Path Planning Model for Unmanned Ships Based on Deep Reinforcement Learning [25]	Trains in a simulated environment	Uses DRL with a DDPG algorithm and utilises real-world AIS data, an APF is used to enhance decision-making
Online modeling and prediction of maritime autonomous surface ship maneuvering motion under ocean waves [24]	Real-time data analysis and environment adaptability	Uses an LS-SVM algorithm in open ocean environments rather than an LSTM CNN
Nonlinear Model Predictive Control for Enhanced Navigation of Autonomous Surface Vessels [30]	Focuses on collision avoidance and various environmental factors	Trained using an NMPC with APFs on data gathered from ENCs instead of an LSTM CNN
Nonlinear Model Predictive Control with Obstacle Avoidance Constraints for Autonomous Navigation in a Canal Environment [31]	Focuses on boats in channel environments whilst using LiDAR data	Trained using an NMPC algorithm then validated in a real-world environment rather than an LSTM CNN
Towards A COLREGs Compliant Autonomous Surface Vessel in a Constrained Channel [32]	Focuses on the navigational challenges of constrained channels	Uses VGIPP algorithms instead of an LSTM CNN to dynamically adjust waypoints
Neural Network-based Genetic Algorithm for Autonomous Boat Pathfinding [33]	Uses neural networks evolved by genetic algorithms	Simulated in a 2D basic environment
NeuroTrajectory: A Neuroevolutionary Approach to Local State Trajectory Learning for Autonomous Vehicles [34]	Trains CNN with an LSTM network, then evolves them using genetic algorithms	Focuses on land vehicles instead of nautical vessels
Neuroevolutionary Multi-objective approaches to Trajectory Prediction in Autonomous Vehicles [35]	Utilises CNN and LSTM algorithms with genetic algorithms to evolve neural networks trained in simulated environments	Focuses on land vehicles
TinyLidarNeyt: A 2D Lidar-based end-to-end deep learning model [36]	Utilises CNN to analyse LiDAR data in real time and pretrains the network on collected data	Doesn't use genetic algorithms to improve performance and is trained in real-world environments instead of simulated ones

Table 2.1: A table comparing methodologies against my final implementation

3 Methodology and Implementation

Navigating narrow channels such as ports, harbours, and passages provides higher navigational difficulty and complexity than open water environments for human and automated pilots. The additional challenges created due to a reduction in manoeuvrable space, an increased number of obstacles, and varying current strengths and directions ensure that this is the most complex part of the passage for any vessel, especially for RAM (restricted in their ability to manoeuvre) vessels such as container ships. These challenges meant it was essential to implement a methodology capable of adapting and responding to various environments in real time.

This led to an implementation that collected LiDAR data from the environment for analysis and produced linear and angular velocities as outputs, controlling the robot's movements. A neural network was utilised as it can learn from environmental data with no explicit programming, allowing it to easily adapt to various scenarios, providing a generalised system, which is essential for navigation, as no two environments are the same. A genetic algorithm was selected to evolve and improve the neural network's performance as it allowed for generalised improvements in decision-making that would improve performance across multiple environments, whilst also optimising the network without requiring a predefined set of rules and parameters. Genetic algorithms utilise reinforcement learning to improve performance based on a custom fitness function, allowing for dynamic improvements.

The Neptune autonomous system aimed to navigate from the start to the end of a channel whilst avoiding all obstacles and ensuring no collisions occurred. The system was trained and evaluated in a Gazebo simulation environment, providing simulations with real-world physics, allowing for accurate simulations [37]. The robot is controlled by various ROS (Robot Operating System) topics, which collect the necessary data from the environment and provide the required output data to control the robot's movements [38].

3.1 Neptune's Simulation Environment and Robot Controller

Different 3D environments were designed and created in Gazebo to simulate and train the system. The environments began with basic channels with few turns and gradually became more complex, adding obstacles, tight turns and varying weather conditions, including wind and current direction and strength. As Gazebo mimics real-world physics, this allowed for a progressive training model, providing a gradual learning curve in a life-like environment. The simulation was set up using a launch file, which defines the world environment, the robot model, and the required files. Throughout the channel, artificial waypoints are placed to help track the robot's progress through the channel A.1 A.2 A.3.

The robot chosen to navigate these simulations was a TurtleBot3 Waffle Pi robot, as it can simulate both linear and angular motion, reflecting a vessel's movements through water. Equipped with a camera, LiDAR, and Odometry sensors, these sensors form the input data for the neural network utilised. The robot used a controller to interact with the simulated environment via ROS2 functions, which provided the necessary callbacks, simulation adjustments, and movement commands to perform the required navigation. The controller also uses the sensors to collect data used for performance evaluation A.15.

3.2 Neptune's Neural Network

3.2.1 Deep Deterministic Policy Gradient Network

The initial neural network implemented a deep deterministic policy gradient (DDPG) agent, a deep reinforcement learning (DRL). The neural network input was LiDAR sensor data, which provided linear and angular velocity outputs. The DDPG agent contained three sections: an Actor, a Critic A.6, and a replay buffer A.8.

The actor agent utilised the 360-degree LiDAR data as an input and provided 2 outputs, which the robot used as linear and angular velocities to control its movement. The network contained 3 linear layers, followed by a normalisation layer and a ReLU activation layer. The normalisation layer stabilises training and reduces covariance shifts, and the ReLU activation layer adds non-linearity. A Tanh activation function ensures the values are between -1 and 1, mapping to the boat's maximum velocities

and steering.

The critic network evaluated the quality of the linear and angular velocities determined by the given input data, also known as the state-action pairs. The critic produced a single output which scored the actor's performance. The critic network contained 4 layers, each followed by a normalisation and ReLU layer, which provided stability and non-linearity. The final output provides a score for both linear and angular motion separately.

A replay buffer stores data about previous decisions made by the actor and the critic. Enabling the agents to learn from past decisions, improving sample efficiency and reducing training instability from correlated data. The DDPG agent updates the actor and critic networks using a gradient descent method, handles the replay buffer to stabilise learning, and applies soft updates and stabilise learning by preventing rapid changes in targets, further improving performance A.5.

Despite numerous simulations and constant hyperparameter changes to the network and genetic algorithm, the model's performance was poor. This led to the decision to start again and revert to a more basic network, which instantly resulted in greater performance and suggested the initial network implementation was overcomplicated and sophisticated.

3.2.2 Feed-Forward Neural Network

The next implementation of this neural network was a fundamental design to maximise computational speed and reduce complexity A.9. The network took LiDAR data as input and produced two outputs corresponding to the linear and angular velocities of the robot. The network consists of three linear layers: the first is followed by a ReLU activation, the second by layer normalisation and another ReLU to aid stability and convergence, and the final layer outputs two values, which are passed through a Tanh function to constrain them between -1 and 1. 3 layers were selected as a balance between speed, efficiency and accuracy. When 4 layers were utilised, the robot moved too slowly and took too long to analyse the LiDAR data, and when 1 and 2 layers were utilised, the robot's performance significantly decreased A.10.

3.2.3 Convolutional Neural Network

Whilst the performance of this neural network architecture was favourable, it could not generalise to different environments, resulting in collisions at

similar locations throughout the channel. This led to another change in neural network implementation; the architecture chosen for this implementation was a one-dimensional convolutional neural network, as the network was now analysing captures of LiDAR data. The network consisted of five convolutional 1D layers; the first layer began with one input channel and 16 output channels; the second layer had 16 input channels and 32 output channels; the third layer had 32 input and 64 output channels; the fourth layer consisted of 64 input and 128 output channels; and the final layer had 128 input and 128 output channels. Each convolutional layer was followed by a batch normalisation layer to stabilise training and a leaky ReLU activation function to allow for negative values and provide non-linearity to the model. Leaky ReLU was chosen over regular ReLU as negative values are required for linear and angular movement 3.1.

A set of three fully connected linear layers followed the convolutional layers; the first layer had 32 output features, the second layer had 32 input and 16 output features, and the final layer had 16 input features and two output features corresponding to the linear and angular velocities of the robot. The first two linear layers were followed by a dropout layer to help with regularisation and a leaky ReLU activation function; the final layer was followed by a Tanh function, ensuring that values were bound between -1 and 1. The value chosen for the dropout layer was 0.2, as this provided a good balance between performance and generalisation, as a smaller value of 0.1 was found to not generalise well enough, but a larger value of 0.4 reduced performance significantly A.11.

```
Class NeuralNetwork:
    Method: Initialise:
        - Define CNN layers
        - Define Fully Connected layers

    Method: Forward:
        - Process input through network layers
        - Return Output

    Method: To:
       - Move neural network to the specified device (CPU or GPU)
```

Figure 3.1: Convolutional Neural Network Pseudocode

3.3 Neptune's Genetic Algorithm

The genetic algorithm is used to improve the neural network's performance over multiple generations by selecting the best individuals in each population and using them as parents to create the next generation 3.2.

