MayFlower - The Unmanned Surface Vehicle Development Framework

Group Report

NAJAT BAQADIR, JUNDA HE, ALEKSANDAR SAŠA JANJANIN, ZIWEI LIN, ROMAN MATIOS, and SIGURDUR BJARNI SIGURDSSON

This paper introduces a framework aiding the development of autonomous boats (Unmanned Surface Vehicles – USVs) for use on inland enclosed water surfaces such as canals, and defines a USV software (USV System) needed to make the boat operate autonomously. Together, these lay the foundation for future work - a broader platform extended to rivers and seas. Our contributions include 1) a Simulator for testing the solution in physically realistic environment, 2) a USV System which allows the boat to operate autonomously, 3) a back-end server for storing and processing data, and 4) a Web Platform for controlling the boat and monitoring its readings. With this, future developers can overcome initial obstacles, and use our application-agnostic solution as a stepping stone towards their use-case goals. Moreover, the solution allows the testing of projects in a simulated environment before moving over to the real world. Lastly, we assess possible ethical issues related to the development of autonomous vehicles based on our solution. In the future, this platform could become a counterpart to land and air autonomous vehicle development frameworks, thus encouraging related research.

1 Introduction

The growing interest in autonomous land vehicles generated by both the public and the government are apparent in the world today [Bansal et al. 2016; Commission 2017]. Similarly, concepts such as aircraft auto-pilot have existed for a long time and have become well-known examples of aircraft autonomy. However, this trend is less noticeable for vehicles which move on the water (Unmanned Surface Vehicles - USV). Motivated by this, IBM[1] and their partner company Deeper Than Blue[2] proposed this project in order to fill the existing gap.

To better explain this domain and our work, we adopted the terminology and definitions from the more mature land vehicle domain. It is important to mention that vehicle autonomy is categorised into levels 0 through 5, implying the requirement of full manual input and full autonomy without the need for a driver, respectively [On-Road Automated Driving (ORAD) committee 2018]. Irrespective of the level of autonomy, the vehicle should be able to reach a given destination safely by sensing the environment, planning the route and acting accordingly. Throughout this section we will aim to define the objectives and the constraints of the proposed system.

1.1 Objectives

The project's initial goal was to develop software for a USV which would operate on enclosed inland water surfaces such as canals, and would survey the environment. Moreover, due to the importance of testing autonomous vehicles within a Software-in-a-Loop Simulator environment [Schöner 2018] and the COVID-19 pandemic complications, the project should also address the need for a physically correct simulator where the said USV System could be tested. Generally, this would serve as a demonstration of the technical abilities of the two companies.

Through requirements elicitation, we have moved from an initially loosely defined, flexible set of ideas and requirements to a broad but precise scope. We familiarised ourselves with the places of the most significant potential impact, thus further adjusting both the goal and the scope. Due to the nature of the project, we aimed to address not only the requirements but also the underlying

[1] https://www.ibm.com/uk-en
[2] https://www.deeperthanblue.co.uk/

need. The redefined goal was to engineer a framework which provides all building blocks and addresses all aspects of the USV system development process: 1) a simulator along with a range of sensors, for testing the solution in physically realistic conditions and environment, 2) a USV system which allows the boat to operate autonomously, 3) a back-end server for data storing and processing, and 4) a web interface for controlling the boat and monitoring its readings. All four components are intended to be flexible so that the user can choose only the necessary modules and exclude or replace the rest. Furthermore, we intended to use this framework to develop the USV for surveying the environment in a canal-like setting.

Our goal therefore became to develop an application-agnostic framework which can be quickly extended, so that more domain-specific USVs can be developed more efficiently. Moreover, rather than focusing only on the initially specified canal-surveying USV, we set the project on a course which is sustainable and long-reaching. We hope that it will encourage future development and expansion of additional features for both existing and new USVs.

1.2 Constraints

In order to maintain a manageable scope, we have refined the ambiguous requirements, mainly concerning the environment and the level of the USV autonomy. As a first step, we agreed that our solution should support a boat of any dimensions and shape, but for development and testing purposes, we used a model whose dimensions are roughly 2 meters in length and 1 meter in width.

The primary intention of IBM and Deeper Than Blue was to display and use the boat in the Sheffield canal. This allowed us to focus on a reasonably complex, yet well-defined environment. Consequently, we focused on developing a solution for a water area 1) defined by flat vertical walls, 2) without many vessels disturbing the surface, and 3) with winds of moderate strength. Together, these produce relatively small waves when combined with a shallow and narrow canal at a moderate distance from open water. This way, the weather conditions taken into account should not fluctuate significantly. Canal bounds would be readily observable, and a wireless network infrastructure for the communication with the boat would be in the vicinity and available. Lastly, we have agreed to omit water locks from the scope as including them would be too complex.

Since the objective of the USV System is to survey the environment, it should be able to make multi-hour trips within the canal bounds. Consequently, expecting a human controller to monitor the actions of the boat would not only be highly impractical but would not address the idea behind the project. This demands developing a system capable of handling everything autonomously, but only under a specific set of constraints (i.e. location and weather conditions). If something in the environment changes drastically enough to compromise the boat, we assume that the change could be observed with enough time to prevent the boat from commencing the mission.

Together, these constraints make a complicated problem solvable while maintaining the validity within the scope set by the requirements. Moreover, they can gradually be relaxed over time as our platform grows and matures.

2 Background

Our research of the Autonomous Vehicle (AV) domain produced good results, and we have identified research related to our field. Below, we survey the state of the art within the research domain.

Looking at the autonomous vehicles in general, we identified four significant Autonomous Vehicle (AV) development frameworks which are similar to what we aim to achieve. These are Airsim [Shah et al. 2017], Carla [Dosovitskiy et al. 2017], Autoware [Kato et al. 2015, 2018] and Apollo [Auto 2020]. Amongst these only the first two have their own environment simulators, while others rely on a separate project [Lab 2020]. Furthermore, all these frameworks focus only on land vehicles, except Airsim which also considers air vehicles.

To overcome this and combine them with our project, one would have to add proper and adjustable water surfaces, canals, boats, water navigation algorithms, additional sensors and connect everything properly into the existing codebase. For these reasons, as well as flexibility and the requirements, it was not feasible to resort to this approach. Therefore, we aim for our work to coexist with these solutions and be their counterpart focused on USV development. For this, we have further researched the underlying technologies needed to tackle this task.

The Robot Operating System (ROS) [3] – a standard open-source solution for programming robots – is a useful framework for the needs of the USV system. It serves as a middle layer and allows for all of the robot's (in our case USV's) modules to be connected into a single system and considered as nodes within it. It is important to highlight that the framework is mature and well-maintained by a large active community, thereby proving itself a perfect long-term solution.

Examining technologies used to simulate the environment and the ROS-based USV system, one of the popular tools available is Gazebo [4] – an established robotics simulation tool. Other more flexible tools include Unity[5] and Unreal Engine (UE)[6]. It is worth noting that, despite flexibility, they are not standardised tools and require creating the environment, supporting the communication with ROS, and making a substantial number of adjustments in order to produce the intended results. Nonetheless, Gazebo is only a physics engine, while the latter two are game engines, allowing them two be deployed unobstructedly. Moreover, out of the three, only Gazebo does not have adequate water surface support.

Between the game engines, UE has a much steeper learning curve but is more realistic and precise, yet Unity is more novice-friendly, thus requiring less time to reach a goal. Although Unity and UE are not designed with a primary goal of being compatible with ROS, as opposed to Gazebo, it is still possible to run ROS applications with them. This is done by relying on one of several projects, including a well-maintained open-source project for Unity – `RosSharp` [Bischoff 2020; Hussein et al. 2018; Mania and Beetz 2019]. Nevertheless, most approaches rely on the RosBridge suite, which establishes a Web Socket connection and acts as a bridge between the two endpoints, thenceforth abstracting them as nodes within the same ROS system [**?**].

Furthermore, while examining the existing frameworks, we have identified that they all include a way to survey the surroundings through various sensors. The most frequently discussed sensors available in both frameworks are range finders, lidars (which determine distances based on a laser's time of flight and generate corresponding 3D representations), and video feed sensors. Out of these, lidar sensor data processing can prove useful for obstacle detection, as shown by the acceptance of this technology among the car manufacturers [Zhao et al. 2019]. Also, the sensor fusion, which combines readings from multiple sensors, can be successfully used for obstacle detection and object classification [Gao et al. 2018; Kocić et al. 2018; Rosique et al. 2019; Varghese 2015].

