COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER EN BIG DATA Y ANALÍTICA AVANZADA

TRABAJO FIN DE MÁSTER

ENTORNO DE SIMULACIÓN DE REINFORCEMENT LEARNING PARA POSICIONAMIENTO DINÁMICO

Autor: Alejandro del Rosal Carmona

Director: Roberto Gómez-Espinosa Martín

Madrid

[illegible]

Fdo.: Alejandro del Rosal Carmona Fecha: 27/ 06/ 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Roberto Gómez-Espinosa Martín Fecha: 27/ 06/ 2023

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER EN BIG DATA Y ANALÍTICA AVANZADA

TRABAJO FIN DE MÁSTER

ENTORNO DE SIMULACIÓN DE REINFORCEMENT LEARNING PARA POSICIONAMIENTO DINÁMICO

Autor: Alejandro del Rosal Carmona

Director: Roberto Gómez-Espinosa Martín

Madrid

Agradecimientos

Me gustaría expresar mi agradecimiento en primer lugar a HI-IBERIA por la confianza que han depositado en mí y por brindarme la oportunidad de trabajar en un proyecto de alto nivel como ePROA. También quiero agradecer a todos mis compañeros de trabajo, quienes con su profesionalismo y conocimientos en el campo han creado un entorno laboral que, gracias a sus altos estándares, me ha motivado a crecer tanto a nivel profesional como personal. En particular, me gustaría agradecer a Beatriz Salcedo por acompañarme durante el proceso de selección y por su atención durante todo este tiempo en la empresa, a Roberto Gómez por ser mi tutor y a José María Moreu por su paciencia y por enseñarme mucho con su amplia experiencia. Por último, quiero agradecer a la Universidad Pontificia Comillas, ICAI, y a Carlos Morras por la oportunidad de realizar mis prácticas y por todo lo que he aprendido durante este año.

Acknowledgements

First, I would like to express my gratitude to HI-IBERIA for the trust they have placed in me and for giving me the opportunity to work on a high-level project like ePROA. I would also like to thank all my colleagues who, with their professionalism and expertise, have created a work environment that, thanks to their ambitious standards, has pushed me to grow both professionally and personally. I would like to extend a special thanks to Beatriz Salcedo for going with me throughout the selection process and for her attentive care during all this time at the company, to Roberto Gómez as my mentor, and to José María Moreu for his patience and for imparting his extensive experience. Finally, I would like to express my gratitude to the Universidad Pontificia Comillas, ICAI, and Carlos Morras for giving me the opportunity to undertake this internship and for all that I have learned during this year.

ENTORNO DE SIMULACIÓN DE REINFORCEMENT LEARNING PARA POSICIONAMIENTO DINÁMICO

Autor: del Rosal Carmona, Alejandro.
Director: Gómez-Espinosa Martín, Roberto.
Entidad Colaboradora: HI-IBERIA

RESUMEN DEL PROYECTO

El objetivo final del proyecto es la aplicación de técnicas de IA (Inteligencia Artificial) para el control del posicionamiento dinámico de un nuevo buque de mantenimiento de un parque de eólica flotante. El foco de este trabajo será principalmente el montaje y configuración de un entorno de simulación apropiado para el entrenamiento de agentes de IA basados en aprendizaje por refuerzo. Dichos agentes se encargarán del control del posicionamiento dinámico del buque mediante el control de los propulsores de la embarcación en función al estado del entorno.

Palabras clave: DP, Python, Reinforcement Learning, Control, Environment, Framework

1. Introducción

El posicionamiento dinámico (DP) es un sistema controlado por ordenador utilizado para mantener automáticamente la posición y el rumbo de una embarcación utilizando sus propias hélices y propulsores. En específico, los sistemas DP se utilizan para controlas los tres grados de libertad el buque, correspondientes al control de navegación (seakeeping): el balanceo, el avance (surge), desplazamiento lateral (sway) y giro en torno al eje vertical (yaw). [MEHR20]

Al ser una tecnología involucrada en misiones en alta mar, operaciones de rescate y muchas otras aplicaciones críticas [ERRV], es, y ha sido desde su primera utilización en la década de 1960 objeto de numerosos estudios y desarrollos en términos de técnicas de control, con el fin de lograr una mayor precisión y reducir el movimiento del barco.

El enfoque más convencional a estos sistemas es el empleo de controladores PID (Proporcional Integral Derivativo) que mediante funciones de transferencia ajustan los propulsores basándose en la retroalimentación del error entre la posición actual y la deseada. Con el tiempo estos modelos se han perfeccionado y han evolucionado para incorporar modelos matemáticos muy próximos a la realidad de las embarcaciones y las condiciones medioambientales. Estos modelos ayudan al sistema a predecir la respuesta para un rango de condiciones más variado.

Más recientemente, hay un interés creciente en la aplicación de aprendizaje automático, específicamente de aprendizaje por refuerzo a los sistemas de DP [OVER21]. En aprendizaje por refuerzo un agente aprende a tomar decisiones a base de tomar acciones en un entorno y recibir una recompensa basada en el resultado de esas acciones. Este enfoque podía potencialmente lleva a sistemas de DP más adaptables y eficientes, y es el origen de este proyecto, en específico el desarrollo de dicho framework de RL (Reinforcement Learning) para el entrenamiento de agentes de IA.

2. Definición del proyecto

El problema que plantea afrontar este proyecto es la falta de ese entono para el entrenamiento de los agentes, en especial de un entorno open-source, realista y representativo para aplicaciones de RL en el contexto de DP. Los objetivos son el desarrollo de un entorno simulado de entrenamiento que cumpla los siguientes criterios:

- **Open Source:** el entorno estará disponible abiertamente para su uso modificación y distribución. Este aspecto es crucial para acelerar la investigación y desarrollo de esta área.

- **Actualizado y realista:** el entorno reflejará con precisión los últimos avances en tecnologías de aprendizaje por refuerzo y ofrecerá ajustes de entrenamiento considerando observaciones realistas de un entorno real como son el viento, las olas, así como características propias del buque.

- **Representativo y configurable:** el entorno podrá representar muchos escenarios ofreciendo la posibilidad de configurar aspectos de la simulación ofreciendo una herramienta versátil para investigación.

- **Compatible con Gym y API (Apliccation Programming Interface) de Python:** el desarrollo del entorno se realizará con el objetivo de controlarse a través de Python dado que es el lenguaje más estandarizado y de mayor integración para el aprendizaje por refuerzo, en especial con el paquete de Gym de OpenAI el cual será base de nuestro entorno.

3. Descripción del sistema

El sistema considerado se ha construido sobre el framework de simulación proporcionado por la competición Virtual RobotX (VRX) [VRXC23], un evento internacional a nivel de universidad que se centra en el desarrollo de sistemas marítimos autónomos.

La plataforma de VRX es un entorno simulado en Gazebo, un simulador de robótica 3D open-source, con el Sistema Operativo de Robots (ROS2) para la comunicación y programación de robots. Permite la experimentación de diversos comportamientos robóticos marítimos, como el mantenimiento de la posición, la navegación por puntos de referencia y la evasión de obstáculos entre otros, empleando como vehículo el WAM-V [WAMV] una embarcación estándar en experimentación marítima no tripulada.

Ilustración 1: Comparación entorno real vs simulado VRX

VRX aparte de por ser open-source y contar con fácil interacción con Python, fue elegida pilar del proyecto también por contar con el patrocinio y la orientación de varias entidades destacadas en las industrias de la robótica y la marítima, incluyendo Open Robotics, la Oficina de Investigación Naval (ONR) y la Escuela de Posgrado Naval (NPS) entre otros.

La infraestructura de ROS2 actúa como la columna vertebral del sistema, proporcionando las herramientas necesarias para la programación y control de los comportamientos del robot en la simulación. Permite una comunicación eficiente entre los nodos de software a través de una arquitectura de publicación y suscripción mediante tópicos y peticiones y respuestas mediante servicios lo que facilita la integración de varios componentes y funcionalidades del sistema robótico.

Gazebo, por otro lado, proporciona el entorno virtual en 3D en el que los robots operan. Se integra sin problemas con ROS2, permitiendo que los robots simulados reciban comandos e informen datos sensoriales al sistema de ROS2. El motor de física de Gazebo simula las interacciones del robot con su entorno, proporciona una representación precisa del comportamiento de un robot en un entorno marítimo. Además, ROS2 y Gazebo juntos facilitan la visualización y el análisis de los comportamientos del robot en tiempo real mediante la GUI de Gazebo.

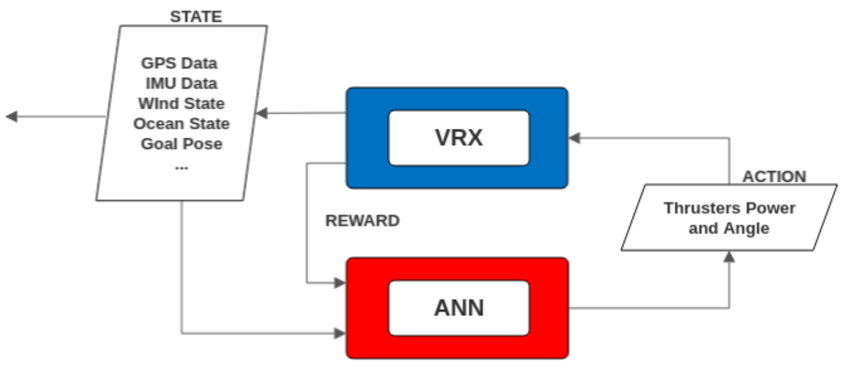


Ilustración 2: Esquema de alto nivel del ciclo de aprendizaje por refuerzo

Sobre esta plataforma, se desarrollará el código en Python requerido para ser capaz de actuar como entorno de aprendizaje por refuerzo y se realizarán ligeras modificaciones en su código base en C++. El objetivo principal será desarrollar un entorno para un entrenamiento básico de agentes según una condición de la mar fija y un viento variable, con la posibilidad a futuro de modificar estas condiciones de oleaje fijas.