The initial population is created with random weights. Once the whole population has been simulated, the top 10% of the population is selected to become the elites of their generation. These elites become individuals in the next generation, allowing for further progression. This preserved the highest-performing individuals whilst allowing for diversity within the next generation. For each child, tournament selection is used to select 2 individuals from the top 25% of the population, and these two individuals become the parents for that child A.12.

Once the parents had been selected, a crossover function was utilised to distribute the parents' attributes to the child using a randomised weighted sum between 0.3 and 0.7, This provided a random blend between both parents, which supplies greater diversity within the new generation. Each child then underwent mutation to randomly modify weights, preventing premature convergence and improving diversity with each generation. The mutation strength begins at 0.049 and slowly decreases with every generation to ensure the later generations can still explore the environment, but converges to a maximum performance A.13. Each child is then further modified to become more like the top-performing individual of the population. Once all the children have been created and altered accordingly, the simulations are run on this new population.

```
Class GeneticAlgorithm:
    Method: Initialise
        - Set parameters for population size, mutation rate, crossover rate, and other configurations
        - Set state and action dimensions
        - Set device for computations

    Method: Create Population
        - Input: population size, state dimension, action dimension, device
        - Output: population (list of neural networks, each representing an individual)
        - For each individual in the population:
            - Initialise a neural network (DDPGAgent)
            - Add the neural network to the population

    Method: Select Parents
        - Input: population, fitness scores, number of parents, tournament size
        - Output: parents (list of selected individuals)
        - For each parent:
            - Randomly select a few individuals from the population using tournament selection
            - Select the candidate with the highest fitness score as the parent
        - Return the selected parents

    Method: Crossover
        - Input: parents (list of selected parents)
        - Output: child (new individual created from parents)
        - Perform a weighted average of the parameters (actor and critic networks) from two parents
        - Return the child

    Method: Mutate
        - Input: child (new individual)
        - Output: mutated child (child after mutation)
        - For each parameter in the child's actor and critic networks:
            - With a certain probability (mutation rate):
                - Apply small random noise to the parameter (mutation strength decreases with generation)
        - Return the mutated child

    Method: Create Next Generation
        - Input: population, fitness scores, current generation
        - Output: next generation (new set of individuals)
        - Sort the population based on fitness scores and select elites
        - Keep top-performing individuals (elites) unchanged
        - For the rest of the population:
            - Select parents using the selection method
            - Create a child using crossover
            - Mutate the child
            - Add the child to the next generation
        - Return the next generation (including elites and new individuals)
```

Figure 3.2: Genetic Algorithm Pseudocode

3.4 Training the Neptune system

The initial implementation method chosen started simulations with an untrained model and provided complete freedom for the neural network and genetic algorithm to control the training and improve the network performance based on the fitness score. However, pre-training the network on LiDAR data before running the simulations greatly improved overall performance while reducing computation cost.

The final implementation of this neural network had two separate stages to the training process. The first stage 3.3 involved manually navigating the channel using a teleoperation node and saving the captured LiDAR data from the robot and the corresponding linear and angular velocities outputted by the robot for each capture. This allowed the network to be trained on each capture and to predict velocities. The dataset used to train the model consisted of 9330 LiDAR captures of different environments containing long straights, tight turns, S bends, and various other features common in channels. The training utilised AdamW as the optimiser, providing a more generalised model compared to Adam, with a learning rate of 0.001 for 500 epochs. Huber loss was utilised, as it delivered a balance between being robust against outliers and sensitive to small errors, which is important for predicting angular velocities.

The second stage needed to train and develop the algorithm using a loop to simulate all individuals of a population for every generation. The pre-trained neural network is loaded, then each simulation is reset to the initial position, the neural network is then initialised for the individual with some mutation, ensuring that there are differences between the population, and the simulation begins. During each simulation, the robot inputs the LiDAR data into the neural network and moves according to the output received. Simulations can end in failure or success. The simulation fails if a collision occurs, the robot gets stuck at the same waypoint for 30 seconds, or the time limit is reached. The simulation will be successful if the robot reaches the final waypoint. Once a simulation has finished, the collected metrics are used to calculate the individual's performance. The performance of each individual is based on various factors, including distance travelled, waypoints reached, time taken, ratio of linear velocity to angular velocity, and oscillation count.

The fitness scores are calculated in two different scenarios: the first being if the robot reaches the goal, and the second being if it does not reach the goal. If the robot reaches the goal position, a bonus is added for achieving the goal, ensuring that these individuals are preferred when creating the new generation. Minimising the distance travelled and time taken by the robot implies greater navigational efficiency and better performance. To

```
Class LidarDataset:
    Method: Initialise:
        - Define data file

    Method: len:
        Return the length of the dataset

    Method: getitem:
        - Get lidar data and velocities for each row
        - Convert lidar and velocities to correct tensor values

        Return lidar and velocities

Function Training Function:
    - Initialise the dataset
    - Create DataLoader with the dataset

    - Set device
    - Initialise model (NeuralNetwork)

    - Define loss function (Huber Loss)
    - Initialise optimiser (AdamW with learning rate)

    - For each epoch:
        - Initialise loss tracking variables

        - For each batch in DataLoader:
            - Move lidar and velocities to device

            - Zero gradients in the optimiser
            - Pass images through the model to get outputs

            - Compute the loss using outputs and labels
            - Backpropagate loss
            - Update model parameters with optimiser

            - Calculate percentage difference between target and predicted velocities
            - Accumulate running loss and percentage differences

        - Compute the average loss and veloctity percentage differences for the epoch
        - Print loss and percentage differences for linear and angular velocities

    - Save the model's state dictionary
```

Figure 3.3: Neural Network Training Pseudocode

reduce unnecessary movements, an individual is penalised if it oscillates forward and backwards, further rewarding smooth, continuous motion A.16.

If the individual does not reach the goal, the fitness score maximises the distance travelled, time taken, and movement efficiency. Any oscillations are penalised to encourage smooth, continuous motion. If a collision occurs, the individual is heavily penalised to ensure that these individuals are not selected when creating the next generation. The fitness score is then multiplied by the number of waypoints reached, greatly prioritising channel progression and reaching further waypoints. Once the fitness scores have been calculated for the whole population, the next generation is created based on these scores, and the process starts again.

4 Results and Evaluation

4.1 Neural Network Architecture Evaluation

My initial neural network architecture utilised a deep deterministic policy gradient (DDPG) critic and agent method, which analysed and evaluated the performance of my network based on the decisions that were made. The aim was to improve performance as the agent and critic learned; however, the model would frequently converge at a local optimum, becoming unable to advance past a specific point in the channel. Despite changes to my genetic algorithm, this model could not diversify enough amongst a population to progress in the navigation of the channel, as many of the individuals would follow the same path with minimal deviation 4.1.

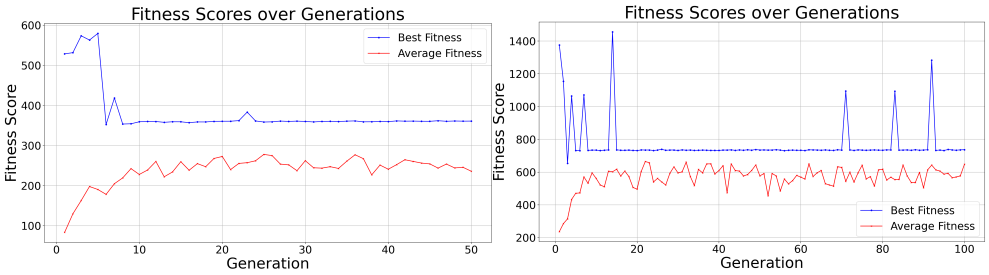

Figure 4.1: A graph showing the lack of deviations in the DDPG neural network

The model's performance would often plateau and become stuck in a local optimum, which could successfully navigate the channel to a specific feature before colliding with the channel 4.2. To combat this, simulations were run with 100 individuals for each population, aiming for enough random mutations to improve diversification. While a single individual would perform much better than the rest of the population, this performance was not passed on to further generations and was often lost when creating the next generation.

The changes made to the neural networks' hyperparameters and the genetic algorithm code improved the performance of this model significantly from an average fitness score of 250 to 900, and the best fitness score

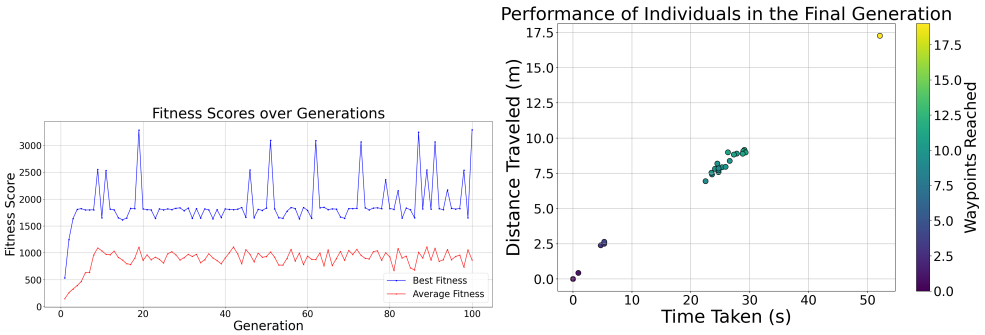

Figure 4.2: A graph showing the plateaus of the DDPG neural network

increased from 580 to 3300 A.17. This meant that the average individual went from getting stuck at the first turn to progressing down the channel and colliding with the channel walls, while the best individuals of the initial simulations could attempt the channel's first significant turn with varying degrees of success. The best individuals of the later simulations could successfully navigate the first important turn, continue to the channel's most difficult turn and collide with the walls. While the performance of this model improved, the results were not satisfactory, as the model could not navigate a simple channel, the most basic objective and the changes to my genetic algorithm could not improve the navigational performance. This led me to change my neural network architecture.