Along with sensors, certain algorithms might prove useful in our use case. We emphasise Simultaneous Localisation and Mapping (SLAM) which allows the agent to move and simultaneously create a map of its environment and use it for navigation [Aulinas et al. 2008; Bailey and Durrant-Whyte 2006; Durrant-Whyte and Bailey 2006; Thrun and Leonard 2008]. Although it is possible to utilise SLAM using only a single camera [Davison 2003], and even successfully recover from rapid movements by applying re-localisation [Williams et al. 2007], this is far from optimal. On the other hand, combining depth data from a six degree of freedom lidar with the camera images is a flexible and scalable technique [Kohlbrecher et al. 2011; Scherer et al. 2012]. Finally, it is worth

[3] https://www.ros.org/
[4] http://gazebosim.org/
[5] https://unity.com
[6] https://www.unrealengine.com/en-US

mentioning ROS tools used for both 2D and 3D SLAM such as Open Slams's Gmapper [Grisetti et al. 2007, 2009; Grisettiyz et al. 2005] and Google's Cartographer project [Hess et al. 2016].

Finally, there has been substantial research regarding USVs recently [Liu et al. 2016]. It resulted in new approaches for detecting obstacles with the least clutter and false positives [Gal and Zeitouni 2013]; removing noise from the lidar-obtained point clouds which arose due to the light refraction caused by water and bubbles [Weon et al. 2017]; and performing SLAM by using a marine radar on the coastline [Han et al. 2019] and a multi-beam echo-sounder on the water bed (bathymetric SLAM) [Kim and Kim 2017]. Furthermore, there are some notable implementations of USVs. These were built for the exploration and mapping of rivers and their shores [Fraga et al. 2014], as an entry for a competition while supporting obstacle avoidance and docking [Kang et al. 2015], and as an autonomous platform for other autonomous underwater and aerial vehicles [Sinisterra et al. 2017].

3 High-Level Design

In order to achieve the objectives specified in Section 1.1, we first designed a high-level architecture of our framework and subsequently used it to develop the USV system. The high-level design is depicted in Figure 1, and consists of four components: Simulator, USV System, Server with the Database, and Web Platform. Here, the first three communicate via a WebSocket (A HTTP-alternative communication protocol relying on TCP which maintains a connection between endpoints upon a successful handshake). In contrast, the last two communicate via a RESTful API. Although the former performs slightly worse than just a TCP protocol (due to operating on a layer above – e.t. the application layer), it is reliable and is the default for RosBridge. Also, it performs well during longer connections as in our case [Skvorc et al. 2014]. Altogether, we use WebSocket to communicate the USV-specific information and the API to communicate with the end-users.

Throughout the remainder of this section, we cover each of the four components one by one, and further expand on the objectives defined in Section 1.1. We also lay out the system in a way that builds on top of the existing work, whilst the Lower-level details are discussed in Section 4.

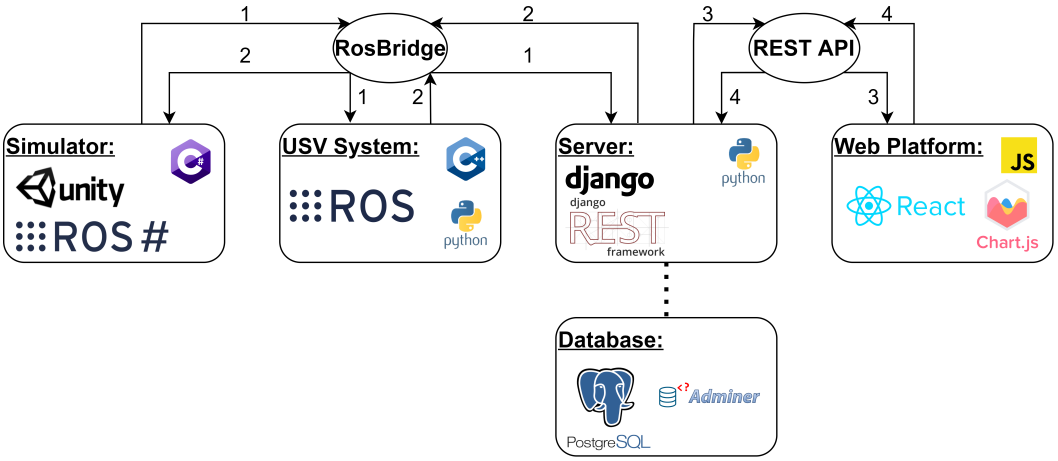

Fig. 1. High-level design of the framework. Information is transferred through arrows: 1) sensor readings, 2) USV instructions, 3) current and historical sensor readings, 4) user instructions

3.1 USV System

We build the USV system using the functionality which ROS provides, and create a RosBridge node to enable its communication with external components, such as sensors in the Simulator. We split

the code into modules according to their purpose into 1) battery management, 2) movement, 3) obstacle avoidance, 4) navigation, 5) autonomous movement, and 6) environment sensing. Internally, they interact among themselves directly trough ROS topics and are designed to be reusable and extensible. As an example, we apply the Facade design pattern to hide the implementation of controls and the interaction with the actuators.

3.2 Simulator

The most impactful decision is to use a game engine rather than Gazebo. Specifically, we choose to use Unity because of a less steep learning curve. The choice is further motivated by the benefits of flexibility in deployment as a standalone application, adjustment of the physics, and modelling the water environment. Additionally, the Simulator includes the RosSharp tool for working with ROS in C#, to support ROS code and communicate with RosBridge within the USV System.

Through using Unity, we are able to speed-up the development of the environment and improve its quality by incorporating the open-source BoatAttack project. The original objective of this project is to demonstrate the Universal Rendering Pipeline (URP) within a boat racing game [McGrail 2020]. The URP is capable of optimising graphics across platforms and creating the visuals quickly and efficiently, therefore fitting perfectly into our project.

This approach allows us to focus on the `MayFlower` package, which is separated from other assets in the project to emphasise modularity and reusability. Therefore, we split this module's functionality into three categories: environmental data, mock sensors, and the physics of the simulator. Since the only coupling is between the sensors and the USV System through ROS topics, the solution does not require changes when transitioning from the simulation to the real world.

Overall, the simulator is responsible for exposing the data available in the environment, which the USV can obtain through sensors, and for creating a realistic environment. Moreover, we allow for the use of Unity Simulation[7] for training, testing and validating AI models in the cloud.

3.3 Server

Throughout the project, we abstract from the size of the vessel and its computational power. Consequently, there is a need for centralised processing and storing of data. To increase the independence and reliability of the USV, we utilise the Server as a single point of contact for communicating with the end-users. It preserves the data more safely, in a relational database, and can process it faster. Since USV and ROS are written in Python and C++, utilisation of the popular Python framework Django [DjangoSoftwareFoundation 2020] is a natural decision. Moreover, we utilise the Django REST framework [EncodeOSSLtd 2020] for writing the REST API. Potentially, the project can benefit from machine learning frameworks such as PyTorch [PyTorch Team 2020] and TensorFlow [TensorFlow Team 2020].

3.4 Web Platform

The last component of the system is the Web Platform, which can be described by two use-cases. Firstly, it provides a control centre where the administrators in charge of the USV can monitor the readings in real-time and issue overriding commands. Secondly, the platform gives an overview of the historical data of the USV's sensor readings. Overall, it has an intuitive and understandable layout, and does not burden the network because of lazy loading. We address the use-cases modularly enough, so that future developers can quickly select only the elements of interest. We opted for the React JavaScript framework [Facebook 2020] due to its well-maintained code and support for various devices.

[7] https://unity.com/products/unity-simulation

3.5 Control Flow

The USV System is a core component of our framework and can be run in the real-world without relying on the Simulator. It has modules responsible for the domain-specific goals, including movement, navigation and environment interaction. As such, it is sufficient for a functioning hardware USV. Nonetheless, we expect that in most cases it will rely on a Server to provide additional resources, flexibility and a single point of access.