Lo que buscamos es realizar una primera demostración preliminar de la capacidad del aprendizaje por refuerzo y de nuestro entorno de resolver condiciones marítimas de alto nivel, para a futuro y desarrollo realizar un entrenamiento más extenso y elaborado para adaptarla a cualquier combinación de condiciones meta oceánicas.

4. Resultados

Se han logrado los siguientes objetivos:

- **Open Source**: el entorno final se ha desarrollado empleando únicamente tecnologías open-source disponibles al público.
- **Actualizado y realista**: al estar basado en la competición VRX, celebrada año a año y con una creciente atención en el sector, el simulador cuenta con una comunidad con la cual interactuar y brindar soporte. A parte se ha conseguido interaccionar sin problema con los sensores propios de un DP en una embarcación como pueden ser el GPS o la unidad inercial (IMU), modificar las condiciones meta oceánicas (viento y oleaje) y recibir lecturas de estas y controlar según convenga los propulsores del vehículo.
- **Representativo y configurable:** al contar de base con la plataforma de VRX, nos asegura una representación fidedigna y contrastada de la realidad contando con modelado del oleaje basado en el espectro de Pierson-Moskowitz o modelado del vehículo siguiendo los seis grados de libertad de Fossen [BBIN19]. Dicha representación permite tanto una sencilla configuración de las condiciones ambientales deseadas como de la propia geometría y características físicas del vehículo.
- **Compatible con Gym y API de Python:** unas sencillas modificaciones sobre el código base han habilitado el control a través de Python y el servicio de /world_control de las funciones clave de un entorno Gym en especial del pausado, reanudado y avance por ‘steps’ o pasos de simulación de duración fija. El resto del código para la ejecución de un episodio se ha desarrollado en Python.

Aparte, se ha logrado la ejecución en remoto del sistema dockerizado y con aceleración por GPU (Graphical Processing Unit) en Linux OS con un entrenamiento de forma centralizada obteniendo velocidades de simulación de 3 veces el tiempo real con solo un 30% de uso de GPU. El usuario especificaría las condiciones de entrenamiento que desea y los parámetros necesarios deseados a través de un archivo de configuración y se levantarían múltiples instancias de entrenamiento cada una con su propio agente basadas en nuestro sistema. El esquema final de nuestro sistema ha seguido la siguiente estructura:

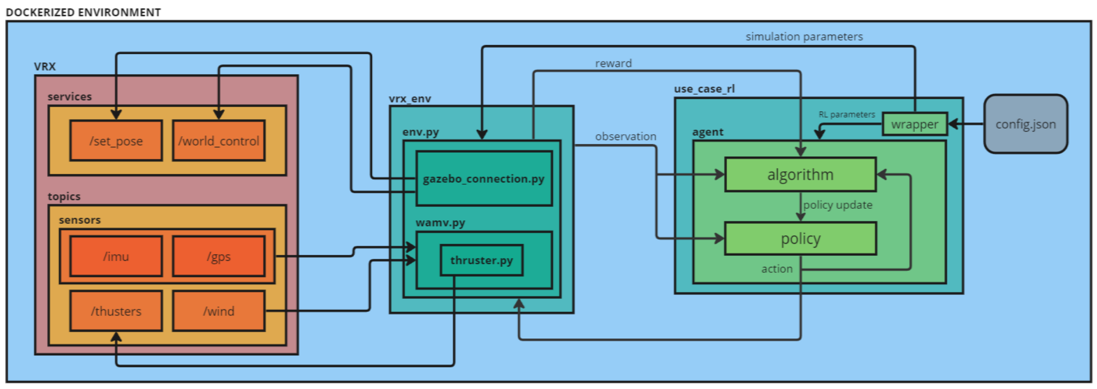


Ilustración 3: Esquema sistema final

5. Conclusiones

La necesidad de entornos simulados realistas y open-source para el entrenamiento de agentes de aprendizaje por refuerzo es cada vez más evidente. Estos entornos permiten la experimentación y optimización de algoritmos en condiciones seguras y reproducibles, aportando un gran valor a la investigación y desarrollo en este campo.

Nuestro sistema, aunque presenta desafíos como la necesidad de reinicio de Gazebo para efectuar cambios en el entorno marítimo, es una solución efectiva para entrenar agentes en condiciones de mar predefinidas. Aun con estos desafíos, esta limitación puede abordarse siguiendo el ejemplo de desarrolladores de la comunidad que han logrado la actualización dinámica del entorno sin interrupción de la simulación.

En cuanto a los objetivos de nuestro sistema, reitero hemos logrado con éxito proporcionar un entorno de entrenamiento para condiciones de mar fijas y predefinidas. Además, se ha explorado la capacidad de introducir nuevos vehículos en la simulación, ampliando así la capacidad del simulador. Por último, se ha explorado la posibilidad de mejorar aún más la eficiencia, restringiendo la complejidad de los mundos simulados.

En resumen, a pesar de algunos desafíos nuestro sistema aspira a ser un recurso valioso para el entrenamiento de agentes por refuerzo en la robótica marítima y confiamos en su contribución al avance en tecnologías de posicionamiento dinámico y seguridad de operaciones marítimas.

6. Referencias

[MEHR20] Mehrzadi, Mojtaba, Yacine Terriche, Chun-Lien Su, Muzaidi Bin Othman, Juan C. Vasquez, and Josep M. Guerrero. 2020. "Review of Dynamic Positioning Control in Maritime Microgrid Systems" Energies 13, no. 12: 3188. https://doi.org/10.3390/en13123188

[ERRV] "ERV - Emergency Response and Rescue Vessel." TAIS Shipyards. Accessed June 21, 2023. https://www.taisshipyards.com/en/erv-emergency-response-and-rescue-vessel

[OVER21] Øvereng, Simen Sem, Dong Trong Nguyen, and Geir Hamre. "Dynamic Positioning using Deep Reinforcement Learning." Ocean Engineering 235 (2021): 109433. https://doi.org/10.1016/j.oceaneng.2021.109433

[VRXC23] "VRX Competition 2023." RobotX, June 20, 2023. https://robotx.org/programs/vrx-2023/.

[WAMV] "The WAM-V Remote Explorer a.k.a. Rex." MIT Marine Autonomy Lab : Robot - WAMV browse. Accessed June 21, 2023. https://oceanai.mit.edu/pavlab/pmwiki/pmwiki.php?n=Robot.WAMV

[BBIN19] B. Bingham et al., "Toward Maritime Robotic Simulation in Gazebo," OCEANS 2019 MTS/IEEE SEATTLE, Seattle, WA, USA, 2019, pp. 1-10, doi: https://doi.org/10.23919/OCEANS40490.2019.8962724

REINFORCEMENT LEARNING SIMULATION ENVIRONMENT FOR DYNAMIC POSITIONING

Author: del Rosal Carmona, Alejandro.
Supervisor: Gómez-Espinosa Martín, Roberto
Collaborating Entity: HI-IBERIA

ABSTRACT

The goal of the project is the application of AI techniques for the dynamic positioning control of a new maintenance vessel for a floating wind farm. The focus of this work will be primarily the assembly and configuration of a proper simulation environment for training reinforcement learning-based AI agents. These agents will oversee the dynamic positioning control of the vessel by controlling the vessel's thrusters according to the state of the environment.

Keywords: Dynamic Positioning, Python, Reinforcement Learning, Environment.

1. Introduction

Dynamic Positioning (DP) is a computer-controlled system used to automatically keep the position and heading of a vessel using its own propellers and thrusters. Specifically, DP systems are used to control the three degrees of freedom of the vessel, corresponding to seakeeping control: surge, sway, and yaw [MEHR20].

Since it is a technology involved in offshore missions, rescue operations, and many other critical applications [ERRV], it is, and has been since its first use in the 1960s, the subject of many studies and developments in terms of control techniques, to achieve greater precision and reduce ship movement.

The most conventional approach to these systems is the use of PID controllers that adjust the thrusters based on the feedback of the error between the current and desired position through transfer functions. Over time, these models have been refined and have evolved to incorporate mathematical models close to the reality of the vessels and environmental conditions. These models help the system predict the response for a wider range of conditions.

More recently, there is growing interest in applying machine learning, specifically reinforcement learning, to DP systems [OVER21]. In reinforcement learning, an agent learns to make decisions based on taking actions in an environment and receiving a reward based on the outcome of those actions. This approach could potentially lead to more adaptable and efficient DP systems, and is the origin of this project, specifically the development of such a simulated training environment.