The next neural network architecture that was implemented utilised a simple feed-forward design, aiming to keep the computational complexity low, allowing for real-time navigation, an important objective for this project 4.3. The navigational performance of this model improved greatly compared to the previous model, with the generations gradually navigating the channel and learning the different environments in real time; this demonstrated an effective genetic algorithm which could improve the performance as it learned.

This model could learn the environmental patterns and retain the relevant information throughout training, improving its navigational performance. While the performance of this model was significantly enhanced, it could only navigate the channel when the environment was simple, with few obstacles 4.4. This failed to navigate more complex environments, resulting in solutions stuck in local optima and unable to navigate the full channel. While the performance displayed for the section of the channel it could navigate was impressive, the inability to navigate more complex environments meant this model did not achieve the most basic objective of successfully navigating a channel A.18.

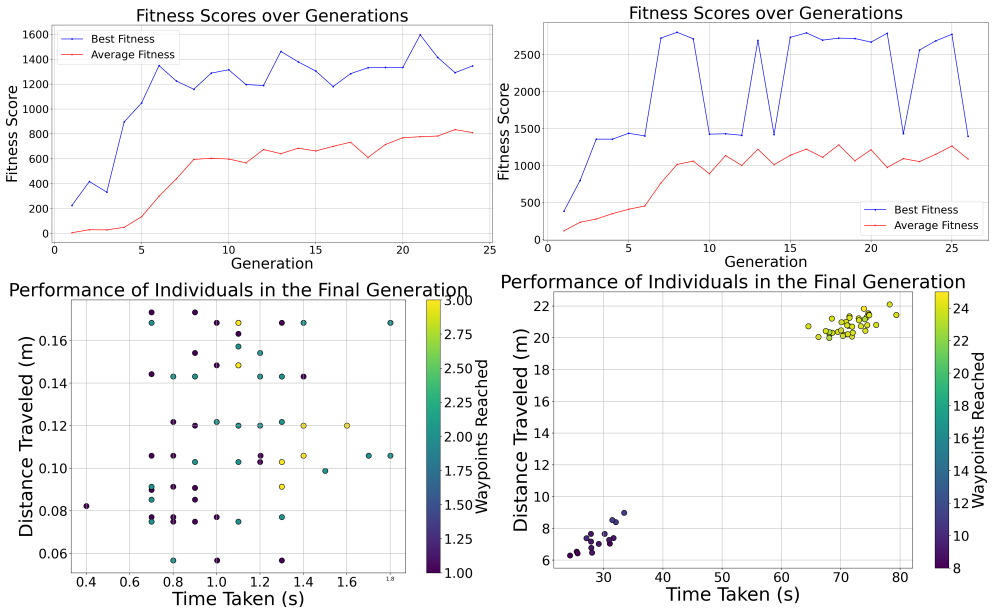

Figure 4.3: Graphs showing the improved performance of a feed-forward neural network

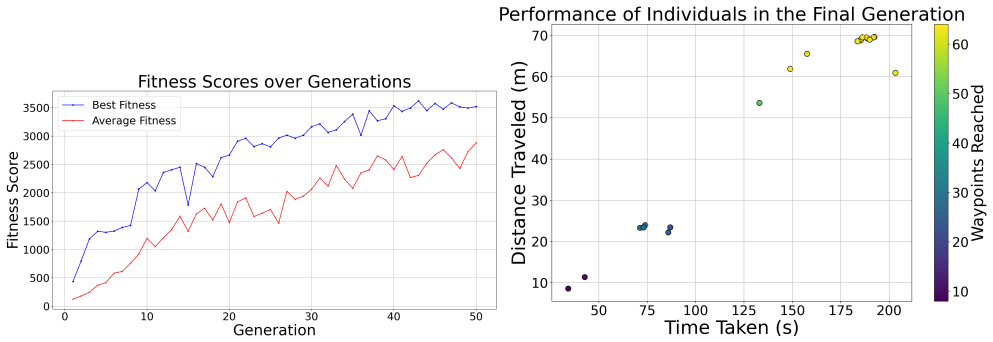

Figure 4.4: A graph showing a feed-forward neural network stuck in local optima

This stall in performance resulted in further research, aiming to find neural network architectures used in similar projects with promising performance. This led to my final implementation: a convolutional neural network architecture. This architecture could capture the spatial information required to navigate the channel, resulting in performance comparable to my previous implementation with minimal training. Despite this early performance, the performance of the average population did not match the best performance of the population, leading to an improvement in my genetic algorithm, aiming to retain the elites of the generation whilst also bringing each individual closer to the elites' performance to increase the average population's

fitness score.

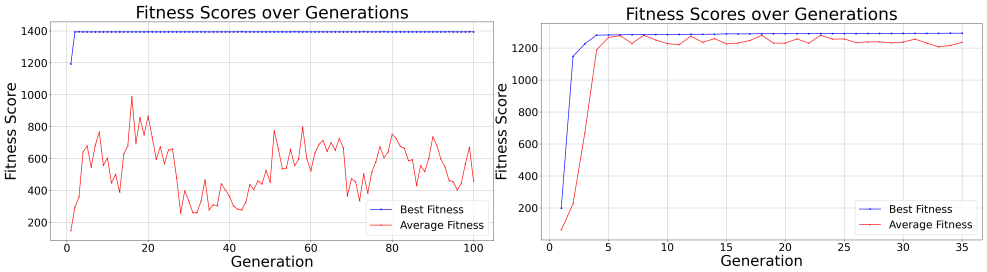

Figure 4.5: A graph showing the improvement in performance for the average population

This would allow for an overall increase in performance as each individual would experience more of the environment and learn more information. While the neural network could learn all the required information about the environment over time, it was very reliant on an individual progressing much further than the rest of the population, and performance would slow down until the performance of the average individual could catch up. While this model completes the fundamental objective for this channel, it takes an extremely long time. To collect this data, simulations were run on 100 generations with a population of 50 individuals. This training would take over four days to train a single model to completion.

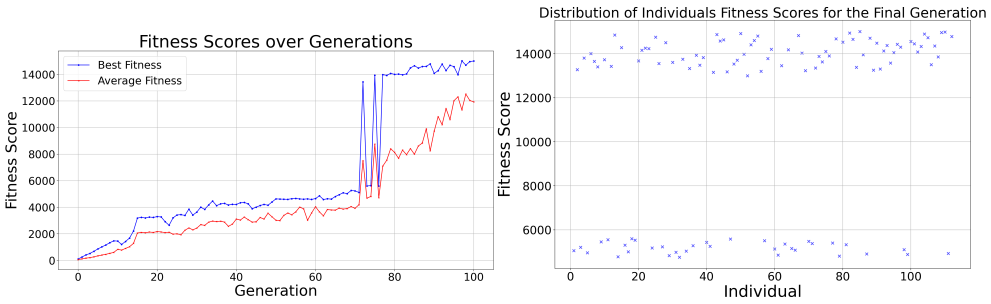

Figure 4.6: A graph showing a full simulation with a CNN architecture and the distribution of the final generation's fitness scores

4.2 Benefits of Pre-training the Neural Network

Despite the CNN architecture's performance being much greater than the previously utilised architectures, the computational resources required to train the network were very high, as it required many generations of large

populations to get an individual to the end of the channel. While this may be possible for large-scale projects with greater computational resources, a solution was required.

The first attempt to find a solution was running simulations faster than real time; this greatly reduced the training time as the simulations could run over 10 times faster than real time, resulting in much quicker training times, provided the algorithms utilised could match this pace. However, due to the complexity of the neural network and robot controller, this speed exceeded the computational limits required to control the robot, collect the required metrics, and manage the simulations. Ultimately, the maximum simulation speed that ensured the neural network and robot controller could still compute without errors was a simulation speed twice as fast as real time. This balances reducing the training time and allowing the robot controller to respond effectively to the environment.

This solution reduced the training times, but the overall computation time was too high. This led to a decision to pre-train the data on LiDAR captures of multiple environments, allowing the network to learn environmental features without navigating the full channel, greatly improving the network's performance and reducing the training time. By simply pre-training the network with data collected from an ideal channel navigation, the initial performance of the model was much greater than a random initialisation. Even just training on a small amount of data for a small number of epochs demonstrated a significant performance improvement.