If this solution does need to be tested virtually, it can be done by connecting to the Simulator component. For this, one needs to add prefabricated models with sensors, and establish a connection with RosBridge running on the USV System. Transitioning between the virtual and real environment requires the corresponding actuators to subscribe to ROS topics published by the USV System.

Provided there is a need to present a data overview or give a convenient way to control the USV, the Web Platform can be connected. Upon utilising needed elements within it, it will be ready to establish the API connection with the rest of the system.

4 Low-Level Design

When defining the overall architecture, we need to specify the system's inner workings and the way the modules operate together. In this section, we discuss the low-level design of the individual components, thereby expanding on the High-Level Design discussed in Section 3.

4.1 Simulator

Recall that the three main features of the Simulator are environmental data, mock sensors, and physics. In our code, these correspond to `MayFlower` (data and sensors) and `BoatAttack` (physics) modules. In addition, it also contains a `RosSharp` module which provides a ROS framework on which we rely within the Unity Script code.

Environment Simulation and Waves. In our solution, we rely on modelling the water surface using Gerstner waves (also known as Trochoidal Waves) which are efficient and perform well with low computational cost. We achieve this by applying the principle of superposition of random periodic functions by adjusting the wavelength, frequency, wave height and the number of waves.

The product of the simulated environment is also a visual representation of the canal, rendered through the game engine. It is available both to the developer as a view in Unity, and to the Web Platform users in the form of image feed from the boat's video camera. Screenshots from the video camera can be found in Appendix C.

Environmental Data. The scripts with relevant information within the `Environment` submodule of the `MayFlower` module encode the needed data in Unity in an easy and adjustable way. They provide input to the sensors and allow for improved control over the environmental data, and transferring it to the USV System. Each script contains fields relating to a specific environmental parameter and possibly expose methods for obtaining more complex data.

Sensors. As described, mock sensors provide an abstraction from the hardware, which allows the data to be collected when the USV System is running within the Simulator. They transmit this data in standardised ROS messages with appropriate types and content. Although this is different from the hardware sensors, which rely on drivers to encode the data from the sensors to ROS messages before publishing them, the core idea is the same. Sensor data is published to a topic where the USV System consumes it, making the two interchangeable.

We have implemented the following sensors in Unity in the `Sensors` submodule: Battery, Compass, Range Finder, GPS, Inertial Measurement Unit (IMU), 2D and 3D Lidar, Light Sensor, Temperature Sensor, Wind Sensor, and the Video Camera. We identified these as essential for the implementation of the algorithms encountered during the literature review. Importantly, extending the list requires adding a new sensor module and publishing the topic onto the ROS network.

We also made it possible to set the sensor's device-specific parameters according to the specification of the corresponding hardware device. For example, this can be a field of view or a maximum distance supported by the lidar sensors. Nonetheless, it is also vital to emphasise that we needed to optimise the scanning to compensate for running purely in software. For example, we have implemented the lidar to scan multiple directions at once at lower frequencies, instead of one direction at a time at a high frequency. As a result, this led to an optimised yet correct performance.

4.2 USV System

The USV System uses ROS as its core since it is a mature and well-supported framework. Although ROS2 has improved middleware interface, packaging and build tool, its first post-beta version was released roughly two years before our project, and neither RosBridge nor RosSharp are yet officially supported and fully compatible with it. On the other hand, ROS1 has been developed and maintained for more than ten years, thus having better community support. Hence, we decided on a sturdier and more supported ROS1 Noetic Ninjemys.

It is worth noting that we have also observed a slow adaptation of some tools for Ubuntu 20.04 which was roughly two months old at the beginning of the project. This was especially unusual as Ubuntu is a standard for ROS development. Specifically, we encountered a lack of RosBridge support for Ubuntu 20.04's new default Python 3. As this blocked communication between the Simulator, Server and USV System, we had to solve the issue. This was achieved through installing RosBridge from project source as a ROS module instead of as a package. This solution is effective but needs to be replaced when the RosBridge package is updated with the fix.

SLAM and TFs. Gmapping is the most widely used SLAM package for mobile robots and is more compatible with Noetic than Cartographer. Thus we chose it for implementing SLAM in our platform. Underlyingly, this package takes the data from the Laser Scan or Point Cloud messages and the odometry information of the boat to build an Occupancy Grid Map (OGM) using the `map_server` package. The present configuration of this package converts input data into a map image.

The `slam_gmapping` node subscribes to `/tf` and `/scan` topics, which provide the needed information. More specifically, the `/tf` topic publishes transform information of the robot frames, including laser, base, and odometry. This is essential for the system to have an estimate of the boat's position and orientation(pose) on the map. By combining the pose of the USV and the readings from sensors, the system estimates robot position in the surrounding environment.

`SLAM` packages are not fully supported by ROS Noetic, thus building the source code locally is needed. For this, `Gmapping` and `slam_gamaping` are located in the USV System's workspace. We utilised URDF that resembles the boat within the Simulator to generate five static links and one rotating. Transformations between links and odometry are handled by custom scripts. Images of transformation graph and SLAM results can be found in the Appendix C.

Controls and Movement. To control the boat, we have implemented the movement control via a ROS topic publisher-subscriber model. It uses the standard *geometry_msgs/Pose* message which expresses linear and angular velocities in free space in Vector3 format. Thereafter, the control abstraction layer listens to a predefined topic (`/boat_teleop/cmd_vel`), extracts linear velocity from the `x` field and angular velocity from the `z` field of the incoming messages and transmits these values to the `/boat/controls` topic. Messages from the latter are then applied to the boat's engine in order to accelerate/decelerate and adjust the boat's heading.

Obstacle Avoidance. In order to avoid obstacles, we have implemented five Range Finder sensors on the front of the boat, as per Figure 2. The sensors are implemented in Unity using Raytracing methods to detect a collision with an obstacle.

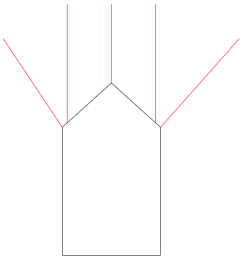

Fig. 2. Sensor placement

In the case that the sensors are triggered and detect an incoming object, the boat will change its course accordingly. More precisely, the sensors send their status to the USV System, and upon processing it, it responds with an appropriate turning information. These messages are in the form of a ROS Range Sensor message, and the sensor transmits them as soon as there is an obstacle within three meters.

However, a high-speed sensor with a three-meter detection range is mostly used for navigation by robot systems with wheel rotation based movement. This allows them to stop or move in a different direction very quickly. Since we are dealing with a boat, it has momentum in the water, making it difficult to swiftly change its course. Other factors, such as strong waves and wind, might cause the USV to pitch. This can result in lasers shooting above obstacles or into the water. Although, this is unlikely to happen since we are dealing with a constrained canal environment, it is something to consider.

Pathfinding. In the project, pathfinding works based on following the ordered GPS checkpoints predefined by a controller until a goal is reached. Underlyingly, the USV receives an array of checkpoint coordinates as a JSON message from Unity or the Web Platform. It then calculates the direction and distance of the first checkpoint, relative to the USV. The USV System then checks this new data, compares it to the compass and updates the rotational degree of the boat until it aligns with the next checkpoint. Once the USV is within five meters of the checkpoint, the checkpoint of interest is updated to the next one, until it finally reaches the intended destination.

Navigation. The general navigation algorithm utilises the Controls, Obstacle Avoidance and Path Finding modules, and gives the boat an Auto-pilot mode. Although this mode can be disabled so that manual controls are used, when it is enabled, the boat gets a constant forward thrust and listens for steering input. The module features a hierarchical structure of commands based on safety importance, starting with obstacle avoidance and followed by pathfinding. Therefore, in the case that the boat encounters an obstacle, precedence to set the heading will be given to obstacle avoidance. After clearing the obstacle, the boat resumes movement towards the next beacon.