2. Project definition

The problem this project aims to address is the lack of an environment for training agents, particularly an open-source, realistic, and representative environment for RL applications in the context of DP. The goals are to develop a simulated training environment that meets the following criteria:

- **Open Source:** The environment will be openly available for use, modification, and distribution. This aspect is crucial to accelerate the research and development in this area.
- **Up-to-date and realistic:** The environment will accurately reflect the latest advances in reinforcement learning technologies and will offer training adjustments considering realistic observations from a real environment such as wind, waves, as well as the ship's own characteristics.
- **Representative and configurable:** The environment will be capable of standing for a wide variety of scenarios offering the possibility of configuring various aspects of the simulation, thus supplying a versatile tool for research.
- **Compatible with Gym and Python API:** The development of the environment will be carried out with the aim of being controlled through Python, as it is the most standardized language and has the greatest integration for reinforcement learning, especially with OpenAI's Gym package, which will be the basis of our environment.

3. System description

The system under consideration has been built on the simulation framework provided by the Virtual RobotX (VRX) competition [VRXC23], an international university-level event that focuses on the development of autonomous maritime systems.

The VRX platform is a simulated environment in Gazebo, an open-source 3D robotics simulator, with the Robot Operating System (ROS2) for robot communication and programming. It allows the experimentation of various maritime robotic behaviors, such as keeping position, navigation through waypoints, and obstacle avoidance among others, using the WAM-V [WAMV] as the vehicle, a standard vessel in unmanned maritime experimentation.

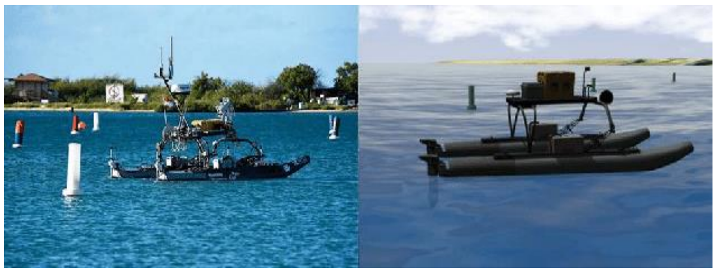


Fig. 1: Real vs Simulated environment VRX

VRX was chosen as the cornerstone of the project not only because it is open-source and has easy interaction with Python, but also because it has the sponsorship and guidance of several entities prominent in the robotics and maritime industries, including Open Robotics, the Office of Naval Research (ONR), and the Naval Postgraduate School (NPS) among others.

The ROS2 infrastructure acts as the backbone of the system, supplying the necessary tools for programming and controlling the robot's behaviors in the simulation. It allows efficient communication between software nodes through a publish-subscribe architecture using topics and request-response through services, facilitating the integration of various components and functionalities of the robotic system.

Gazebo supplies the 3D virtual environment in which the robots operate. It integrates seamlessly with ROS2, allowing the simulated robots to receive commands and report sensory data to the ROS2 system. Gazebo's physics engine simulates the robot's interactions with its environment, supplying an accurate representation of a robot's behavior in a maritime environment. Furthermore, ROS2 and Gazebo together help the visualization and analysis of robot behaviors in real time through the Gazebo GUI.

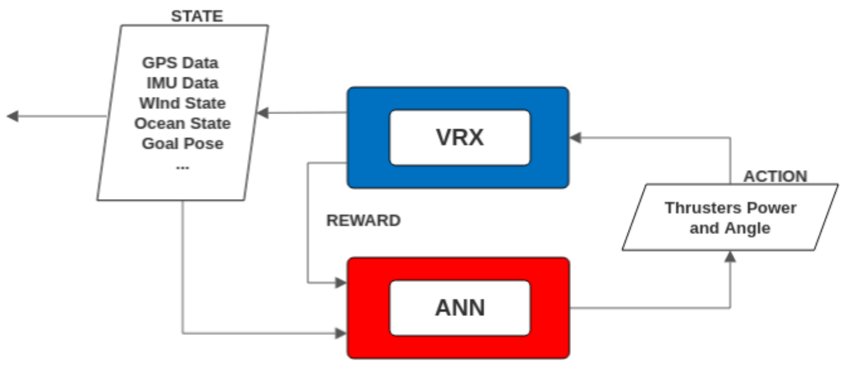


Fig. 2: High level scheme of the cycle of reinforcement learning

On this platform, the required Python code will be developed to be able to act as a reinforcement learning environment, and slight modifications will be made to its base code in C++. The main aim for now will be the development of an environment for basic training of agents based on a fixed sea condition and variable wind, with the possibility in the future of changing these fixed wave conditions.

What we are looking for with this, is to make a preliminary demonstration of the capability of reinforcement learning and our environment to solve high-level maritime conditions, so that in the future, with further development, we can conduct more extensive and elaborate training that allows full adaptability to any combination of target oceanic conditions.

4. Results

The following goals have been achieved:

- **Open Source:** the final environment has been developed using only open-source technologies available to the public.
- **Updated and realistic:** being based on the VRX competition, which is held annually and is receiving growing attention in the sector, the simulator has a community with which to interact and supply support. Moreover, it has been possible to interact without problem with the sensors typical of a DP on a vessel such as the GPS or the inertial unit (IMU), change the target oceanic conditions (wind and waves) and receive readings from these and control the vehicle's thrusters, as necessary.
- **Representative and configurable:** having the VRX platform as a base ensures a faithful and contrasted representation of reality, with wave modeling based on the Pierson-Moskowitz spectrum or vehicle modeling following Fossen's six degrees of freedom [BBIN19]. This representation allows both an easy configuration of the desired environmental conditions and of the vehicle's own geometry and physical characteristics.
- **Compatible with Gym and Python API:** a few simple modifications to the base code have enabled control through Python and the /world_control service of the key functions of a Gym environment, especially pausing, resuming, and advancing by 'steps' or fixed duration simulation steps. The rest of the code for running an episode has been developed in Python.

In addition, it has been possible to run the system remotely in a dockerized form and with GPU acceleration on Linux OS training agents in a centralized manner, achieving simulation speeds of 3 times real time with only 30% GPU usage. The user would specify the training conditions they want and the necessary desired parameters through a configuration file and multiple training instances would be launched, each with its own agent based on our system. The final scheme of our system has followed the following structure:

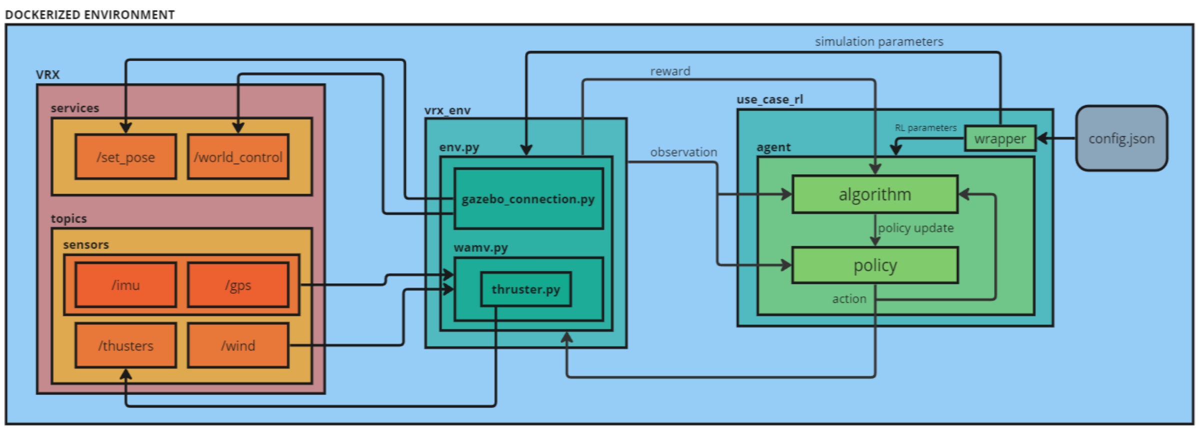


Fig. 3: Final system schema

5. Conclusions

The need for realistic and open-source simulated environments for training reinforcement learning agents is becoming increasingly clear. These environments allow for the experimentation and optimization of algorithms under safe and reproducible conditions, supplyinh excellent value to research and development in this field.

Our system, while presenting challenges such as the need for a Gazebo restart to make changes in the maritime environment, is an effective solution for training agents under predefined sea conditions. Even with these challenges, this limitation can be addressed by following the example of community developers who have achieved dynamic environment updating without simulation interruption.

Regarding the goals of our system, I reiterate that we have successfully managed to supply a training environment for fixed and predefined sea conditions. In addition, the capability to introduce new vehicles into the simulation has been explored, thus expanding the simulator's ability. Finally, we have investigated the possibility of further enhancing efficiency by restricting the complexity of the simulated worlds.

In summary, despite some challenges, our system aspires to be a valuable resource for training reinforcement agents in maritime robotics, and we trust in its contribution to advances in dynamic positioning technologies and maritime operation safety.

6. References

[MEHR20] Mehrzadi, Mojtaba, Yacine Terriche, Chun-Lien Su, Muzaidi Bin Othman, Juan C. Vasquez, and Josep M. Guerrero. 2020. "Review of Dynamic Positioning Control in Maritime Microgrid Systems" Energies 13, no. 12: 3188. https://doi.org/10.3390/en13123188

[ERRV] “ERV - Emergency Response and Rescue Vessel.” TAIS Shipyards. Accessed June 21, 2023. https://www.taisshipyards.com/en/erv-emergency-response-and-rescue-vessel

[OVER21] Øvereng, Simen Sem, Dong Trong Nguyen, and Geir Hamre. "Dynamic Positioning using Deep Reinforcement Learning." Ocean Engineering 235 (2021): 109433. https://doi.org/10.1016/j.oceaneng.2021.109433

[VRXC23] “VRX Competition 2023.” RobotX, June 20, 2023. https://robotx.org/programs/vrx-2023/.