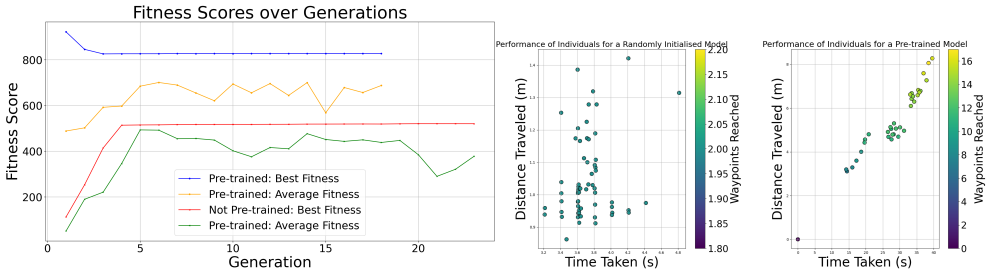

Figure 4.7: Graphs showing the fitness Scores and metrics for the initial generation of a pre-trained model vs a random initialisation model

When training the data, various learning rates, optimisers and loss functions were tested to find the optimal hyperparameters to maximise performance. Huber loss was chosen as it is robust against outliers and sensitive to small errors, preventing overfitting to the dataset. A learning rate of 0.001 was selected to balance convergence speed whilst not overfitting to the training data, as this training only needed to provide a baseline to begin training the model using the genetic algorithm. The optimiser AdamW was chosen

to provide better regularisation and generalisation, ensuring that the model did not overfit to the data.

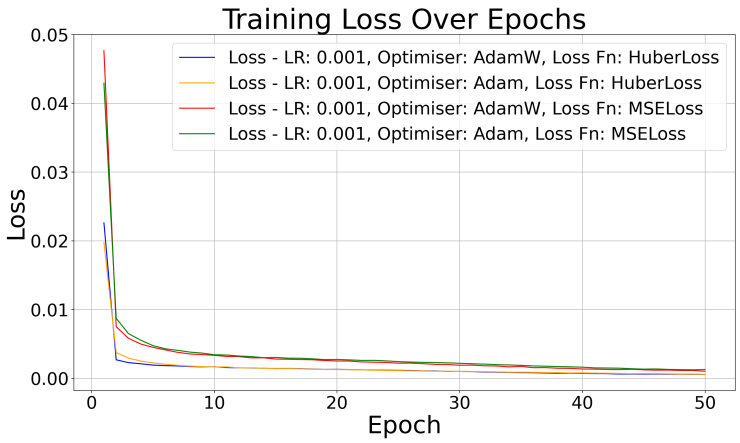

Figure 4.8: A graph showing the loss per epoch

This training resulted in a model which could navigate the channel as a whole, removing the time required to explore the channel, reducing the total training time and allowing the genetic algorithm to focus training on improving the actual navigation performance rather than channel exploration. This resulted in improvements across the overall population for both the elite and the average individuals, as the algorithm could spend more time fine-tuning performance for the whole channel and avoiding overfitting to a smaller section of the channel.

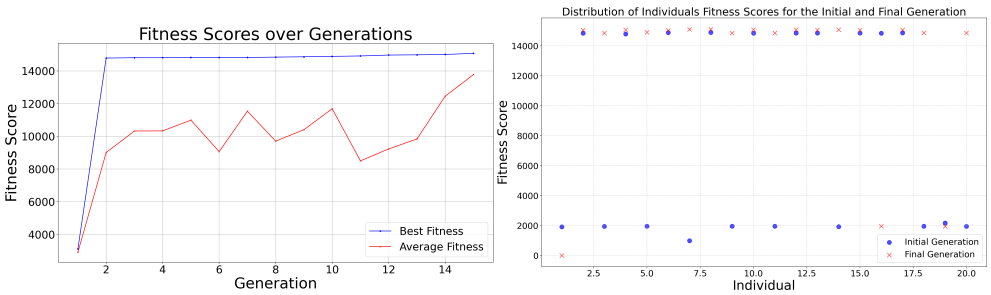

Figure 4.9: A graph showing the performance of a pre-trained model over generations

4.3 Neptune's Final Performance

The model was tested in various environments to evaluate its performance against complex channel features. The environments chosen consisted

of four different channels, each with varying degrees of complexity and diverse features, allowing for a comprehensive evaluation of the model's performance. The most basic channel was wide, with a small 90-degree bend and two 180-degree bends with straight walls connecting each turn A.2. This gave plenty of room for non-optimal movement patterns and allowed the robot to progress and still make mistakes. This demonstrates the efficiency and effectiveness of the systems performance as this resulted in individuals successfully navigating the channel in an average of 144 seconds and travelling a distance of 16.2 meters, which is faster than the average time taken to manually navigate the channel of 152 seconds and shorter than the average distance travelled of 17.1 meters.

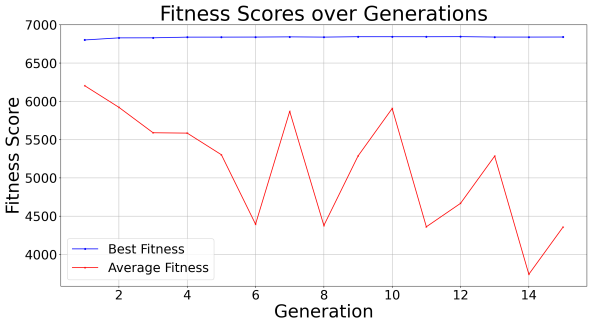

Figure 4.10: A graph displaying the performance of the model over generations on the basic channel

As shown by the graph, the average performance of the model decreased over time, whilst the best performance remained relatively constant. This demonstrates that the genetic algorithm used was not successful at improving the overall performance, suggesting the mutation or crossover functions were decreasing the performance of the next generation, resulting in an overall decline in average performance. Upon discovering this result, altering my genetic algorithm to ensure the individuals of the population were brought closer to the top individual of the population increased the average performance over time.

This was evident when testing my model on a more complex channel; this channel was narrower, had natural walls containing curves and other imperfections, and tighter turns with varying degrees of difficulty A.1. This resulted in not only the average performance of the channel increasing, but the best performance increasing as well, demonstrating the performance of the genetic algorithm over time. The system performance on this channel was relative to the performance when manually navigating the channel, as the system averaged a time of 312 seconds and a distance of 39.0 meters, the manually navigated runs averaged 285 seconds and 38.4 meters. While this performance is marginally worse than the manually navigated performance, this data is encouraging as it provides evidence

that the autonomous system can replace a human pilot.

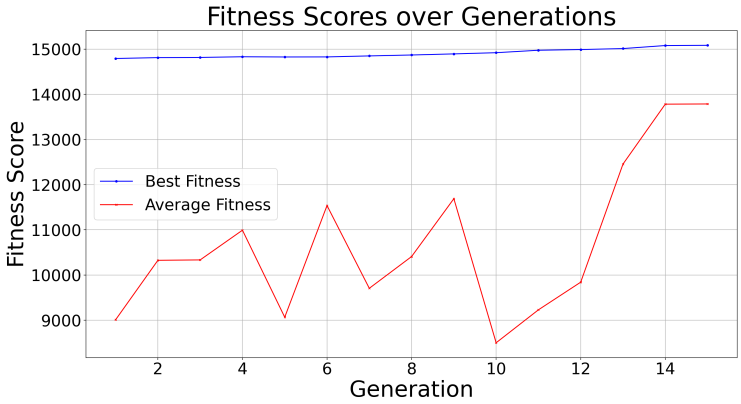

Figure 4.11: A graph displaying the performance of the model over generations on a standard channel

While the performance on these two channels was excellent when testing the model against a more complex channel with many difficult turns, narrower sections and other complicated features A.3, the model could not successfully navigate this environment with the current population and generation size of 20 and 15, respectively. However, when increasing the population size to 50 and the generation size to 30, the results improved greatly, and as the performance has not converged, this suggests that with even more training time, the results would gain further performance. Even this performance shows great promise for the navigation system.

Collecting the manually navigated performance scores required multiple attempts to successfully navigate this channel without collisions, illustrating the difficulty of this channel. The average time taken for the system to navigate the channel was 705 seconds, travelling a distance of 71.4 meters. This data is comparable to the manually navigated time of 650 seconds and a distance of 67.4 meters, and further emphasises the use of this system in real-world scenarios.