4.3 Server

In order to abridge the gap between Django and ROS, we rely on the roslibpy library [Kohler et al. 2019] due to its lack of need for a local ROS core. This means that we can run ROS code on the Server and rely on the USV's RosBridge instance. Moreover, we implemented a micro utility framework within the Server, which supports ROS Connection, publishers and subscribers. Further, we created modules corresponding to the available sensors, handled the sensor data by persisting it into the database and provided RESTful APIs. Each module consists of 1) `model.py` for specifying data models, 2) `serializers.py` for JSON parsing, 3) `ros_listen.py` for RosBridge interaction, 4) `views.py` for defining API functions, and 5) `urls.py` for declaring the exposed endpoints' URLs.

4.4 Web Platform

As the implementation is mainly described in Section 3, here we want to emphasise the way that we utilise the natural modularity of the React framework. We implemented all the controls, readings and chart overviews supported by the USV System so that the users can select those corresponding to their use-case and USV implementation. Specifically, we implemented a component housing all the mentioned sub-components independently. A possible layout is presented in the Appendix E.

In the end, even though we make a distinction between the control centre and historical sensor reading overview, our components can be combined in any way and do not need to be separated. Moreover, due to loading the content lazily, there will be no excessive burden on the network.

5 Ethical Impact of the Solution

Due to the nature of the project, we must discuss the possible ethical consequences of our framework. This should benefit anyone using it in the future, further developing it, or encountering similar tasks. Therefore, this chapter will examine some important points which need to be considered.

Controversial Readings. Having implemented the sensors which survey the environment, our solution can spark controversy when implemented on a hardware boat. If relaying the sensor readings to the public via the Web Platform, there is no guarantee about the obtained data. Although this can also happen if the datasets encoded in the Simulator are controversial, it is less imminent and is preventable. As a fitting example, there has been recent discussion over the question of water quality in Britain's rivers. As the potentially questionable practices and views on enforcing the preservation rules might be violated, measuring and publicly displaying the results could have a high impact and may lead to legal issues [Laville 2020; Monbiot 2020].

Privacy Violation. It is nearly certain that the USV will encounter people at some point while moving along the canal. More importantly, it might pass houseboats, shores, and other vessels. Enabling the image feed can violate privacy of the people it encounters. This problem is not trivial nor easy to solve, as can be seen from Google's experience with Street View. Although it received a green light in the UK [NEWS 2009], the company 1) had to address numerous requests for removal of individuals' photos [DerSpiegel 2010], 2) sparked a discussion with a strong sentiment among the population [Salzberg 2008], 3) created controversy over publishing delicate images [Bowers 2008], and 4) had to go through numerous legal investigations [Center 2012].

Safety of Sensors. As there are numerous sensors which can be attached to the boat and which are incorporated into our project, it is important to assess their impact on the environment around them and to examine whether they are safe and acceptable. Under the assumption that the boat will pass other boats and house boats, and that it will move next to the canal edges, we can see a situation where the lidar sensors can possibly be aimed at the eyes of people and animals, or camera lenses. Depending on the laser type, it may be harmful to living beings [Automotive 2018; Hecht 2019], or optical devices [Kleinman 2019].

Malicious Use. As our framework is open-source, it can be adapted and utilised for the purposes which were not originally intended and which are malicious. It will be relatively easy to build a basic USV System capable of taking images, using intrusive sensors such as thermal imagery. However, this is an ethical and philosophical topic which does not have a definitive answer. Nevertheless, due to the ease of maintaining a USV operational and idle, less traffic, and simpler controls, the question of possible misuse is highly emphasised.

6 Evaluation

The project was quite demanding due to the nature of various technologies, languages and tools used. Therefore, it was imperative to get a solid understanding of all of them, their purpose and best practices. We tried to tackle the problem set before us with extensibility and future development in mind. Our framework is functioning and can be used for the development of USVs, as we proved by implementing the initially required USV system. Moreover, we tested the USV within our virtual Sheffield and a more complex custom canal, and it performed well and within the set constraints. Below, we break the evaluation into its main components and discuss them separately.

6.1 Impact for the Clients and Other Stakeholders

Our primary clients – IBM and Deeper Than Blue – intended for this project to give them new publicity and demonstrate the quality of their engineering and technologies. Although we did not use their existing products directly due to lack of overlap, we observed various future opportunities of integration which utilise their Cloud and other technologies. However, up to now, integrating

those with any of the four components was simply not an optimal solution. Furthermore, since the client intends the boat to be used in front of the Deeper Than Blue offices on the Sheffield canal, we tested our USV System in the simulator on that stretch of water. We are confident that they will be able to fulfil their intentions once the project moves forward with a physical implementation. Along with this, as we built a framework for further development of USV systems, it can attract attention and publicity as a counterpart to land and air autonomous vehicle development platforms.

6.2 Suitability of the solution for the use case

A good benchmark for the fitness of our project is to draw comparisons with AirSim and Carla. We decided on this as both of the frameworks are state-of-the-art, mature, and well-maintained. Additionally, we can compare how well it performs and the extent of the provided functionality. For this, we assess how helpful our framework is to the development and how many features it possesses. Additionally, we describe our experience of using it to build a USV defined in Section 1.1.

Our platform provides a total of ten sensors, as opposed to AirSim and Carla, which provide six and thirteen respectively. All three frameworks cover the core sensors such as Camera, IMU, GPS, Range Finder, and 3D lidar. As opposed to our remaining sensors described in Section 4.1, the latter two support sensors which predominantly relate to their domains. For AirSim these are Barometer and Magnetometer, while for Carla they are Collision detector, Depth camera, GNSS sensor, Lane invasion detector, Radar sensor, RSS sensor, Semantic LIDAR sensor, Semantic segmentation camera and DVS camera. The differences among sensors are mostly due to environment and purpose, where the benchmark frameworks do not have surroundings exploration in mind and focus on training self-driving cars and drones. Furthermore, both AirSim and Carla support both Unity and UE, but neither provide a base implementation of an autonomous system. Naturally, as more mature platforms they have more additional features which we could not match in such a short period.

Overall, by using the platform to develop the the USV, we validated that it supports development and performs well. It boosts the development speed by allowing engineers to skip the development of the underlying functionality common for the majority of use cases, including movement, navigation, and sensors. It also allows testing the solution without the actual hardware boat. Additionally, our solution provides a set up server which persists all the readings automatically and exposes an API for use by a web platform which has most of the components readily available.

In the project, we did not account for modelling Internet connectivity malfunctions, which needs to be addressed in real-world testing. Moreover, we strove to implement sensors which are as similar to real-world as possible, but must emphasise the underlying limitations of simulation. These primarily include the inability to match the sensors' high frequencies and hardware capabilities.

6.3 Testing

We have mainly relied on unit testing methods. For our ROS components, we created a test suite using Python's unit testing framework, and for the Simulator we relied on Unity's unit testing framework. Additionally, we have created test scenes and included in them various environments with static and dynamic obstacles that test the boat's obstacle avoidance and navigation systems. The test scenes revealed that the navigation does not work when the boat is placed in tight corridors that trigger all the laser sensors at once. However, we assume that this is unlikely to happen in a large open canal.

To test the Server, we created unit tests focusing on the data model creation and RESTful API functions. Using Django's testing tools, we generated fake clients to send requests and get responses for API endpoint testing. In total we have 77 test cases for model creation and APIs related to CRUD functions. Nonetheless, it might be advisable to examine unit testing within the Simulator and the Web Platform in more depth.

It is quite difficult to achieve full coverage throughout the project due to the need for mocking and integration testing. Because of this, we have settled with a lower coverage score.

6.4 Shortcomings and encountered issues

Despite covering much ground and providing a solid implementation, we did identify certain shortcomings. Our lack of specific hardware and URDF models meant that we could not test our model thoroughly against the future hardware implementation of the boat. This limitation meant building a generalised solution which should perform well but without a guarantee in terms of physical properties. When combined with the already massive scope, the resources were thinned down substantially, and the two factors proved to be quite demanding. Somewhat relatedly, our lack of previous knowledge and the intense tasks of creating a simulator using a game engine increased the already high level of difficulty. We could not commit to the entirely realistic Simulator graphics, nor creating custom models. Thankfully, we integrated the BoatAttack project and were significantly aided by it, but we see these points as places of further improvement.

We expect the tools to support our technology stack better over time, thus allowing us to remove the components installed within our project from source code, and replace them by packages themselves. Most important examples of this are Gmapper and RosBridge suits. Presumably, the community will also embrace the differences which Ubuntu 20.04 brought and will adapt its default Python 3, thereby allowing the utilisation of the potential of ROS2 and associated tools.