[WAMV] “The WAM-V Remote Explorer a.k.a. Rex.” MIT Marine Autonomy Lab : Robot - WAMV browse. Accessed June 21, 2023. https://oceanai.mit.edu/pavlab/pmwiki/pmwiki.php?n=Robot.WAMV

[BBIN19] B. Bingham et al., "Toward Maritime Robotic Simulation in Gazebo," OCEANS 2019 MTS/IEEE SEATTLE, Seattle, WA, USA, 2019, pp. 1-10, doi: https://doi.org/10.23919/OCEANS40490.2019.8962724

PART I

PROJECT REPORT

Table of contents

Figure Index

Table Index

CHAPTER 1. INTRODUCTION

1.1 PROBLEM STATEMENT

Dynamic positioning (DP) is a critical aspect of maritime navigation, particularly vital in situations where precise control of the vehicle's movement is crucial. In maritime environments, such as offshore operations, rescue missions, and maintenance activities in locations such as wind farms, keeping a vessel in a stable position despite the motion of waves, wind, and currents is of utmost importance.

Figure 1: eco-SOV vessel accessing an offshore wind farm.

The movements of a maritime vessel can be described using its six degrees of freedom (6 DoF), which are further divided into maneuverability movements referring to the vessel's capability to perform maneuvers (roll, pitch, and heave), and seakeeping movements related to the horizontal motions caused by waves and currents (surge, sway, and yaw). DP refers to the computer-controlled system that automatically keeps the position and orientation of a vessel using its own propellers and thrusters, primarily focusing on keeping control over the seakeeping degrees of freedom: surge, sway, and yaw.

While maneuverability is also important in navigation, it is not the primary goal of DP. DP is primarily used in scenarios that require stability and the ability to remain static in a position with high precision. Maneuverability, on the other hand, refers more to the vessel's responsiveness and its ability to make quick and agile changes in direction.

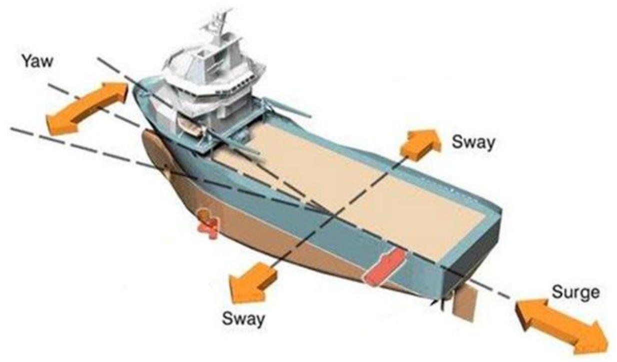


Figure 2: Horizontal Movements controlled by the DP

It is this demand of maximum accuracy what makes DP a field in constant development and evolution, looking to apply new advances in technology to reduce risks, ensure the well-being of the crew and the maximize the success of maritime operations.

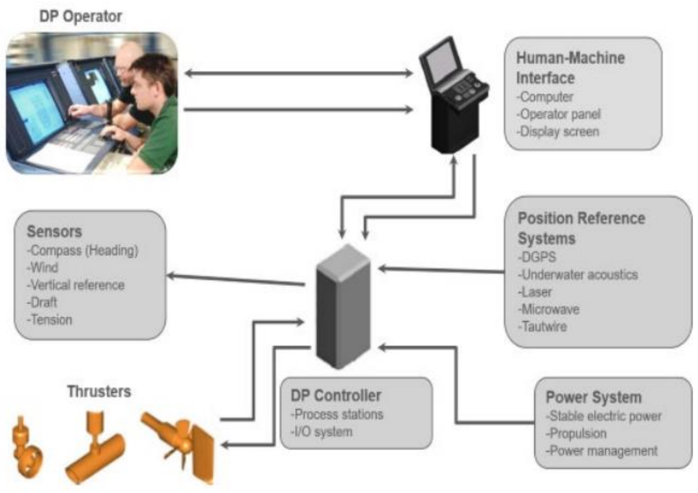


Figure 3: Summary of a conventional DP system

A conventional (DP) system consists of several key components [DPWM] that work together to achieve this precise control of a ship's position and orientation at sea. These components include:

- **Sensors:** Sensors play a crucial role in the system as they supply real-time information about the position, velocity, orientation, and environmental conditions. Commonly used sensors include GPS receivers, inertial measurement units (IMUs), wind speed sensors, lidar sensors, and cameras.
- **Control computer:** The control computer is the brain of the system. It receives data from the sensors and uses control algorithms to make real-time decisions and calculate the necessary corrections in the propellers and thrusters.
- **Propellers and thrusters**: Propellers and thrusters are the actuators used to control movement. They are continuously adjusted by the DP to generate the required force and direction to counteract the effects of waves.
- **Reference system:** The reference system is essential for DP as it supplies a reference coordinate basis for calculating the position and orientation. It can be based on global positioning systems (GPS), inertial navigation systems (INS), or other specific reference systems.

CHAPTER 2. STATE OF THE ART

DP has come a long way since it was first used in the decade of 1960s for offshore drilling operations. Today it is a fundamental technology for certain ships and operations. There is one key area that represent the state of the art in DP and it is the control system and the techniques behind it [WLQW19]:

- **Proportional Integral Derivative Algorithm:** PID controllers are the most known, enhancing the control system's adaptability and robustness. However, although it is simple and easy to use, the control on complex nonlinear systems is still poor.
- **Fuzzy Logic Control:** the essential thing is to achieve adaptive control with inaccurate mathematical models. It is often used in combination with PID to improve adaptability.
- **Model Prediction Control:** it is widely used due to its ability to deal with multivariate, non-linear features and its robustness. It is a predictive model, feedback correction and rolling optimization.
- **Artificial Intelligence:** it is a modern technology for researching and developing theories, methods, and application systems for simulating, extending, and expanding human intelligence. The most advanced techniques are based on the application of deep leaning and reinforcement learning [OVER21].

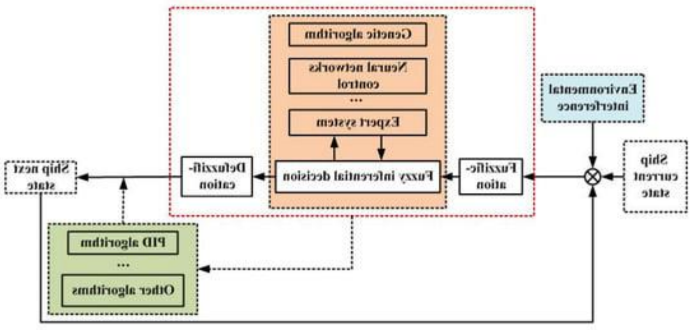


Figure 4: Fuzzy control combined with other algorithms.

These are some of the most common techniques still used to this day, in terms of the first three, they are usually applied in combination with each other or other control algorithms. On the other hand, the latter has been rapidly growing in popularity due to the recent boom of artificial intelligence.

RL is a type of machine learning paradigm where an agent learns to make decisions by interacting with its environment. It is inspired by the process of learning from trial and error, like how a human's or animal's learning process would go.

The fundamental components of RL include the agent, environment, actions, observations (also known as state most of the time), policy, and rewards. The agent is the entity that makes decisions and performs actions. The environment is where the agent works. Actions are the set of possible choices the agent can make, and observations are the perceived states of the environment. The policy is a strategy of possible choices the agent can make, and observations are the agent's perceived states. The reward is the feedback signal that the agent receives after performing an action. The goal of the agent is to maximize the reward.

The goal of this project is to supply an open-source, representative, configurable and realistic environment for the training of agents specialized in the dynamic positioning of maritime vessels.

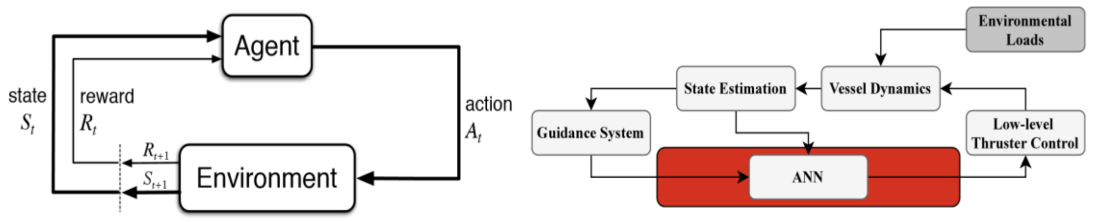


Figure 5: RL control loop VS example DP loop with neural network

CHAPTER 3. DESCRIPTION OF THE SYSTEM

To tackle the creation of our desired environment we first set up to find a framework or simulator that could act as the pillar for further development and adaptation to our needs. Given the complexity, time, and resource constraints of the project the development of our own simulator was considered unfeasible and out of the question.

3.1 SIMULATOR CHOICE

In search of this first base simulator, we set some guidelines on key characteristics we were looking for which were tightly related to the goals we had in mind for our environment. These characteristics were the following:

- **Open Source:** we are looking to avoid any license trouble at the time of parallelizing work and aiming to make the environment fully publicly available for the community to keep a continuous use, testing and development of it.
- **Updated and realistic:** the environment and therefore the simulator must accurately reflect the latest technological advancements and not be outdated or abandoned. It will allow for realistic observations supplying all the necessary features to recreate the scenario at hand.
- **Representative and configurable:** the environment must be standing for a vast variety of cases representative of offshore sea states, offering the possibility of configuring several simulation parameters allowing for a versatile tool for research.
- **Python API:** we are looking to build our environment upon the standard toolkit of Open AI’s Gym package which commonly runs on Python, therefore an interactive and easy to control Python API would greatly benefit this cause.