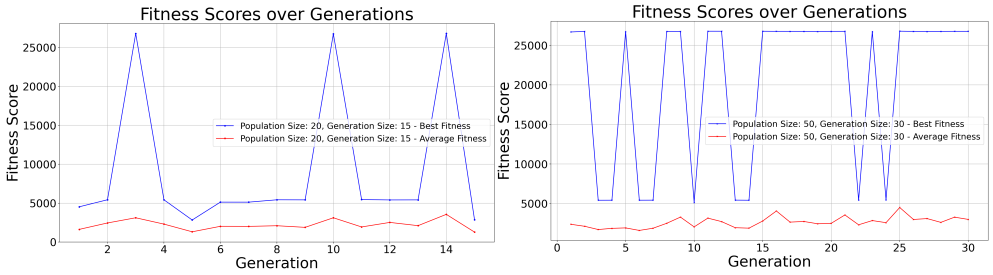

Figure 4.12: A graph displaying the performance of the model over generations on a complex channel

An important objective of this project is to create a system capable of navigating unseen channels and provide a generalised model which can perform well across various environments. The objective was met as the navigational best performance demonstrated when training on sequential channels was greater than the initial training performance. The average performance of this model ends up at approximately 9000; this is due to the majority of individuals being able to navigate the whole channel, and a few individuals not making the first turn. If further training occurred, this average would improve, and all individuals could navigate the full channel.

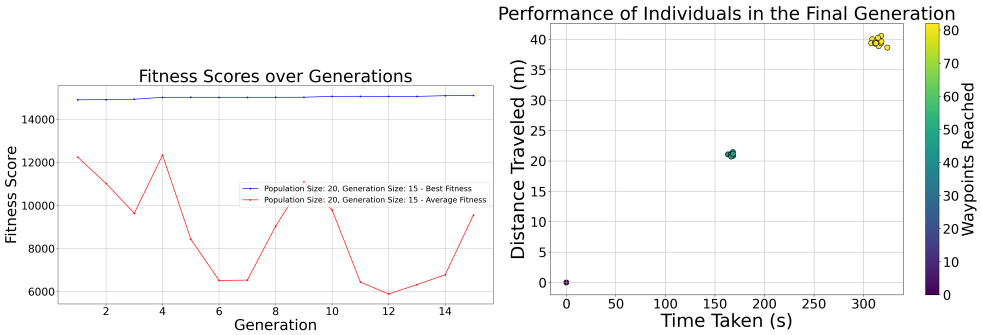

Figure 4.13: A graph displaying the performance of the model over generations on a standard channel

When testing the model on an unseen realistic channel A.4, the model could accurately learn the channel's features, leading to many successful navigations of the channel and demonstrating a generalised system that can easily be trained on any specific channel to navigate efficiently and safely 4.14.

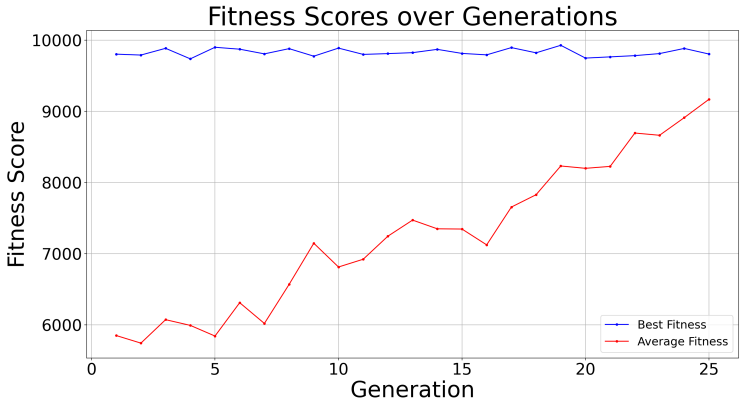

Figure 4.14: A graph displaying the performance of the model over generations on a realistic, unseen channel

5 Conclusion

This project aimed to develop an adaptive, efficient, autonomous navigation system for vessels to utilise in channels such as ports, harbours, and passages. The system leverages neural networks evolved with genetic algorithms to maximise performance across various metrics, ensuring safety, real-time responses, and efficiency. The resulting system provides insight into the viability of combining convolutional neural networks and evolutionary methods for nautical navigation in dynamic, complex environments.

5.1 Project Outcomes

The project outcomes are based on the performance of each architecture against the determined criteria outlined. The model was thoroughly evaluated based on metrics determined by the pre-defined success criteria, including collision avoidance, navigational smoothness and efficiency, and navigational progression of the channel.

The final system not only navigates a simple channel but also exceeds this objective with the successful navigation of more complex channels with a variety of complicated features and components. This navigation of complex environments illustrates further success for this system as it can accurately avoid collisions to ensure the vessel's safety, which is this project's most important objective. Using physics-based simulations, the results collected establish an underlying framework for practical applications of MASS (Maritime Autonomous Surface Ships).

Another essential objective of this project was developing a generalised system capable of safely navigating various channels with various environments it had not encountered before. Due to the system's training process, the system could accurately detect features such as turns, outcrops in the channel walls, stacks and various other features, allowing the system to navigate unseen channels successfully. In real-world scenarios, to ensure safety before the system would be deployed to a new channel, it would be trained to capture the channel's features to improve performance; however, the ability to successfully navigate an unseen channel creates a baseline to further train the system on specific channels, which would ultimately

improve performance. This would also reduce the vulnerability of vessels to navigational disruption, as demonstrated by the Ever Given incident in the Suez Canal.

5.2 Limitations and Future Work

An objective this project could not complete was evaluating the system's performance in various weather conditions, such as different wind speeds and directions and current strengths and directions. All channels are affected by these factors, and evaluation against these metrics would provide invaluable insight into real-world performance. Further testing into these environmental factors would greatly improve the system's performance in real-world scenarios and provide insights to evaluate the performance across all areas. If the system could successfully navigate a channel in various weather conditions, then the system would successfully meet all the pre-determined success criteria and outcomes.

Another objective this project did not complete was testing and evaluating the system's performance in different real-world scenarios. This would have provided important data for comparison between other real-world systems, gaining insight into the system's real-world performance relative to these systems. As this was an ambitious objective with a low likelihood of occurring, this does not affect the project's overall outcome, as the previously completed objectives were more influential and impactful to the project's performance.

Although these simulations utilised a land-based robot to conduct navigation, the insights provided are still relevant as the motion controls reflect those of a nautical vessel. If this project could be extended, the focus of the evaluation would further align with real-world scenarios by initially training and testing the land-based model in the real world, which would provide essential performance data to evaluate the model. Further simulations based on a nautical environment with a vessel would provide much-needed data to gain additional insights into real-world performance in specific nautical scenarios.

Overall, this project addresses the challenges of autonomous navigation while ensuring that a balance between safety and efficiency is maintained. This project also provides a baseline for further developments to expand and improve performance in the real-world application of autonomous nautical navigation.