6.5 Advice for Other Software Engineers

Despite conducting thorough research and designing our solution to incorporate the most appropriate projects and best practices, there are specific points which we would do differently based on our current knowledge. Developing the solution to address the scope at hand resulted in gaining a deeper understanding of the field, and encountering the common pitfalls.

Keeping such a vast and complex project working across all development environments is a demanding task. In our technology stack, ROS requires Linux, Unity runs on Windows, macOS and Ubuntu, but is less efficient and supported on Linux. Server and Web Platform can run everywhere. Moreover, Windows platform cannot support both a virtual machine and Docker[8]. Due to this diversity being a constant struggle, we had to resort to Docker. Yet, even as a standard for virtualisation, it cannot run essential GUI applications such as RViz for visualisation of the ROS system. To solve this, we tried running the USV System component in a Virtual Machine. Nonetheless, setting the project up without Docker is a complicated and demanding task. Overall, the easiest solution with the least shortcomings is running the entire project on Ubuntu and relying on Docker. This was our experience, and we believe that depending on the task, this can change.

It is important to know that teams more experienced within the fields of Robotics and Game Development might want to consider alternatives to the tools we used. In the case that the team is confident in their skills, we advise on considering ROS2 and UE. With a strong background, former's potential issues and lack of compatibility with RosBrisge could be solved more easily, and the steep learning curve of the latter mitigated. By using ROS2, the USV System can run on Windows and utilise Python 3. It performs better for real-time tasks and has a more optimal design. In addition, UE is a more conventional tool for simulation purposes, with more realistic graphics.

7 Future Improvements

In this section, we outline a few possible future improvements which complement the fixes for shortcomings stated in Section 6.4. We split these depending on whether they alter the scope, and examine them in detail.

[8] https://www.docker.com/

7.1 Improvements Within the Current Scope

Although we completed the majority of tasks, there are still some possibilities for improvement within the set scope. Most significantly, adding new sensors and algorithms to cover a broader spectrum of use cases for which our framework could be used out-of-the-box. These could be a Sonar for bathymetric exploration, Radar for keeping track of other vessels, or any other water-related environment surveying sensors. Additionally, algorithms such as bathymetric SLAM and the Fusion algorithm could be added.

Moreover, implementing the Fusion algorithm could serve as a base for AI obstacle detection and avoidance, especially if using the Lidar and Image Feed sensors. It would allow for greater precision and improved confidence that the USV will not crash. Additionally, we want to draw attention to the 3D SLAM, as a natural progression from the currently supported 2D SLAM. Lastly, future improvement should also account for the need for inertial correction to help the boat move more accurately, and the need to increase the efficiency of sensors.

7.2 Improvements Outside the Current Scope

Apart from the in-scope improvements, our project and the related field could benefit from additional features. The most important one is the physical implementation of the USV and a more accurate URDF model. Although this would require some research regarding the design and the characteristics of the vessel, it would yield immense benefits such as improved simulation realism and more accurate algorithms.

Although we focused on developing a framework and a USV solely for the canal environment, we see a vast area of opportunity to expand towards the support of other environments. Including rivers and seas would allow us to design and develop a single solution spanning various environments with the boat navigating through them freely. To achieve this, we would need to include models such as faster currents, irregular terrains without well-defined walls, higher waves, stronger winds, and lower visibility due to rain or fog. Also, we would need to find a way to include all these within the same environment while keeping them distinct areas.

8 Conclusion

Altogether, our work introduces a framework which supports the USV development process by addressing four different parts of the solution with four corresponding components. Engineers are empowered to create their USVs using existing modules; test and observe the behaviour of their implementation in real-time within the Simulator; process and store data on the server for later use; and communicate with users through the web platform. We have removed the issues arising from the initial project setup and abridged the communication and functionality of the individual segments. Overall, we believe that our framework provides all the components needed to reach a domain-specific solution quickly and efficiently.

Regardless of the next steps, we have produced a project which addresses both the initial requirements and the underlying need of the client. Through our research, we have realised the true scope of the domain and the different possible applications, and are glad to provide a tool which could help both newcomers and experts alike. We proved this by implementing a USV system, and are eager to see how our contributions will be used in future work. With the architecture design, implementation, and modularity, we firmly believe that the MayFlower Framework can continue to grow and provide a foundation for future development in the area of USV research and engineering.

References

Josep Aulinas, Yvan Petillot, Joaquim Salvi, and Xavier Lladó. 2008. The SLAM Problem: A Survey. In *Proceedings of the 2008 Conference on Artificial Intelligence Research and Development: Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*. IOS Press, NLD, 363–371.

Apollo Auto. 2020. ApolloAuto/apollo. https://github.com/ApolloAuto/apollo original-date: 2017-07-04T19:03:31Z.

Automotive. 2018. The Disrupters: Some LiDARs could cause blindness, warns Aeye – TU Automotive. https://www.tu-auto.com/the-disrupters-some-lidars-could-cause-blindness-warns-aeye-2/

T. Bailey and H. Durrant-Whyte. 2006. Simultaneous localization and mapping (SLAM): part II. *IEEE Robotics & Automation Magazine* 13, 3 (Sept. 2006), 108–117. https://doi.org/10.1109/MRA.2006.1678144

Prateek Bansal, Kara M. Kockelman, and Amit Singh. 2016. Assessing public opinions of and interest in new vehicle technologies: An Austin perspective. *Transportation Research Part C: Emerging Technologies* 67 (June 2016), 1–14. https://doi.org/10.1016/j.trc.2016.01.019

Martin Bischoff. 2020. siemens/ros-sharp. https://github.com/siemens/ros-sharp original-date: 2017-09-16T18:27:26Z.

Mary Bowers. 2008. Google's Street View raises concerns. *The Guardian* (April 2008). https://www.theguardian.com/technology/2008/apr/10/news.google

Electronic Privacy Information Center. 2012. Investigations of Google Street View. https://epic.org/privacy/streetview/

European Commission. 2017. Autonomous cars – the future of the automotive industry. https://ec.europa.eu/growth/tools-databases/dem/monitor/content/autonomous-cars-%E2%80%93-future-automotive-industry

Davison. 2003. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings Ninth IEEE International Conference on Computer Vision*. IEEE, Nice, France, 1403–1410 vol.2. https://doi.org/10.1109/ICCV.2003.1238654

DerSpiegel. 2010. Low Visibility Ahead?: More than 100,000 Germans Ask Google to Blur their Homes. https://www.spiegel.de/international/germany/low-visibility-ahead-more-than-100-000-germans-ask-google-to-blur-their-homes-a-718374.html

DjangoSoftwareFoundation. 2020. The Django Web framework. https://www.djangoproject.com/

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. 1–16.

H. Durrant-Whyte and T. Bailey. 2006. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine* 13, 2 (June 2006), 99–110. https://doi.org/10.1109/MRA.2006.1638022

EncodeOSSLtd. 2020. Django REST framework. https://www.django-rest-framework.org/

Facebook. 2020. React – A JavaScript library for building user interfaces. https://reactjs.org/

Jorge Fraga, João Sousa, Gonçalo Cabrita, Paulo Coimbra, and Lino Marques. 2014. Squirtle: An ASV for Inland Water Environmental Monitoring. In *ROBOT2013: First Iberian Robotics Conference*, Manuel A. Armada, Alberto Sanfeliu, and Manuel Ferre (Eds.). Vol. 252. Springer International Pub., Cham, 33–39. https://doi.org/10.1007/978-3-319-03413-3_3

Oren Gal and Eran Zeitouni. 2013. Tracking Objects Using PHD Filter for USV Autonomous Capabilities. In *Robotic Sailing 2012*, Colin Sauzé and James Finnis (Eds.). Springer, Berlin, Heidelberg, 3–12. https://doi.org/10.1007/978-3-642-33084-1_1

Hongbo Gao, Bo Cheng, Jianqiang Wang, Keqiang Li, Jianhui Zhao, and Deyi Li. 2018. Object Classification Using CNN-Based Fusion of Vision and LIDAR in Autonomous Vehicle Environment. *IEEE Transactions on Industrial Informatics* 14, 9 (Sept. 2018), 4224–4231. https://doi.org/10.1109/TII.2018.2822828

Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. 2007. Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. *IEEE Trans. on Robotics* 23, 1 (Feb. 2007), 34–46. https://doi.org/10.1109/TRO.2006.889486

Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. 2009. OpenSLAM.org. https://openslam-org.github.io/gmapping

G. Grisettiyz, C. Stachniss, and W. Burgard. 2005. Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, Barcelona, Spain, 2432–2437. https://doi.org/10.1109/ROBOT.2005.1570477

Jungwook Han, Yonghoon Cho, and Jinwhan Kim. 2019. Coastal SLAM With Marine Radar for USV Operation in GPS-Restricted Situations. *IEEE Journal of Oceanic Eng.* 44, 2 (April 2019), 300–309. https://doi.org/10.1109/JOE.2018.2883887

Jeff Hecht. 2019. Safety questions raised about 1550 nm lidar. https://www.laserfocusworld.com/blogs/article/14040682/safety-questions-raised-about-1550-nm-lidar

Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. 2016. Real-Time Loop Closure in 2D LIDAR SLAM. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 1271–1278.