With all these guidelines in mind and after careful consideration of many simulators, the conclusion we arrived at was that the simulator of choice was Virtual RobotX competition simulator.

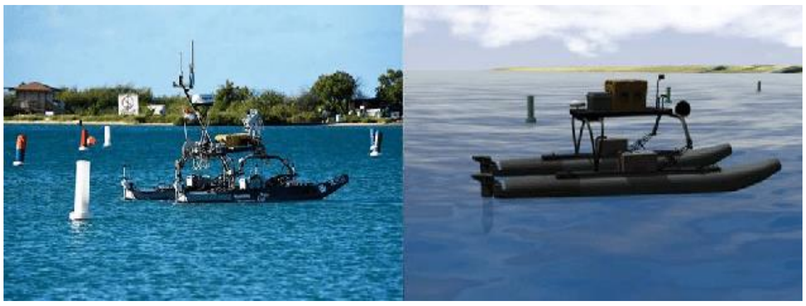


Figure 6: VRX real vs simulation comparison

3.2 SIMULATOR DESCRIPTION

VRX is an innovative, high-fidelity marine robotics simulator. VRX is based on the Gazebo Garden simulation software and uses the Robot Operating System (ROS2) for communication and control, thus receiving help from the wide array of tools and libraries in the ROS ecosystem which will be key in stablishing communication between our environment setup and the simulator. This combination of technologies enables VRX to accurately simulate complex marine environments and the physical characteristics of marine robots.

VRX provides a realistic and dynamic environment with capabilities for simulating and configuring winds and waves and supplies a set of standard tasks often met in marine robotics applications such as station keeping (like DP) waypoint navigation and object detection. VRX also supplies realistic feedback of the vehicle from a variety of sensors like GPS, IMU, Lidar or cameras.

Finally, another reason behind its choice was the sponsorship and support behind the project by renown entities in the maritime and robotics industry such as the Office of Naval Research (ONR), the Naval Postgraduate School (NPS) and Open robotics among others.

3.2.1 ROS2 ECOSYSTEM

Although ROS stands for Robot Operating System, it is not actually an OS, but a set of common libraries and tools on which to build more complex robotic systems. ROS has many distributions, and, in this project, we will be working with ROS2 Humble Python API. Let us do a brief overview on the basics of the ROS ecosystem:

- **Nodes:** they are the fundamental block of the system; they are smaller programs that all run at once and talk to each other. We will use these nodes to define our classes inside the program, some example usage in our system will be a thruster control node for each, a vehicle control node, or a simulation node, among others.
- **Topics and Messages:** it is the way nodes communicate with each other. A topic is a named location that one (or multiple) nodes can publish a message to, these are called publishers. To these topics one (or multiple) nodes can subscribe to receive the messages (subscribers). A node can be both a subscriber of one topic and a publisher of another. We will use these topics to communicate information like the desired angle on a specific thruster, receive the GPS input or the wind's speed.
- **Services:** unlike topics which allow sharing many messages to anyone, services offer a single request/reply communication from one node to another. They supply a way for nodes to perform remote procedure calls (RPCs) and request specific tasks or actions from other nodes. They ease synchronous communication which in the context of RL will be key, for topics we will have to do a little workaround to avoid their asynchronous nature.
- **Launch files:** it is a Python-based system to configure and launch all nodes we need together. We will be using the provided launch scripts by the competition with little adjustments.

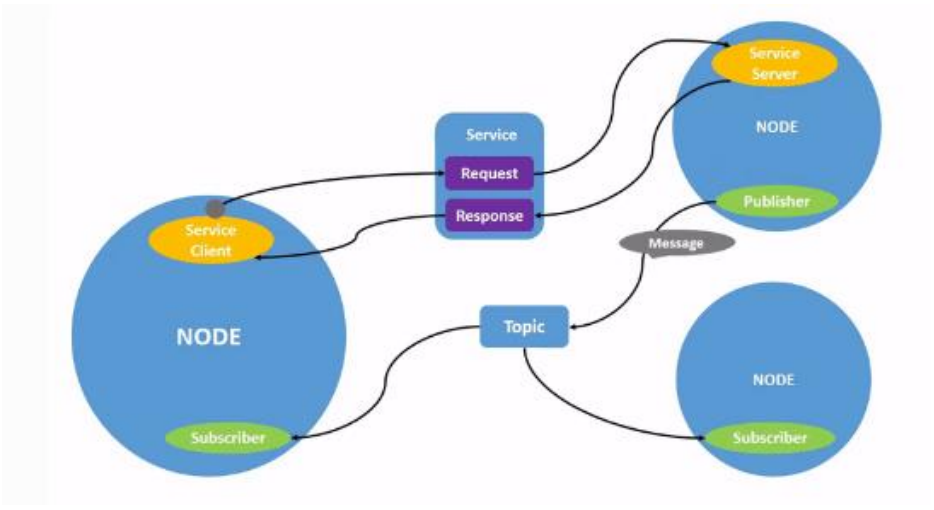


Figure 7: Topics and Services in ROS2

- **Packages and Workspace:** packages are the first level of organization in ROS2, it is how we group closely related files together. All the packages of your project are stored inside the workspace which is the centralized location for organizing and managing them. Developing the whole code as a ROS2 package was considered, however for our purposes it was better to have it stand as a separated Python module.

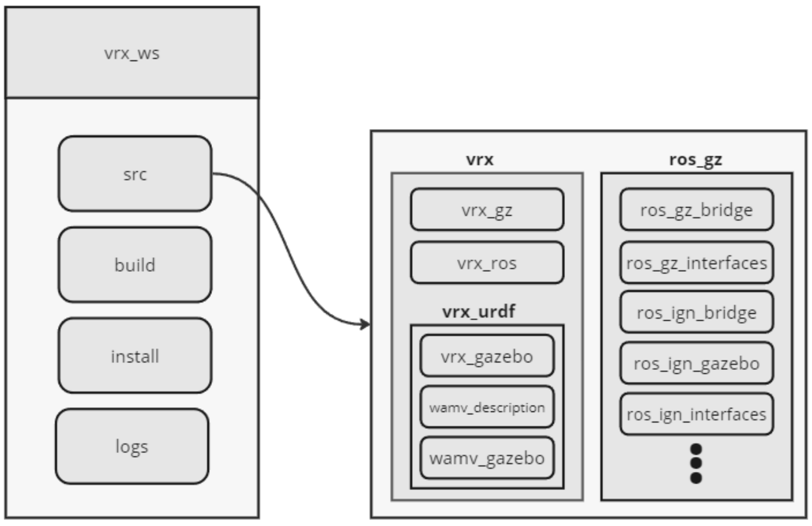


Figure 8: VRX Workspace and main packages

Additionally, ROS supplies a system called tf2 (TransForm version 2), to handle transformations for us. Transformations refer to the mathematical operations and data structures used to represent and manipulate the position and orientation of entities in a 3D space. It allows us to describe relative positions and orientations of various parts of a robot.

To define the structure, geometry, physics, and visual appearance of the robot. It is a standardized way in XML code to describe the physical properties of a robot, such as its links, joints, sensors, and visuals. This file is then read by a ROS node called robot_state_publisher that broadcasts all the transforms, a static value for 'fixed' joints and a new value based on external information for dynamic transforms. It then publishes its contents to the topic /robot_description.

This information in this topic is then shared to Gazebo via the launch file we previously described to allow the robot to be spawned in the simulation and interact with it. Then the information about the robot state in the simulation is broadcasted outside of gazebo through a plugin that publishes the updates to /joint_states which is read by the robot_state_publisher to update the dynamic transforms and repeat the cycle.

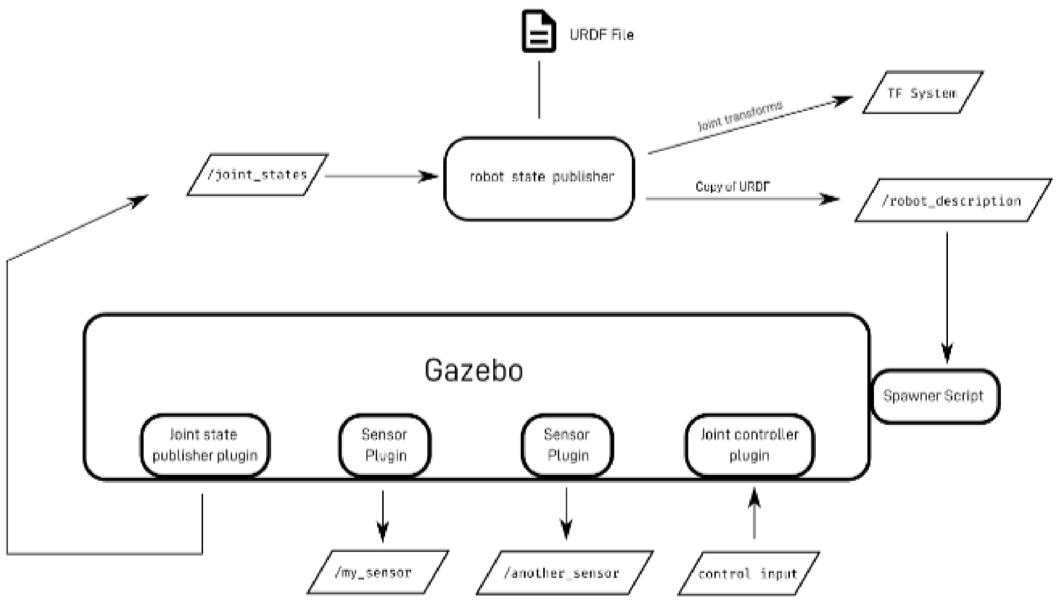


Figure 9: ROS-Gazebo integration

3.2.2 INPUTS

The main inputs available for definition and configuration within VRX that need to be defined and configured beforehand are:

Input variables				
Variable	**Type**	**Dimension**	**Range**	**Units**
Timestep	Continuous	Scalar	-	s
Wave Direction	Continuous	Vector	[0,1]	-
Wave Gain	Continuous	Scalar	-	-
Wave Peak Period	Continuous	Scalar	-	s
Wave Steepness	Continuous	Scalar	[0,1]	-
Wind Direction	Discrete	Scalar	[0,360]	Degrees
Wind Mean Speed	Continuous	Scalar	-	m/s
Wind Var. Gain	Continuous	Scalar	-	m/s
Wind Time Constant	Discrete	Scalar	-	1/s
Added Mass	Continuous	Scalar	-	Kg
Linear Drag	Continuous	Scalar	-	Nm/(rad/s)
Quadratic Drag	Continuous	Scalar	-	Nm/(rad/s)
Yaw Damping	Continuous	Vector	-	N/(m/s)
Max Velocity	Continuous	Scalar	-	m/s

Table 1: VRX main inputs

The above mentioned are only a few considered the most relevant of the many configurable parameters in the simulation. They are main parameters of the own simulation (timestep), sea state (wave and wind) or the vehicle (mass, drag and max velocity).

Besides those pre-configurable parameters, we need to highlight the update rates of the several sensors and topics and the control inputs our agent will be interacting with to drive the vehicle, that are:

Control variables

Variable	**Type**	**Dimension**	**Range**	**Units**
Thruster Thrust	Continuous	Vector	[-1, 1]	N
Thruster Angle	Continuous	Scalar	[-PI, PI]	rad

Table 2: control variables

3.2.3 OUTPUTS

The main outputs available from the simulation that we are going to be reading and working with are:

Output variables				
Variable	**Type**	**Dimension**	**Range**	**Units**
Latitude	Continuous	Scalar	[-90, 90]	Degrees
Longitude	Continuous	Scalar	[-180, 180]	Degrees
Altitude	Continuous	Scalar	-	m
Orientation	Continuous	Quaternion	[-1,1]	-
Angular velocity	Continuous	Vector	-	rad/s
Linear Acceleration	Continuous	Vector	-	m/s²
Wind Speed	Continuous	Scalar	-	m/s

Table 3: VRX main outputs

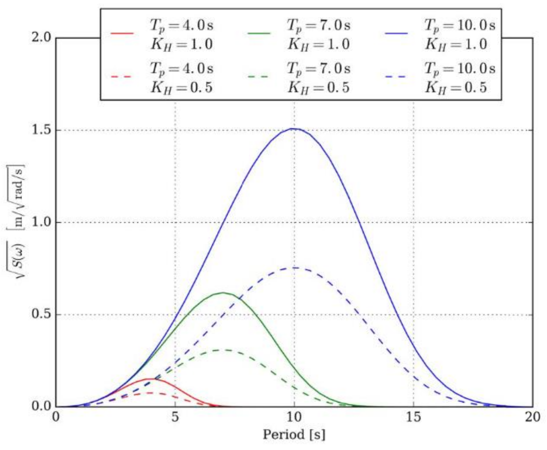
3.2.4 REPRESENTATION AND REALISM

In this section, we will briefly cover how the simulator models the different key components of the environment. This is just going to be a summary, for the explicit details and full explanation please check VRX official paper [BBIN19].

3.2.4.1 Waves

To summarize, waves are modelled using Gerstner waves model. After the wave input parameters described in the Inputs section are set (direction, gain, peak period, and steepness), the directional two-parameter Pierson-Moskowitz spectrum is defined and used to figure out the remaining parameters through sampling a linear deep water dispersion relation.

$$S_B(\omega) = (K_H)^2 \frac{\alpha g^2}{\omega^5} \exp\left[-\frac{5}{4}\left(\frac{\omega_P}{\omega}\right)^4\right]$$



$$\mathbf{x}(\mathbf{x}_0, t) = \mathbf{x}_0 - \sum_{i=1}^{N} q_i(\mathbf{k}_i/k_i)A_i \sin(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t + \phi_i) \tag{1a}$$

$$\zeta(\mathbf{x}_0, t) = \sum_{i=1}^{N} A_i \cos(\mathbf{k}_i \cdot \mathbf{x}_0 - \omega_i t + \phi_i), \tag{1b}$$

Equation 1: Gerstner Waves

Figure 10: Two parameter Pierson-Moskowitz wave spectrum

3.2.4.2 Wind

Total wind speed consists of the sum of the constant mean wind speed and the temporally varying, zero-mean, variable wind speed:

$$V_w(t) = \bar{v} + v_g(t)$$

Equation 2: Wind formulation

The variable part of the wind speed is modeled as a first-order, linear approximation of the Harris Spectrum with its time constant defined by the user as shown in the Inputs table. The spectrum is expressed as the following transfer function:

$$h(s) = \frac{K_w}{1 + \tau_g s}$$

Equation 3: Harris Spectrum transfer function

3.2.4.3 Vehicle

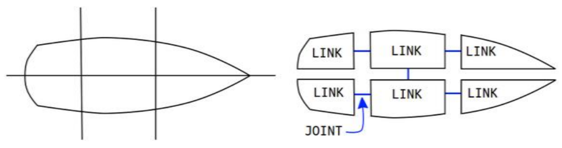
- Wave forces

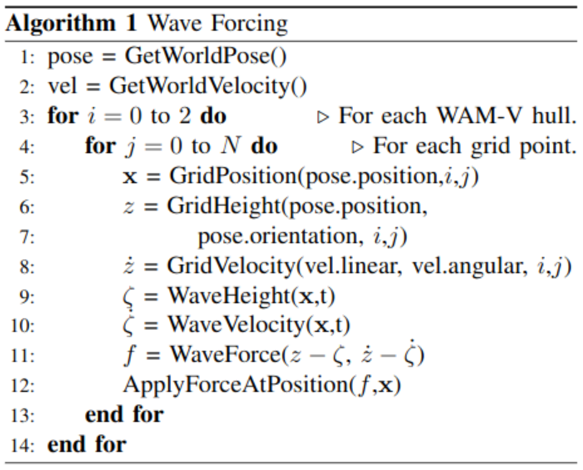
  To approximate the motion of a vehicle in an ocean environment, Fossen's six degee-of-freedom robot-like vectorial model for marine craft is adapted:

$$\underbrace{\boldsymbol{M}_{RB}\dot{\boldsymbol{\nu}} + \boldsymbol{C}_{RB}(\boldsymbol{\nu})\boldsymbol{\nu}}_{\text{rigid body forces}} + \underbrace{\boldsymbol{M}_{A}\dot{\boldsymbol{\nu}}_r + \boldsymbol{C}_{A}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r + \boldsymbol{D}(\boldsymbol{\nu}_r)\boldsymbol{\nu}_r}_{\text{hydrodynamic forces}} + \underbrace{\boldsymbol{g}(\boldsymbol{\eta})}_{\text{hydrostatic forces}} = \boldsymbol{\tau}_{propulsion} + \boldsymbol{\tau}_{wind} + \boldsymbol{\tau}_{waves}$$

Equation 4: Fossen's 6 DoF model

Added mass and Non-linear terms added in the maneuvering model but neglected for complexity in the seakeeping, a simplified model is used preserving the right type of vessel response.





Algorithm 1 Wave Forcing

```
pose = GetWorldPose()
vel = GetWorldVelocity()
for $i = 0$ to 2 do                              ▷ For each WAM-V hull.
    for $j = 0$ to $N$ do                        ▷ For each grid point.
        x = GridPosition(pose.position,$i$,$j$)
        $z$ = GridHeight(pose.position,
                pose.orientation, $i$,$j$)
        $\dot{z}$ = GridVelocity(vel.linear, vel.angular, $i$,$j$)
        $\zeta$ = WaveHeight(x,t)
        $\dot{\zeta}$ = WaveVelocity(x,t)
        $f$ = WaveForce($z - \zeta$, $\dot{z} - \dot{\zeta}$)
        ApplyForceAtPosition($f$,x)
    end for
end for
```

Equation 5: Wave Forcing Algorithm

- Wind Forces

The forcing terms are expressed as follows, where $\bar{c}_x$, $\bar{c}_y$and $\bar{c}_n$ are the dimensional wind coefficients and the remaining terms ae the simulated (apparent) wind velocities:

$$\begin{aligned} X_{wind} &= \bar{c}_x u_{rw} |u_{rw}| \\ Y_{wind} &= \bar{c}_y v_{rw} |v_{rw}| \\ N_{wind} &= -2.0 \bar{c}_n u_{rw} v_{rw} \end{aligned}$$