A Appendix

A.1 Navigation Channel Designs

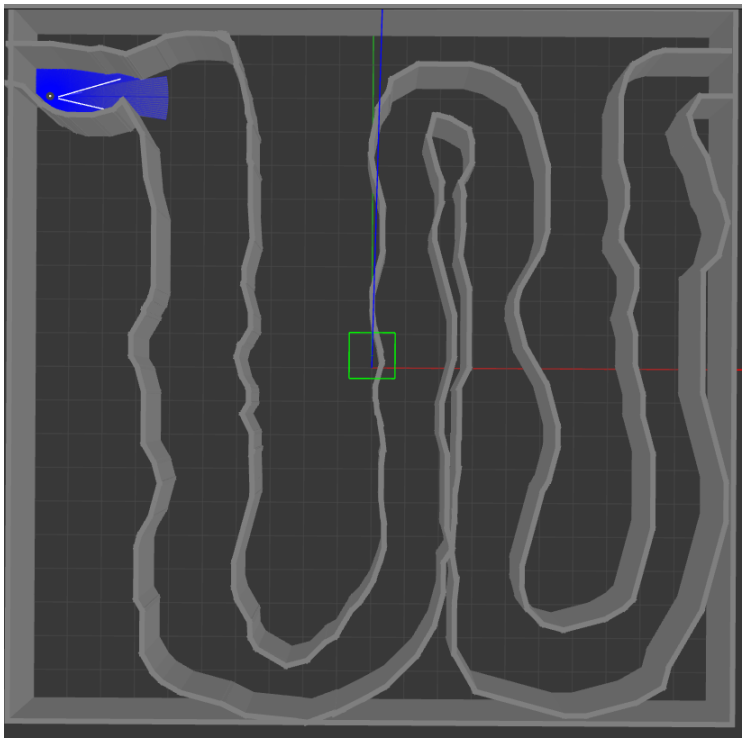

Figure A.1: Standard Channel Design

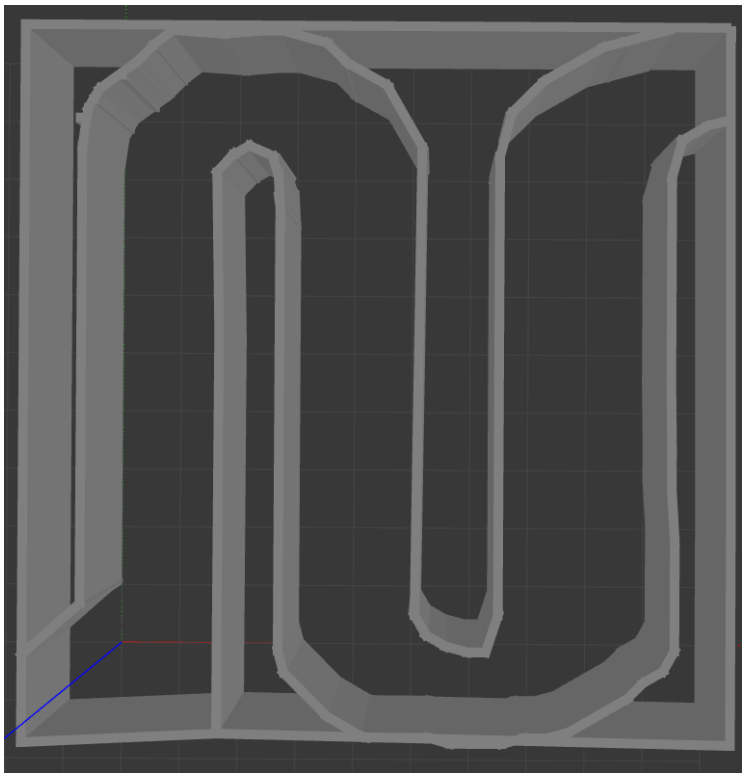

Figure A.2: Basic Channel Design

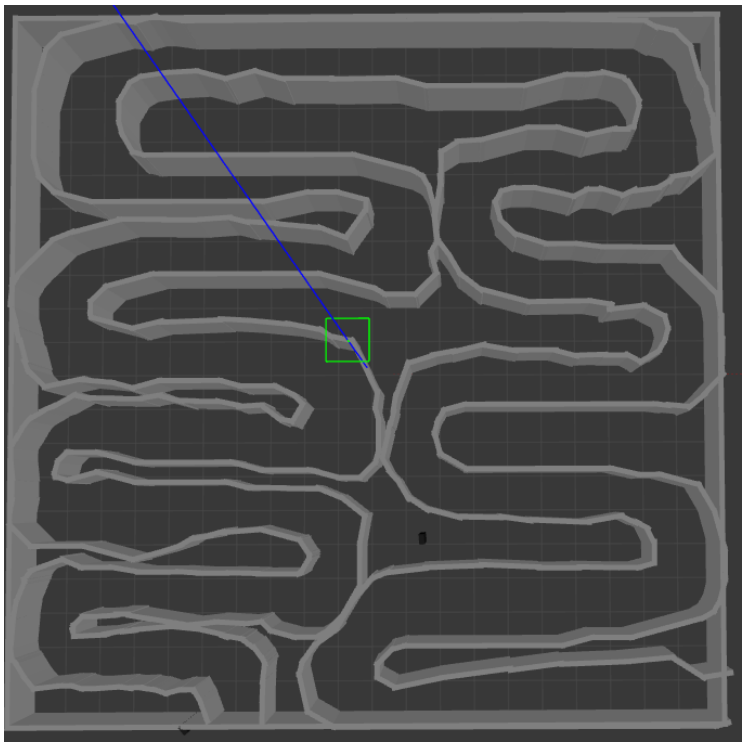

Figure A.3: Complex Channel Design

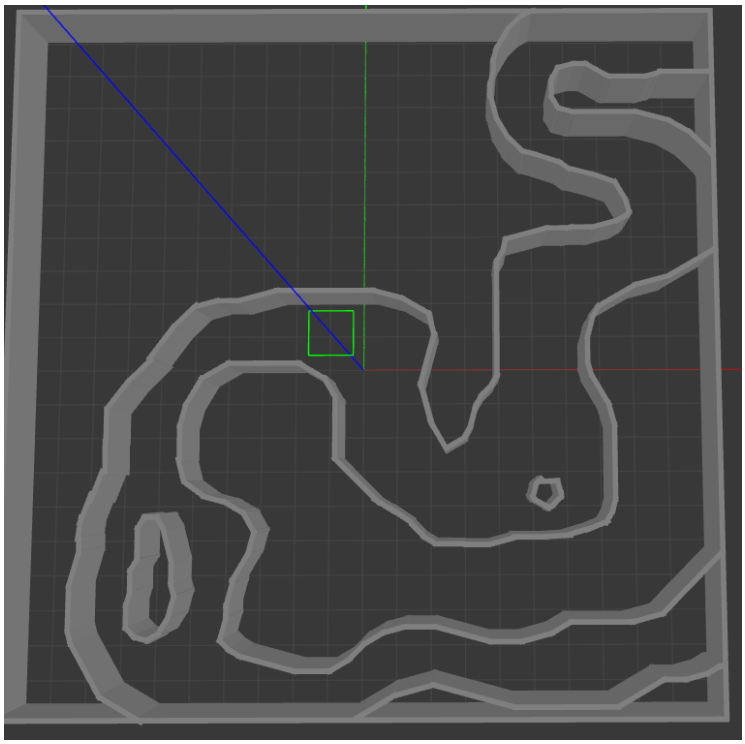

Figure A.4: Realistic Channel Design

A.2 Neural Network Designs

```
Class DDPGAgent:
    Method: Initialise
        - Initialise actor and critic networks
        - Initialise target networks (copy weights from the main networks)
        - Initialise optimisers for actor and critic
        - Initialise replay buffer
        - Set parameters for batch size, learning rates, etc.
        - Initialise target network update rate (tau)

    Method: Update
        - If the buffer has enough samples:
            - Sample a batch from the replay buffer
            - Update critic network:
                - Calculate the target Q-value using target actor and critic networks
                - Compute loss between predicted Q-value and target Q-value
                - Update critic network using the loss
            - Update actor network:
                - Compute loss by maximising the Q-value with the current actor and critic
                - Update actor network using the loss
            - Soft update target networks using tau
            - Step optimisers for both actor and critic

    Method: Act
        - Input: current state
        - Pass state through actor network to generate action
        - Optionally add noise for exploration
        - Clip action within valid action range
        - Return action

    Method: To
        - Move actor and critic networks to the specified device (CPU or GPU)
```

Figure A.5: DDPG Agent Pseudocode

```python
class Actor(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(Actor, self).__init__()
        self.module = nn.Sequential(
            nn.Linear(state_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, action_dim),
            nn.Tanh()
        )

    def forward(self, state):
        return self.module(state)
```

```python
class Critic(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(Critic, self).__init__()
        self.module =nn.Sequential(
            nn.Linear(state_dim + action_dim, 128),
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, action_dim)
        )

    def forward(self, state, action):
        return self.module(torch.cat([state, action], 1))
```

Figure A.6: Actor and Critic Neural Network Architectures

```
# Critic loss
with torch.no_grad():
    next_actions  = self.actor_target(next_state)
    target_q = self.critic_target(next_state, next_actions)
    target_q = reward + (1 - done) * self.gamma * target_q

current_q = self.critic(state, action)
critic_loss = nn.MSELoss()(current_q, target_q)
self.critic_optimizer.zero_grad()
critic_loss.backward()
self.critic_optimizer.step()

# Actor loss
actor_loss = -self.critic(state, self.actor(state)).mean()
self.actor_optimizer.zero_grad()
actor_loss.backward()
self.actor_optimizer.step()

for param, target_param in zip(self.critic.parameters(), self.critic_target.parameters()):
    target_param.data.copy_(self.tau * param.data + (1 - self.tau) * target_param.data)
for param, target_param in zip(self.actor.parameters(), self.actor_target.parameters()):
    target_param.data.copy_(self.tau * param.data + (1 - self.tau) * target_param.data)
```

Figure A.7: Actor and Critic Loss Functions and Soft Update Functions

```
class ReplayBuffer():
    def __init__(self, capacity):
        self.buffer = deque(maxlen=capacity)

    def push(self, state, action, reward, next_state, done):
        self.buffer.append((state, action, reward, next_state, done))

    def sample(self, batch_size, device):
        states, actions, rewards, next_states, dones = zip(*random.sample(self.buffer, batch_size))
        return (torch.tensor(np.array(states)).to(device),
                torch.tensor(np.array(actions)).to(device),
                torch.tensor(np.array(rewards)).unsqueeze(1).to(device),
                torch.tensor(np.array(next_states)).to(device),
                torch.tensor(np.array(dones)).unsqueeze(1).to(device))

    def __len__(self):
        return len(self.buffer)
```

Figure A.8: Replay Buffer Architectures

```
Class NeuralNetwork:
    Method: Initialise:
       - Define layers for input, hidden layers, and output
       - Initialise layers with appropriate activation and normalisation functions

    Method: Forward:
       - Process input through network layers
       - Return Output

    Method: To:
       - Move neural network to the specified device (CPU or GPU)
```

Figure A.9: Feed Forward Neural Network Pseudocode

```python
class NeuralNetwork(nn.Module):
    def __init__(self, input_dim, output_dim, device, hidden_dim=128):
        super(NeuralNetwork, self).__init__()

        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, 64),
            nn.LayerNorm(64),
            nn.ReLU(),
            nn.Linear(64, output_dim),
            nn.Tanh()
        )
        self.device = device

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

    def to(self, device):
        self.device = device
        self.model.to(device)
        return self
```