A. Hussein, F. García, and C. Olaverri-Monreal. 2018. ROS and Unity Based Framework for Intelligent Vehicles Control and Simulation. In *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*. 1–6.

Minju Kang, Sungchur Kwon, Jeonghong Park, Taeyun Kim, Jungwook Han, Jeonghyeon Wang, Seonghun Hong, Yeonjoo Shim, Sukmin Yoon, Byunghyun Yoo, and Jinwhan Kim. 2015. Development of USV Autonomy for the 2014 Maritime RobotX Challenge. *IFAC-PapersOnLine* 48, 16 (2015), 13–18. https://doi.org/10.1016/j.ifacol.2015.10.251

Shinpei Kato, Eijiro Takeuchi, Yoshio Ishiguro, Yoshiki Ninomiya, Kazuya Takeda, and Tsuyoshi Hamada. 2015. An Open Approach to Autonomous Vehicles. *IEEE Micro* 35, 6 (Nov. 2015), 60–68. https://doi.org/10.1109/MM.2015.133

S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi. 2018. Autoware on Board: Enabling Autonomous Vehicles with Embedded Systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*. IEEE, Porto, 287–296. https://doi.org/10.1109/ICCPS.2018.00035

Taeyun Kim and Jinwhan Kim. 2017. Panel-based bathymetric SLAM with a multibeam echosounder. In *2017 IEEE Underwater Technology (UT)*. IEEE, Busan, South Korea, 1–5. https://doi.org/10.1109/UT.2017.7890321

Zoe Kleinman. 2019. Driverless car laser ruined camera. *BBC News* (Jan. 2019). www.bbc.co.uk/news/technology-46875947

Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. 2018. Sensors and Sensor Fusion in Autonomous Vehicles. In *2018 26th Telecommunications Forum (TELFOR)*. 420–425. https://doi.org/10.1109/TELFOR.2018.8612054

Stefan Kohlbrecher, Oskar von Stryk, Johannes Meyer, and Uwe Klingauf. 2011. A flexible and scalable SLAM system with full 3D motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. IEEE, Kyoto, Japan, 155–160. https://doi.org/10.1109/SSRR.2011.6106777

G. Kohler, G. Gasas, M. Lüdtke, B. Lytle, and A. Jeandeau. 2019. roslibpy. https://roslibpy.readthedocs.io/en/latest/

LG Silicon Valley Lab. 2020. lgsvl/simulator. https://github.com/lgsvl/simulator original-date: 2018-07-02T22:07:26Z.

Sandra Laville. 2020. Environment Agency chief supports plan to weaken river pollution rules. www.theguardian.com/environment/2020/aug/19/environment-agency-chief-backs-plan-to-water-down-river-cleanliness-rules-james-bevan

Zhixiang Liu, Youmin Zhang, Xiang Yu, and Chi Yuan. 2016. Unmanned surface vehicles: An overview of developments and challenges. *Annual Reviews in Control* 41 (05 2016). https://doi.org/10.1016/j.arcontrol.2016.04.018

Patrick Mania and Michael Beetz. 2019. A Framework for Self-Training Perceptual Agents in Simulated Photorealistic Environments. In *International Conference on Robotics and Automation (ICRA)*. Montreal, Canada.

Andre McGrail. 2020. Verasl/BoatAttack. https://github.com/Verasl/BoatAttack original-date: 2018-02-12T17:54:06Z.

George Monbiot. 2020. The government is looking the other way while Britain's rivers die before our eyes. http://www.theguardian.com/commentisfree/2020/aug/12/government-britains-rivers-uk-waterways-farming-water-companies

BBC NEWS. 2009. All clear for Google Street View. (April 2009). http://news.bbc.co.uk/1/hi/technology/8014178.stm

On-Road Automated Driving (ORAD) committee. 2018. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Technical Report. SAE International. https://doi.org/10.4271/J3016_201806

PyTorch Team. 2020. PyTorch. https://www.pytorch.org

Francisca Rosique, Pedro J. Navarro, Carlos Fernández, and Antonio Padilla. 2019. A Systematic Review of Perception System and Simulators for Autonomous Vehicles Research. *Sensors* 19, 3 (Feb. 2019), 648. https://doi.org/10.3390/s19030648

Chris Salzberg. 2008. Japan: Debate over Google Street View continues · Global Voices. https://globalvoices.org/2008/08/14/japan-debate-over-google-street-view-continues/

S. A. Scherer, D. Dube, and A. Zell. 2012. Using depth in visual simultaneous localisation and mapping. In *2012 IEEE Int. Conf. on Robotics and Automation*. IEEE, St Paul, MN, USA, 5216–5221. https://doi.org/10.1109/ICRA.2012.6224864

Hans-Peter Schöner. 2018. Simulation in development and testing of autonomous vehicles. In *18. Internationales Stuttgarter Symposium*, Michael Bargende, Hans-Christian Reuss, and Jochen Wiedemann (Eds.). Springer Fachmedien Wiesbaden, Wiesbaden, 1083–1095. https://doi.org/10.1007/978-3-658-21194-3_82 Series Title: Proceedings.

Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. 2017. AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles. In *Field and Service Robotics*. arXiv:arXiv:1705.05065 https://arxiv.org/abs/1705.05065

A. Sinisterra, M. Dhanak, and N. Kouvaras. 2017. A USV platform for surface autonomy. In *OCEANS 2017 - Anchorage*. 1–8.

D. Skvorc, M. Horvat, and S. Srbljic. 2014. Performance evaluation of Websocket protocol for implementation of full-duplex web streams. In *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. IEEE, Opatija, Croatia, 1003–1008. https://doi.org/10.1109/MIPRO.2014.6859715

TensorFlow Team. 2020. TensorFlow. https://www.tensorflow.org/

S. Thrun and J. J. Leonard. 2008. Simultaneous Localization and Mapping. In *Springer Handbook of Robotics*, B. Siciliano and O. Khatib (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 871–889. https://doi.org/10.1007/978-3-540-30301-5_38

Jaycil Z Varghese. 2015. Overview of Autonomous Vehicle Sensors and Systems. (2015), 14.

Ihn-Sik Weon, Soon-Geul Lee, and Jae-Kwan Ryu. 2017. Virtual bubble filtering based on heading angle and velocity for unmanned surface vehicle (USV). In *2017 17th International Conference on Control, Automation and Systems (ICCAS)*. IEEE, Jeju, 1954–1958. https://doi.org/10.23919/ICCAS.2017.8204276

Brian Williams, Paul Smith, and Ian Reid. 2007. Automatic Relocalisation for a Single-Camera Simultaneous Localisation and Mapping System. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, Rome, Italy, 2784–2790. https://doi.org/10.1109/ROBOT.2007.363893 ISSN: 1050-4729.