Equation 6: Wind forcing terms

- Propulsion forces

The characteristics of an individual thruster are specified by a user-defined relationship between commanded effort and the resulting thrust force. Two static mappings are included. The resulting max thrust command (F) from a thruster is calculated using max speed term (v) defined by the user and linear (C_1) and quadratic (C_2) drag terms:

$$F = 0.5 * ((C_1 + C_2 * v) * v)$$

Equation 7: Max thrust command

CHAPTER 4. RESULTS

The resulting environment was thought for a distributed use case where many training instances are raised each with their own agent based on a copy of our system with the following schema:

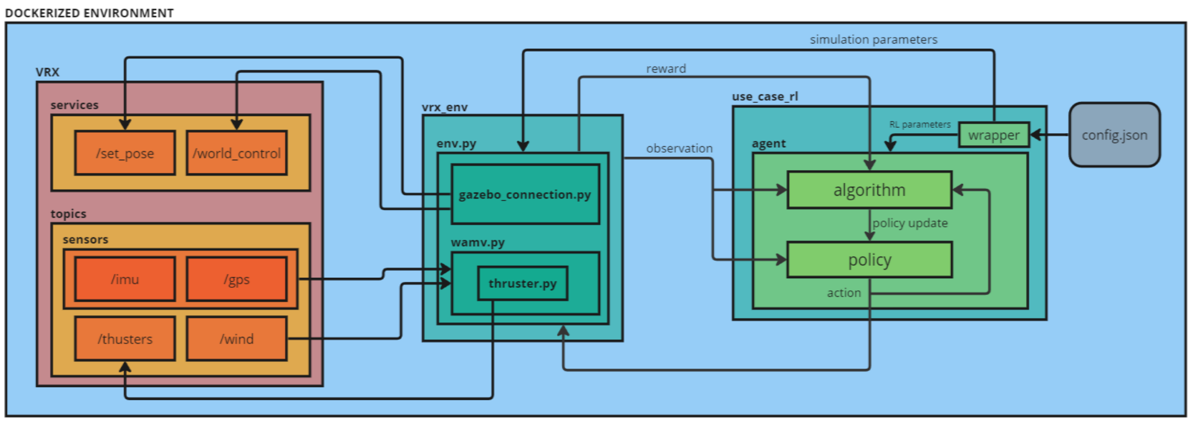


Figure 11: Final distributed system schema

4.1 VRX

Minor tweaks needed to be done to the base code of the simulator to allow RL training, however by far the most important one was the activation of the world control functions. We needed to be able to pause, resume, reset and advance the simulation by steps at a constant rate and despite gazebo coming with world control service requests, these functions were unavailable at the beginning.

The problem was that the world control function was displayed when listing gazebo services, but it was uncallable from ROS2 and therefore its Python API, making it unreachable for our purposes. The solution was obvious, if we managed to set up a bridge between ROS2 and Gazebo we would be able to access the controls. A bridge is a software layer that allows ROS2 nodes to communicate with Gazebo's environment. In this case we bridged the world plugins to world service.

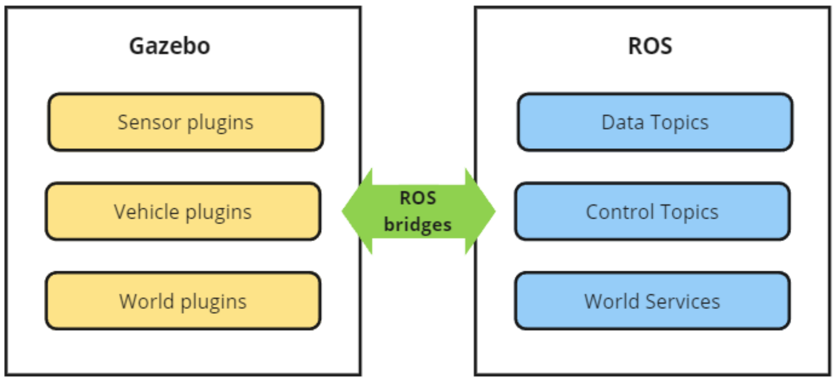


Figure 12: ROS bridges

After bridging the world control, we must add it to the launch file and then it is ready to use by service requests. After some testing it was noticed that the reset function was not working properly, debugging this could take a long time since it would require digging deeper in the source code. Fortunately, for our purposes we could make a workaround the reset function, which we will comment further below, by enabling another of gazebo's function: set pose. So, we bridged this one too. To do these modifications, we had to install ros_gz package by directly cloning its GitHub instead of by apt.

The latest modification to the source code made had to do with the wind plugin (USVWind.cc), It was causing the simulator to crash when we interacted more than once with the world control service giving out a division by zero. The problem lied in the lack of an escape route in the plugin in case the simulation was stopped, since the plugin used delta time in many of its calculations and when the simulation was stopped it was drawing nan value (zero).

In terms of the world used for training, we used the default sydney_regatta.sdf file, however, we removed unnecessary elements like the land, buoys, tents, or antennas to name a few. Further acceleration could be achieved by removing sensors we are not planning on using (which are not for DP) like lidar or cameras.

Finally, we had to edit the simulation parameters related to the step, specifically the world max step size which was set to 0.005s and the topics update rate which refers to the publishing frequency. The final values of the update rate changed to match the step size and considering sensor's limitations were:

Topic	**Update Rate (Hz)**
/gps	20
/imu	100
/wind	20

Table 4: Topic's update rates

4.2 *USE_CASE_RL*

Before digging deep into the environment definition, let us review this block which covers mainly the setting of the RL parameters and training, it is in an early stage and thus only covers a simple example, training locally with a set PPO algorithm using GPU in a centralized way. It consists basically of a main file which pulls the parameters from a config.json and sets up the training and environment with those parameters.

In terms of the definition of the RL parameters our environment the main characteristics are:

- **Actions:**
  - <u>Thruster Angle</u>: To try to ease the neural network's job we are going to do some encoding to try and solve the discontinuity issue in the angles. The issue is mainly that independently of the way you define the limits, the net will

always find two angles equivalent despite them being utterly different numbers, with no continuity (0 and 2PI or PI and -PI depending how you define the action space) which could presumably cause issues in the neural network reasoning. We will be defining its action space from -PI to PI.

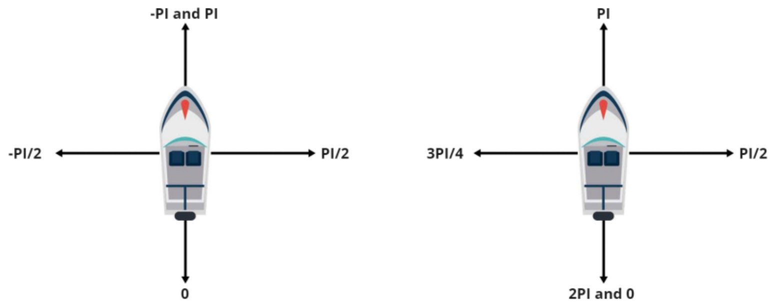


Figure 13: Discontinuity issue representation in 1D

To work around this, the angle's action is going to be split in two, its sine and cosine; however, this might lead to impossible combinations, giving us results that invalidate $\cos^2 + sen^2 = 1$. To prevent this, when decoding the angle using the arctan function, we are going to first transform them by dividing by the module of the vector (sen, cos) thus supplying a combination that fulfills the condition. Another alternative would be to use Euler angles [CZZC20], or combine the capabilities of reverse thrust with only a half the angle circle, forcing the neural net to interpret that to use the missing half it needs to change from forward to reverse.

- <u>Thruster Thruster</u>: since neural networks work better with small, scaled input variables, we are going to limit its action space from -1 (full reverse thrust) to 1 (full forward thrust) and then decode it by multiplying by the max thrust command available. [BJHU20]

- **Observations:** as for the observations we are going to start by using the goal position we aim to keep (lat_init, lon_init), GPS data (lat, lon), the IMU data (orientation quaternion, linear acceleration, angular velocity) and the wind speed and direction. We add the goal pose because, although the agent in training could infer the relationship with its current position from the reward when it is deployed it will not count with a reward. Wind direction could help us in the future if we add an anemometer sensor to measure wind relative direction to the vessel direction. We have not added altitude not anything related to the z axis since it escapes the scope of the DP control.

- **Reward:** as for the reward we are simply going to be converting both the goal pose and the current pose to local cartesian and calculating the module of each of their vectors, then the difference between their values will be the negative reward. We will try to scale this reward to make it representative of the separation with the goal.
- **Done Condition:** for now, we will only consider an episode done when it reaches its maximum number of steps.

4.3 VRX_ENV

vrx_env is the core of the system, it sets up the interactions with the environment supplying the Python code to take all actions necessary in a RL experiment. Let us go over its components from bottom to up:

- **thruster.py:** this module creates a class defining the behavior of a thruster. It inherits the node class and publishes to the thrust and angle right topics. It uses class decorators to avoid unnecessary latency in the simulation when a thruster state does not change, a new value of thrust or angle will be published only when it differs from the earlier one. This module will be inherited by the wamv.py and initialized for each thruster in the ship.
- **wamv.py:** the wamv module defines the behavior and data inputs of the vehicle. It inherits the node class, initializes left and right thrusters using the above defined thruster class and subscribes to the gps, imu, wind speed and wind direction topics. A spin_until_callbacks method is defined to fight the asynchronous nature in ROS2 message passing which generated latency in between a step is executed and the new data is published to the topic causing discrepancies. The function keeps the node

spinning thus preventing the advance to the next step until all callback's flags are marked done showing the new data has been successfully received. A _check_topics_started method is defined to block the advance of the program in the initialization until the topics needed have been raised. This fixed a recurrent bug where the code would enter the spin_until_callbacks_done method and would not escape it.