Figure A.10: Basic Neural Network Architecture

```
class NeuralNetwork(nn.Module):
    def __init__(self):
        super(NeuralNetwork, self).__init__()

        self.CNN = nn.Sequential(
            nn.Conv1d(1, 16, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(16),
            nn.LeakyReLU(0.1),

            nn.Conv1d(16, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(32),
            nn.LeakyReLU(0.1),

            nn.Conv1d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(64),
            nn.LeakyReLU(0.1),

            nn.Conv1d(64, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(0.1),

            nn.Conv1d(128, 128, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm1d(128),
            nn.LeakyReLU(0.1),
        )

        with torch.no_grad():
            dummy = torch.randn(1, 1, 360)
            dummy_output = self.CNN(dummy)
            self.flattened_dim = dummy_output.size(1) * dummy_output.size(2)

        # self.lstm = nn.LSTM(input_size=self.flattened_dim, hidden_size=64, batch_first=True)

        self.FC = nn.Sequential(
            nn.Linear(self.flattened_dim, 32),
            nn.Dropout(0.2),
            nn.LeakyReLU(0.1),
            nn.Linear(32, 16),
            nn.Dropout(0.2),
            nn.LeakyReLU(0.1),
            nn.Linear(16, 2),
            nn.Tanh()
        )
```

Figure A.11: Convolutional Neural Network Architecture

A.3 Genetic Algorithm Code

```python
def select_parents(self, population, fitness_scores, num_parents, tournament_size=3):
    sorted_population = sorted(zip(population, fitness_scores), key=lambda x: x[1], reverse=True)
    top_population = sorted_population[:len(sorted_population) // 4]
    parents = []
    for _ in range(num_parents):
        candidates = random.sample(top_population, tournament_size)
        winner = max(candidates, key=lambda x: x[1])[0]
        parents.append(winner)
    return parents
```

Figure A.12: Parent Selection Code

```python
def crossover(self, parents):
    child = NeuralNetwork().to(self.device)
    alpha = random.uniform(0.3, 0.7)

    for parent1, parent2, new_child in zip(parents[0].parameters(), parents[1].parameters(), child.parameters()):
        new_child.data = alpha * parent1.data + (1 - alpha) * parent2.data

    return child

def mutate(self, child, generation):
    decay_factor = max(0.5, 1.0 - generation / 100)
    mutation_strength = 0.05 * decay_factor

    with torch.no_grad():
        for param in list(child.parameters()):
            if random.random() < self.mutation_rate:
                param.add_(torch.randn_like(param) * mutation_strength * param.std().item())

    return child
```

Figure A.13: Crossover and Mutation Functions

```
self.metrics["distance_traveled"] += abs(linear) * 0.1
self.metrics["time_taken"] = (self.last_cmd_time - self.start_time).nanoseconds / 1e9
self.metrics['distance_from_goal'] = math.sqrt((self.goal_position_x - self.pose.position.x)**2 + (self.goal_position_y - self.pose.position.y)**2)
self.metrics['distance_from_start'] = math.sqrt((self.start_position_x - self.pose.position.x)**2 + (self.start_position_y - self.pose.position.y)**2)
self.metrics['total_linear_velocity'] += abs(linear) * 0.1
self.metrics['total_angular_velocity'] += abs(angular) * 0.1
self.metrics['oscillation_count'] += osciallation_count

distances = [np.linalg.norm(np.array([self.pose.position.x, self.pose.position.y]) - np.array(wp)) for wp in self.waypoints]
nearest_index = np.argmin(distances)
if nearest_index + 1 == self.metrics['waypoint_reached']:
    self.waypoint_count += 1
else:
    self.waypoint_count = 0
    self.metrics['waypoint_reached'] = nearest_index + 1

if self.metrics['distance_from_goal'] <= 0.5:
    self.metrics['reached_goal'] = True
    self.terminate_simulation('Goal Reached')
    return

# if self.metrics['oscillation_count'] == 25:
#     self.terminate_simulation('Too Many Oscillations')
#     return

if self.waypoint_count > 600:
    self.metrics['stuck_at_waypoint'] = True
    self.terminate_simulation('Stuck at Waypoint')
    return
```

Figure A.14: Metrics Collection Process

```
self.metrics = {"distance_traveled": 0,
                "collisions": 0,
                "time_taken": 0,
                'distance_from_goal': float('inf'),
                'reached_goal': False,
                'distance_from_start': 0,
                'total_linear_velocity': 0,
                'total_angular_velocity': 0,
                'oscillation_count': 0,
                'waypoint_reached': 0,
                'total_waypoints': len(self.waypoints) + 1,
                'stuck_at_waypoint': False
                }
```

Figure A.15: Metrics Collected throughout Navigation

```
if reached_goal:
    fitness_score = total_waypoints * weights["waypoint_bonus"] * 2
    fitness_score -= (time_taken * weights["time_weight"]) * waypoint_progress
    fitness_score -= (distance_travelled * weights["distance_travelled_weight"]) * waypoint_progress
    # fitness_score *= movement_efficiency * weights["movement_efficiency_weight"]
    if oscillation_count > 2:
        fitness_score -= (1/(oscillation_count * weights["oscillation_count_weight"])) * waypoint_progress
    if collisions > 0:
        fitness_score /= collisions * 2

else:
    fitness_score = waypoint_reached * weights["waypoint_bonus"]
    fitness_score -= (time_taken * weights["time_weight"]) * waypoint_progress
    fitness_score -= (distance_travelled * weights["distance_travelled_weight"]) * waypoint_progress
    # fitness_score *= movement_efficiency * weights["movement_efficiency_weight"]
    # fitness_score += total_linear_velocity * weights["forward_movement_weight"]
    if oscillation_count > 2:
        fitness_score -= (1/(oscillation_count * weights["oscillation_count_weight"])) * waypoint_progress
    if collisions > 0:
        fitness_score /= collisions * 2
    if stuck_at_waypoint:
        fitness_score *= 0.75
```

Figure A.16: Fitness Calculations for simulations

A.4 Additional Performance Data

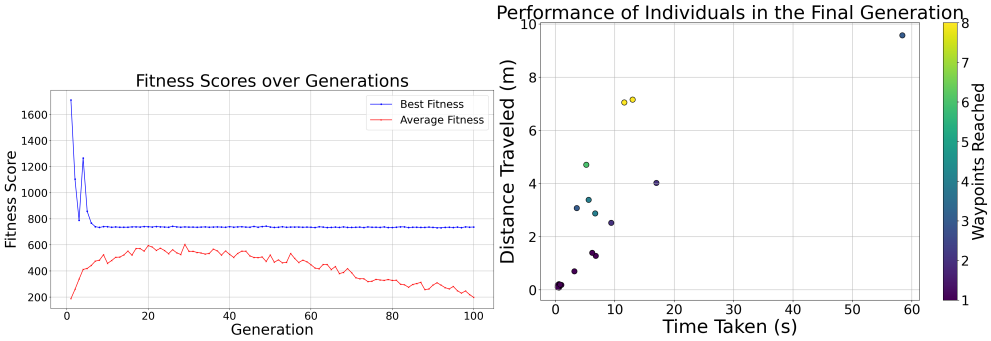

Figure A.17: Additional graphs showing the plateaus of the DDPG neural network

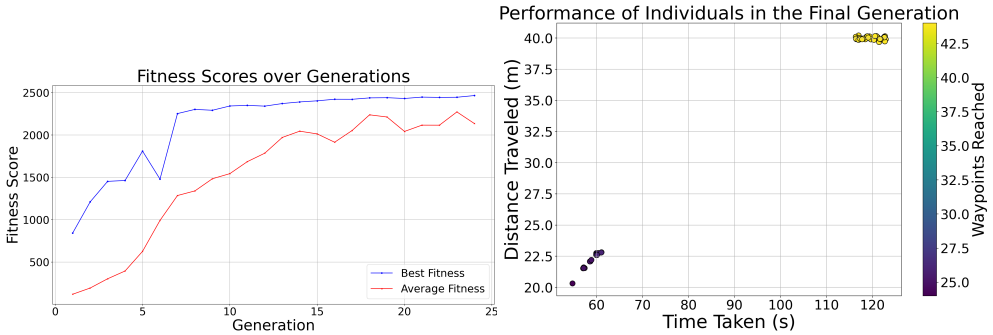

Figure A.18: Additional graphs showing a feed-forward neural network stuck in local optima

Bibliography

[1] U. Nations, *Review of maritime transport 2024: Navigating maritime chokepoints*, Un-ilibrary.org, Oct. 2024. [Online]. Available: https://www.un-ilibrary.org/content/books/9789211065923/read (visited on 19/11/2024).