F. Zhao, H. Jiang, and Z. Liu. 2019. Recent development of automotive LiDAR technology, industry and trends. In *Eleventh Intnl. Conf. on Digital Image Processing (ICDIP 2019)*, X. Jiang and J.-N. Hwang (Eds.). SPIE, Guangzhou, China, 178. https://doi.org/10.1117/12.2540277

A Access to Source Code

The overall MayFlower Framework project can be found in its repository on Github - https://github.com/alexandar1000/MayFlower. This repository encompasses all four components and serves as a way of keeping the separate units together. Nonetheless, individual components are connected to it only as git submodules, and reside in their own repositories:

- Simulator - https://github.com/alexandar1000/MayFlower-Simulator
- USV System - https://github.com/alexandar1000/MayFlower-RobotSystem
- Server - https://github.com/alexandar1000/MayFlower-Server
- Web Platform - https://github.com/alexandar1000/MayFlower-WebPortal

In the case that the repository is not accessible, please contact via GitHub one of the maintainers to gain access:

- `@AtonalStar`
- `@aunroel`
- `@dennis9707`
- `@Najtmb`
- `@sbs61`
- `@alexandar1000`

B Installation Instructions

As the structure of the project is quite complex and possible to change, please refer to the corresponding repository's Readme and Wiki pages for the most up-to-date instructions. The instructions for installing and running the entire Framework at once using Docker can be found on the main repository, but you will be required to install additional tools. Apart from this, it is possible to install only specific components, and for this please refer to their own repository documentation.