- **gazebo_connection.py:** this module inherits the node class and sets up the connection with the gazebo simulator to enable its control. It creates a client to the world control and set pose services with a timeout limit of 10 seconds which will cause the program to be stopped in case these services are not found, be it because the launch is faulty, or the program hasn't been run at all. Once the connection to world control is made, the first thing done is pausing the simulation. Three methods are defined, one to use the three functionalities available of world control (pause, step and multistep, resume would be just sending a false pause), another for set pose which will then be used in the third and last method to reset the wamv's pose.
- **env.py:** the most important script out of four, it encapsulates the three already defined and aims to achieve our goal of turning the vrx into a trainable Open AI's gym environment. It inherits the node class, and we defined an added node for logging and debugging purposes. It has the typical Gym methods, reset, step, and close (render is not truly defined as a method since it would only involve running the simulation with headless parameter). The input parameters are:
    - max_steps: max number of steps per episode.
    - start_pose: first position and orientation of our vehicle.
    - dt: the timestep of the simulation, if adjusted we must remember to adjust the sensor's update rate accordingly to match it and always have at least one read at the end of each step, if not a warning will be raised.
    - world: name of the world to run.
    - headless: Boolean value that controls if the simulation is run without or with gui, a false value would be equivalent to rendering the environment, however, keep in mind that doing so would slow the simulation down greatly.

- Wave parameters: direction, gain, peak period, and steepness, already described in the inputs table.
- Wind parameters: direction, mean velocity, variable gain, and time constant, described in the inputs table.
- thruster_path: file path where the thruster configuration file is kept, it will help us pull thruster parameters and calculate max thrust command.
- scale: scale to apply to the reward

After storing these inputs in attributes, using an auxiliar method to pull the thruster max thruster command, we run the launch method, initialize the gazebo connection and wamv nodes and take an initial step (gazebo_connection step not the proper method of the env class, so a 'simulation' step not a 'training' one) that will load a position value in the gps topic to help us track the initial position of the wamv and thus the goal pose.

Let us briefly cover the main methods of the environment class and how they have been implemented:

- launch: first it kills any existing simulation by finding its process id, then it sets the different input parameters (timestep, wave and wind) and finally starts a new simulation running a subprocess (a gnome terminal was the first approach however given the lack of gui working remotely with docker it was later removed). The setting of the parameters is done by directly changing the world file found at the install folder parsing its xml content.
- step: this method runs one timestep of the environment dynamics. First, it takes the action to be performed as input, and sets the action by performing the already explained decoding of the sine and cosine. Then it takes one gazebo step and obtains the resulting observation, checks if the episode is done, calculates the reward and returns the necessary variables of a gym step (observation, reward, done and info).

- reset: as it stands right now the reset method only reinitializes variables and returns the WAMV to its original position. Ideally, in the long run the idea would be to figure out the reason behind the world control reset malfunction, fix it, and use it. Soon, a better approach would be to use a world consisting of only water, defining a grid of coordinates of where the wamv can spawn and pulling a random value from this grid each time the simulation is restarted.

4.4 HARDWARE AND DEPLOYMENT

In terms of hardware, the VRX simulator is designed to be run under the following minimum system requirements:

- Modern multi-core CPU, e.g., Intel Core i5
- 8 Gb of RAM
- Nvidia Graphics Card, e.g., Nvidia GTX 650
- Ubuntu Desktop 22.04 Jammy (64-bit)

For MacOS we did not go ahead with any installation. As for Windows OS the installation was done successfully using WSL and Docker, but we did not to achieve hardware acceleration through the GPU thus getting a simulation speed way below real time. The assumption was that the issue was lying in compatibility issues between d3d12 mesa driver used for hardware acceleration in WSLg does not implement a base GL function used by the Ogre engine of Gazebo, thus development in this front was stopped.

As for the setup for the training, we have gone ahead with a basic core training in our host machine, however the idea would be to run distributed training running multiple instances of the container each with its own agent therefore using distributed learning. Each agent learns independently from its own observation, but the overall policy would receive help from the experiences of each other [WGDR99].

Another valid approach to be considered would be centralized learning, having only one agent receiving multiple observations from multiples training instances raised and learning from all of these, however, it can be harder to set up and more computationally expensive.

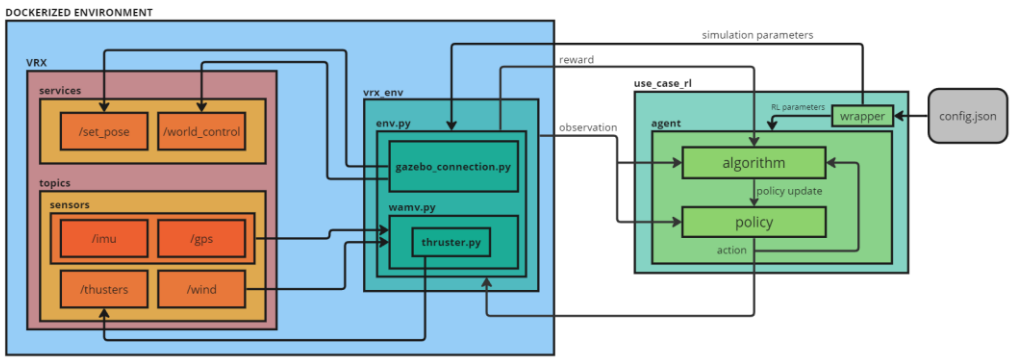


Figure 14: Central system proposal

CHAPTER 5. CONCLUSIONS

Overall, I would qualify the results achieved as a success. We managed to achieve the most important thing which was generating a trainable environment based on the VRX simulator which is open-source, configurable, realistic, representative, and most important of all compatible with OpenAI’s Gym wrapper.

We successfully migrated the simulator control to our Python API, supplying the possibility of editing oceanic parameters, simulation time step and update rates. We presented the possibilities of the simulator to handle the input of both new vessels defined through the several URDF and configuration files and new worlds.

There is still much room for improvement; the simulation could be sped up beyond the real-time factor of 3 we achieved by removing unnecessary elements in the world files or optimizing GPU usage. We only considered the most basic use case having the wavefield parameters fixed for a given training and we did not have the time to explore and dig deeper into different training setups.

Despite the above, the most relevant achievement was still carried out, which was the proposal and development of an early-stage platform to carry out reinforcement learning training of agents for the task of maritime dynamic positioning.

CHAPTER 6. BIBLIOGRAPHY

[ABBE18] *"ABB to Equip High-Spec Service Operations Vessel for the World's Largest Offshore Wind Farm." ABB Group. Leading digital technologies for industry - ABB Group, May 30, 2018. https://new.abb.com/news/detail/51620/abb-to-equip-high-spec-service-operations-vessel-for-the-worlds-largest-offshore-wind-farm*

[OHUS] "Operations Home." United States Coast Guard (USCG). Accessed June 21, 2023. https://www.dco.uscg.mil/OCSNCOE/DP/Overview/

[IDPS] Intro to Dynamic Positioning (DP) systems - united states coast guard. Accessed June 21, 2023. https://www.dco.uscg.mil/Portals/9/OCSNCOE/References/Custom-Ref-Books/Intro-to-DP-Systems-Dec2019.pdf?ver=d1Z9tUwX9p__Mi05A_NkwA%3d%3d

[DPWM] "Dynamic Positioning." Wikipedia, May 2, 2023. https://en.wikipedia.org/wiki/Dynamic_positioning

[WLQW19] Wang, Le, Qing Wu, Jialun Liu, Shijie Li, and Rudy R. Negenborn. 2019. "State-of-the-Art Research on Motion Control of Maritime Autonomous Surface Ships" Journal of Marine Science and Engineering 7, no. 12: 438. https://doi.org/10.3390/jmse7120438

[OVER21] Øvereng, Simen Sem, Dong Trong Nguyen, and Geir Hamre. "Dynamic Positioning using Deep Reinforcement Learning." Ocean Engineering 235 (2021): 109433. https://doi.org/10.1016/j.oceaneng.2021.109433

[BBIN19] B. Bingham et al., "Toward Maritime Robotic Simulation in Gazebo," OCEANS 2019 MTS/IEEE SEATTLE, Seattle, WA, USA, 2019, pp. 1-10, doi: https://doi.org/10.23919/OCEANS40490.2019.8962724

[UNRO] "Understanding Nodes." Understanding nodes - ROS 2 Documentation: Humble documentation. Accessed June 22, 2023. https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Nodes/Understanding-ROS2-Nodes.html

[JGRR21] Josh. "Getting Ready for Ros Part 8: Simulating with Gazebo." Articulated Robotics, November 14, 2021. https://articulatedrobotics.xyz/ready-for-ros-8-gazebo/

[CZZC20] Cao, Zhiwen, Zongcheng Chu, Dongfang Liu, and Yingjie Chen. "A Vector-Based Representation to Enhance Head Pose Estimation." arXiv.org, December 8, 2020. https://arxiv.org/abs/2010.07184

[BJHU20] Brownlee, Jason. "How to Use Data Scaling Improve Deep Learning Model Stability and Performance." MachineLearningMastery.com, August 25, 2020. https://machinelearningmastery.com/how-to-improve-neural-network-stability-and-modeling-performance-with-data-scaling/

[WGDR99] Weiβ, Gerhard. "Distributed Reinforcement Learning." Robotics and Autonomous Systems, December 22, 1999. https://www.sciencedirect.com/science/article/abs/pii/092188909500018B