[2] M. Boviatsis, 'M/v ever given: Legal assessment of the causes and consequences of the accident,' *WIT Transactions on The Built Environment*, Nov. 2022. DOI: 10.2495/umt220161. [Online]. Available: https://www.witpress.com/Secure/elibrary/papers/UMT22/UMT22016FU1.pdf.

[3] Z. Wan, Y. Su, Z. Li, X. Zhang, Q. Zhang and Z. Wan, 'Analysis of the impact of suez canal blockage on the global shipping network,' *Ocean & Coastal Management*, vol. 245, pp. 106 868–106 868, Nov. 2023. DOI: 10.1016/j.ocecoaman.2023.106868. (visited on 19/11/2024).

[4] J. M.-y. Lee and E. Y.-c. Wong, 'Suez canal blockage: An analysis of legal impact, risks and liabilities to the global supply chain,' *MATEC Web of Conferences*, vol. 339, p. 01 019, 2021. DOI: 10.1051/matecconf/202133901019. [Online]. Available: https://www.matec-conferences.org/articles/matecconf/pdf/2021/08/matecconf_istsml2021_01019.pdf.

[5] F. Boats, *Navigation at sea: From stars to the modern gps*, Formula Boats, Jun. 2019. [Online]. Available: https://www.formulaboats.com/blog/history-of-navigation-at-sea-from-stars-to-the-modern-day-gps/.

[6] E. W. Anderson, *Navigation | technology*. 2019. [Online]. Available: https://www.britannica.com/technology/navigation-technology.

[7] A. Tikkanen, *Wind rose | meteorology*, Encyclopedia Britannica. [Online]. Available: https://www.britannica.com/science/wind-rose.

[8] A. Tikkanen, *Portolan chart | britannica*, www.britannica.com. [Online]. Available: https://www.britannica.com/technology/portolan-chart.

[9] *Latitude and longitude*, Open Learning, 2019. [Online]. Available: https://www.open.edu/openlearn/history-the-arts/history/history-science-technology-and-medicine/history-science/latitude-and-longitude/.

[10] T. Lake, *In-depth: The microscopic magic of h4, harrison's first sea watch | sjx watches*, watchesbysjx.com, Sep. 2019. [Online]. Available: https://watchesbysjx.com/2019/09/john-harrison-marine-chronometer-h4-diamond-pallets.html.

[11] *Dead reckoning | navigation*, Encyclopedia Britannica. [Online]. Available: https://www.britannica.com/technology/dead-reckoning-navigation.

[12] *gyrocompass | Alignment Development | Britannica*. 2019. [Online]. Available: https://www.britannica.com/technology/gyrocompass.

[13] N. Greenwood, *From paper charts to satellites: A journey through the evolution of marine navigation*, Clearseas.org, 2024. [Online]. Available: https://clearseas.org/insights/evolution-of-marine-navigation/.

[14] Hexagon, *What are global navigation satellite systems?* novatel.com, 2024. [Online]. Available: https://novatel.com/tech-talk/an-introduction-to-gnss/what-are-global-navigation-satellite-systems-gnss.

[15] O. of Coast Survey National Oceanic and A. Administration, *Nautical chart data built for modern navigational systems*, nauticalcharts.noaa.gov. [Online]. Available: https://nauticalcharts.noaa.gov/charts/noaa-enc.html.

[16] K. C, *30 types of navigation equipment and resources used onboard modern ships*, Marine Insight, Mar. 2019. [Online]. Available: https://www.marineinsight.com/marine-navigation/30-types-of-navigational-equipment-and-resources-used-onboard-modern-ships/.

[17] *Understanding the autopilot system on ships - sea news*, Sea News - Global Maritime News, Mar. 2018. [Online]. Available: https://seanews.co.uk/shipping-news/understanding-the-autopilot-system-on-ships/ (visited on 21/11/2024).

[18] *Port pilots: Experience and poise in all conditions – professional mariner*, Professionalmariner.com, 2023. [Online]. Available: https://professionalmariner.com/article/port-pilots-experience-and-poise-in-all-conditions/ (visited on 21/11/2024).

[19] IMO, *Autonomous shipping*, www.imo.org, 2024. [Online]. Available: https://www.imo.org/en/MediaCentre/HotTopics/Pages/Autonomous-shipping.aspx.

[20] B. Hayden, *The state of autonomous vessels*, Workboat.com, May 2024. [Online]. Available: https://www.workboat.com/the-state-of-autonomous-vessels.

[21] R. Mckie, *Maritime autonomous surface ships (mass) and sar*, International Maritime Rescue Federation, Nov. 2023. [Online]. Available: https://www.international-maritime-rescue.org/News/maritime-autonomous-surface-ships-mass-and-sar.

[22] D. Schmidt, *Hands free: Avikus's autonomous navigation system | yachting*, Yachting, Jun. 2023. [Online]. Available: https://www.yachtingmagazine.com/electronics/avikus-autonomous-navigation-system/ (visited on 02/12/2024).

[23] *Autonomous navigation system*, Maritime Robotics, 2024. [Online]. Available: https://www.maritimerobotics.com/autonomous-navigation-system?utm_source=unmannedsystemstechnology.com&utm_medium=referral (visited on 02/12/2024).

[24] L. Chen, P. Yang, S. Li, K. Liu, K. Wang and X. Zhou, 'Online modeling and prediction of maritime autonomous surface ship maneuvering motion under ocean waves,' *Ocean engineering (Print)*, vol. 276, pp. 114 183–114 183, May 2023. DOI: 10.1016/j.oceaneng.2023.114183. (visited on 11/04/2024).

[25] S. Guo, X. Zhang, Y. Zheng and Y. Du, 'An autonomous path planning model for unmanned ships based on deep reinforcement learning,' *Sensors*, vol. 20, p. 426, Jan. 2020. DOI: 10.3390/s20020426. (visited on 26/04/2020).

[26] A. Gupta, 'Machine learning algorithms in autonomous driving,' Mar. 2018. [Online]. Available: https://www.iiot-world.com/artificial-intelligence-ml/machine-learning/machine-learning-algorithms-in-autonomous-driving/.

[27] W. Koehrsen, *Introduction to bayesian linear regression*, Medium, Apr. 2018. [Online]. Available: https://towardsdatascience.com/introduction-to-bayesian-linear-regression-e66e60791ea7.

[28] N. Beheshti, *Random forest regression*, Medium, Mar. 2022. [Online]. Available: https://towardsdatascience.com/random-forest-regression-5f605132d19d.

[29] R. Tatiwar, 'Neuroevolution: Evolving neural network with genetic algorithms,' Dec. 2023. [Online]. Available: https://medium.com/@roopal.tatiwar20/neuroevolution-evolving-neural-network-with-genetic-algorithms-8ca2165ad04c.

[30] D. Menges, T. Tengesdal and A. Rasheed, 'Nonlinear model predictive control for enhanced navigation of autonomous surface vessels,' 2024. [Online]. Available: https://arxiv.org/abs/2403.19028 (visited on 04/12/2024).

[31] C. Lee, D. Chung, J. Kim and J. Kim, 'Nonlinear model predictive control with obstacle avoidance constraints for autonomous navigation in a canal environment,' 2023. [Online]. Available: https://arxiv.org/abs/2307.09845.

[32] J. C. Meyers, T. S. McCord, Z. Zhang and H. Singh, 'Towards a colregs compliant autonomous surface vessel in a constrained channel,' 2022. [Online]. Available: https://arxiv.org/abs/2204.12906 (visited on 04/12/2024).

[33] N. Hamid, W. Dharmawan and H. Nambo, 'Neural network-based genetic algorithm for autonomous boat pathfinding,' *Neural Network-based Genetic Algorithm for Autonomous Boat Pathfinding*, pp. 497–502, Aug. 2023. DOI: 10.1109/ickii58656.2023.10332606. [Online]. Available: https://ieeexplore.ieee.org/document/10332606 (visited on 04/12/2024).

[34] S. Grigorescu, B. Trasnea, L. Marina, A. Vasilcoi and T. Cocias, 'Neurotrajectory: A neuroevolutionary approach to local state trajectory learning for autonomous vehicles,' 2019. [Online]. Available: https://arxiv.org/abs/1906.10971 (visited on 04/12/2024).

[35] F. Stapleton, E. Galván, G. Sistu and S. Yogamani, 'Neuroevolutionary multi-objective approaches to trajectory prediction in autonomous vehicles,' 2022. [Online]. Available: https://arxiv.org/abs/2205.02105 (visited on 04/12/2024).

[36] M. M. Zarrar, Q. Weng, B. Yerjan, A. Soyyigit and H. Yun, 'Tinylidarnet: 2d lidar-based end-to-end deep learning model for f1tenth autonomous racing,' 2023. [Online]. Available: https://arxiv.org/html/2410.07447 (visited on 26/03/2025).

[37] O. Robotics, *Gazebo*, gazebosim.org. [Online]. Available: https://gazebosim.org/home.

[38] *Ros: Home*, ros.org. [Online]. Available: https://ros.org/.