C Look of the Rendered Environment

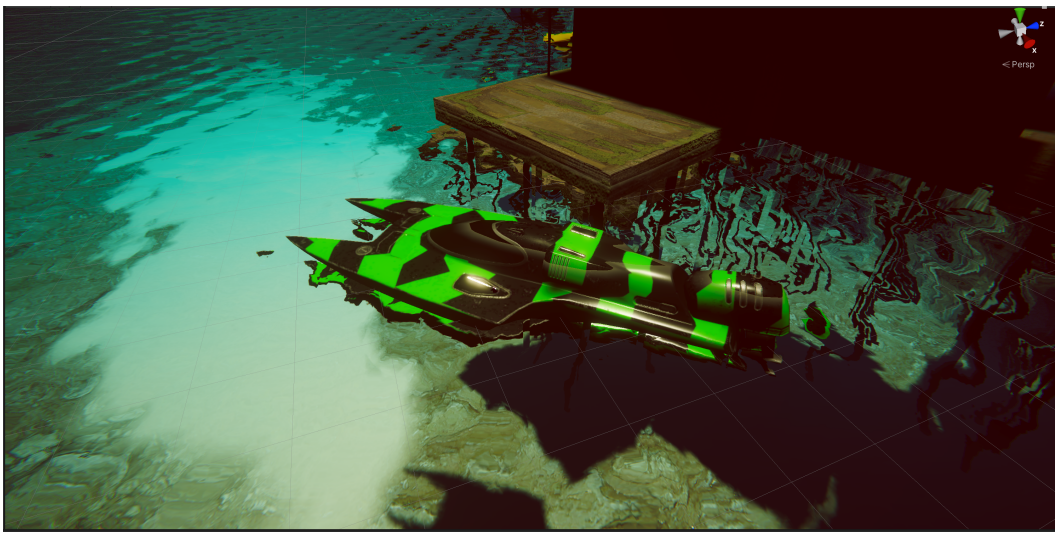

Fig. 3. The Boat Model used within in the project for development purposes

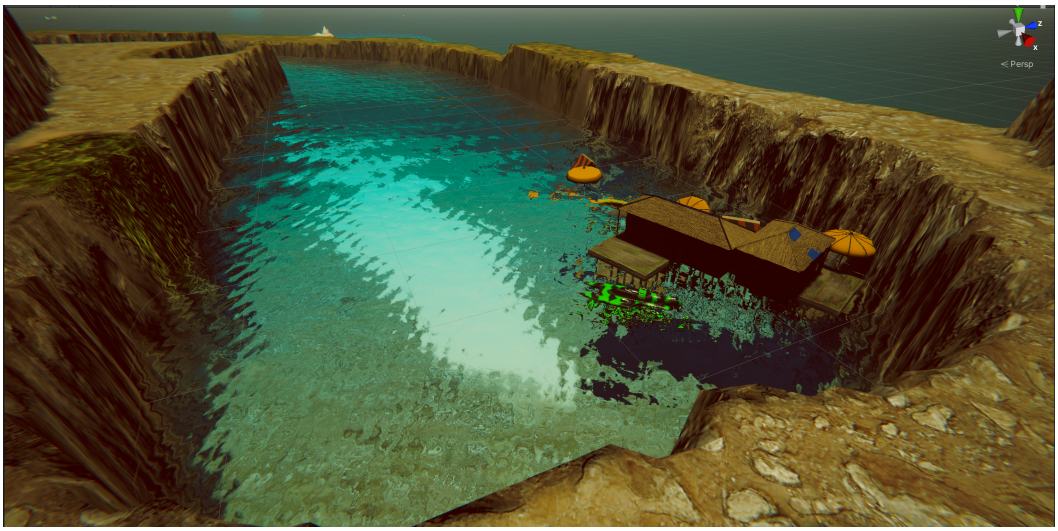

Fig. 4. The closeup image of the boat port within the Simulator

Fig. 5. The boat port within the Simulator

Fig. 6. The areal view of the canal within the Simulator

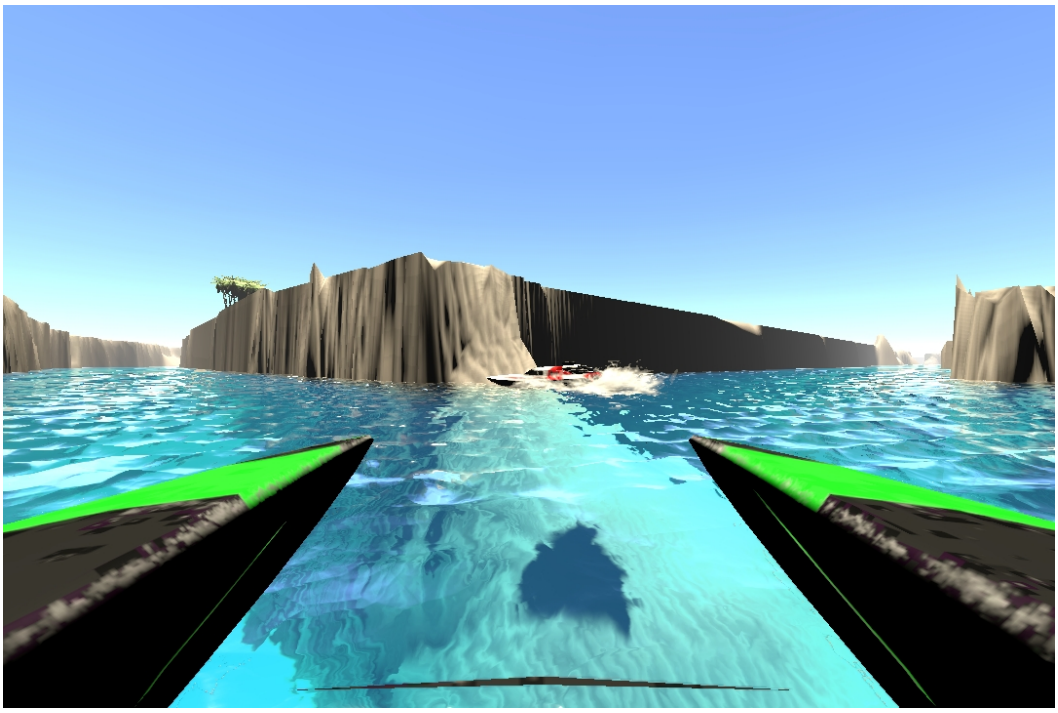

Fig. 7. Video feed image received by the Server at the start of the simulation

D SLAM & TF

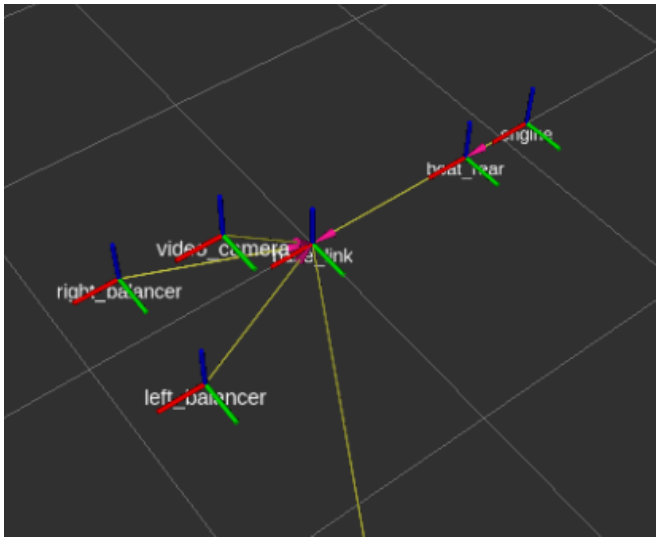

Fig. 8. URDF representation of the simulated boat

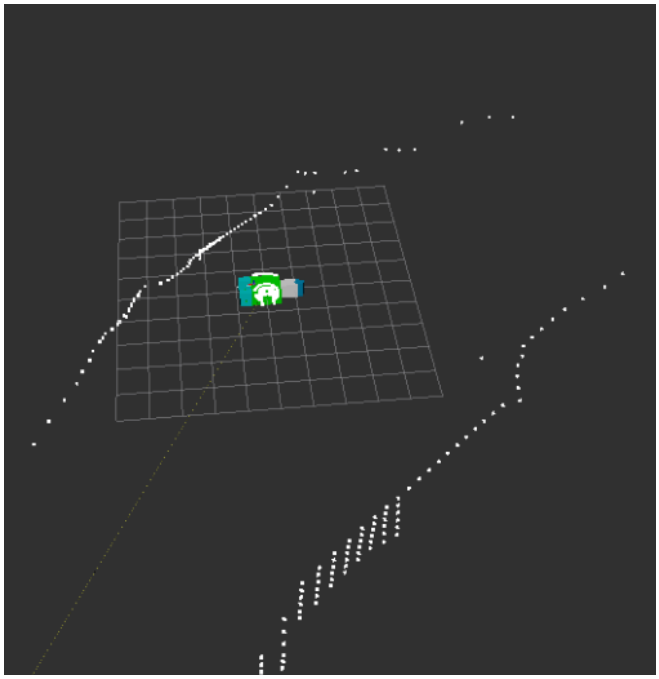

Fig. 9. Localisation of the boat using 3D Lidar data in RViz

E Web Platform

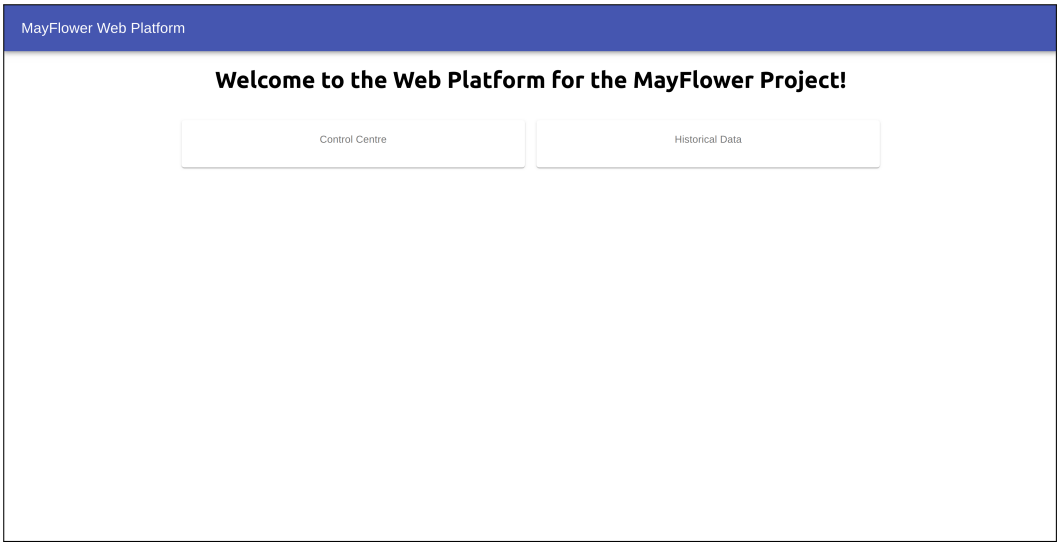

Fig. 10. The Web Platform Landing Page

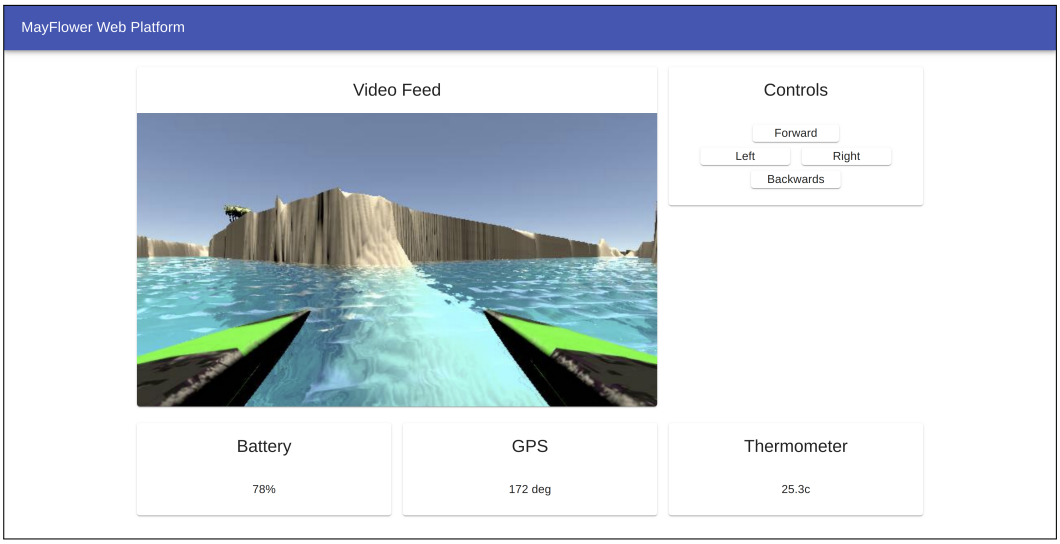

Fig. 11. The Web Platform Control Centre

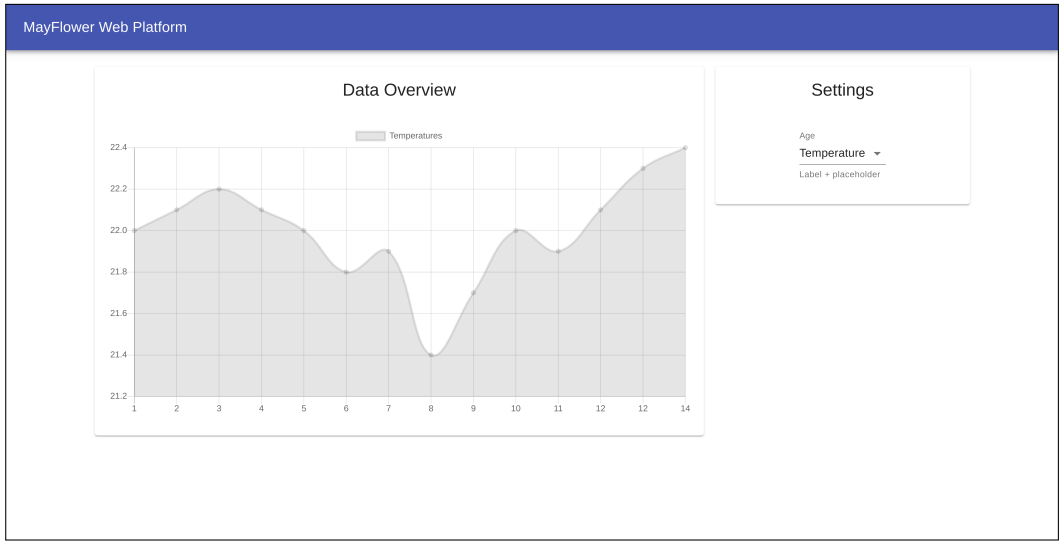

Fig. 12. The Web Platform Historical Data Overview

Declarations

This report is submitted as part requirement for the MSc Software Systems Engineering degree at UCL. It is substantially the result of our group's own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.